



并行计算基础实验介绍

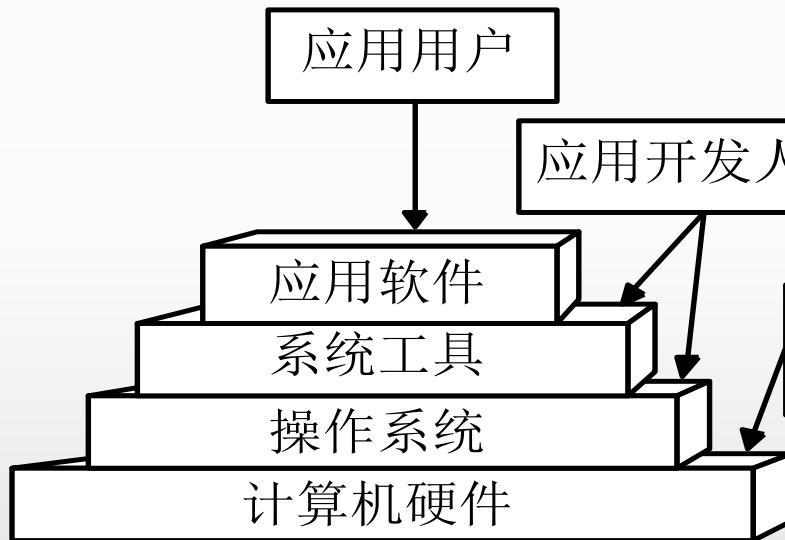


提 纲

- 计算机系统层次和操作系统
- 实验环境配置
- 编译工具链
- Linux常用命令演示
- 性能优化工具

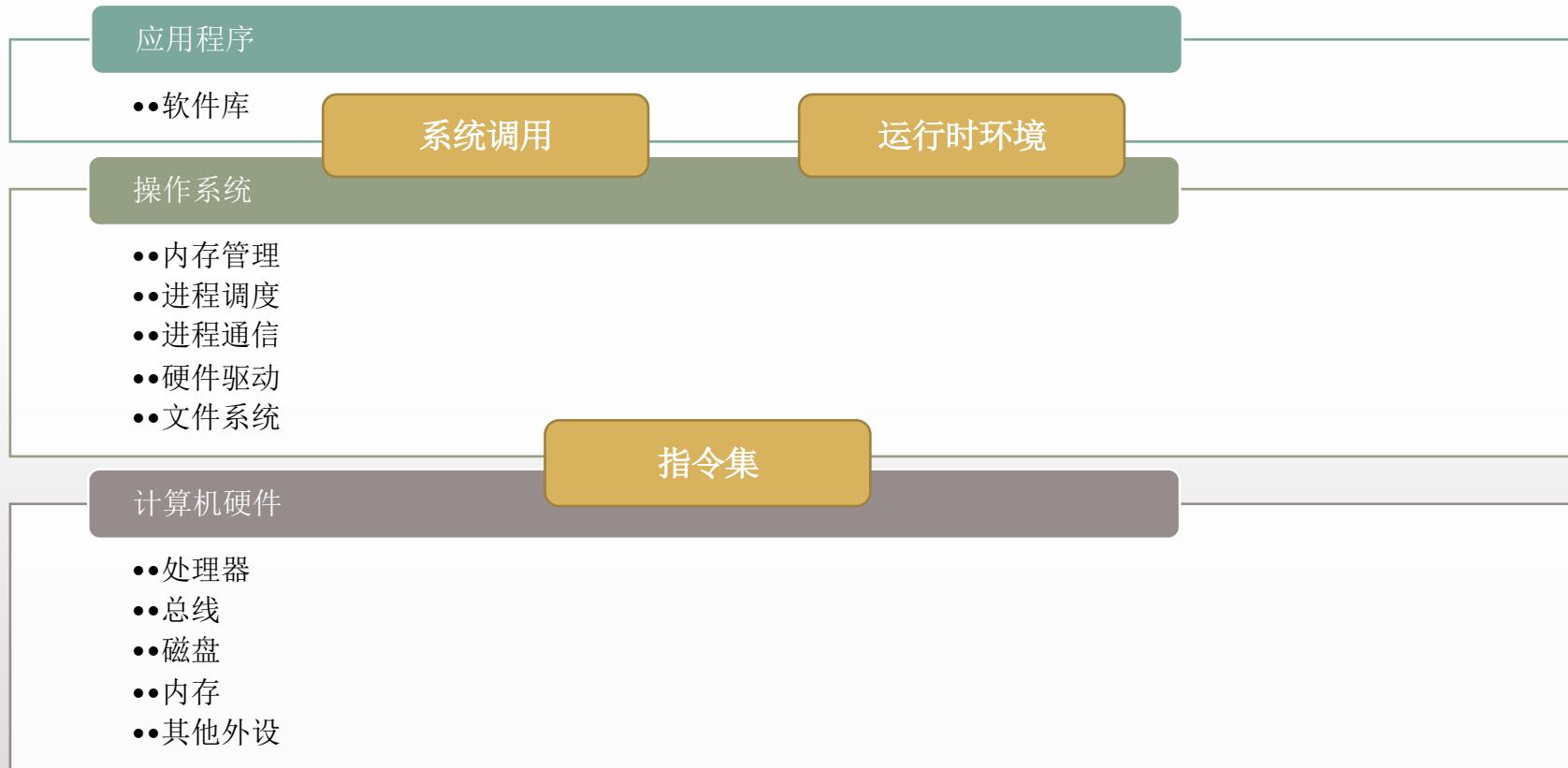


计算机系统层次



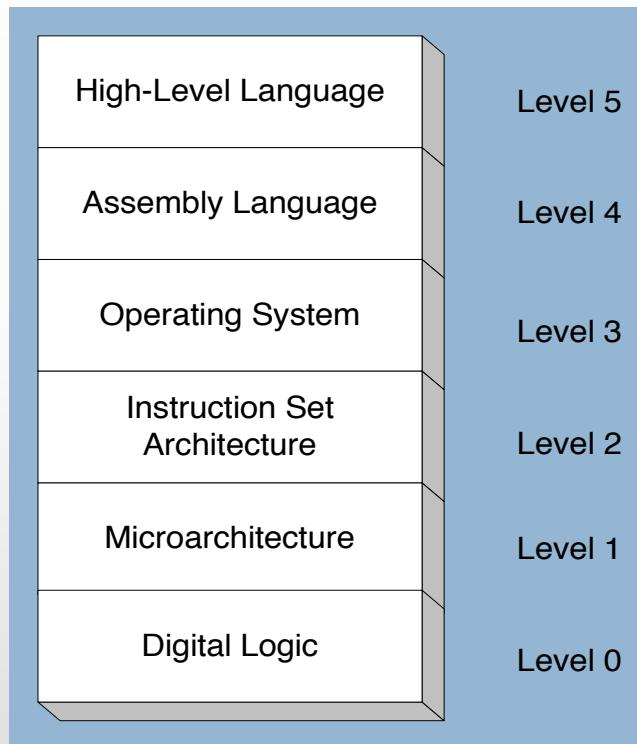


计算机系统层次





计算机语言



English: Display the sum of A times B plus C.

C++: cout << (A * B + C);

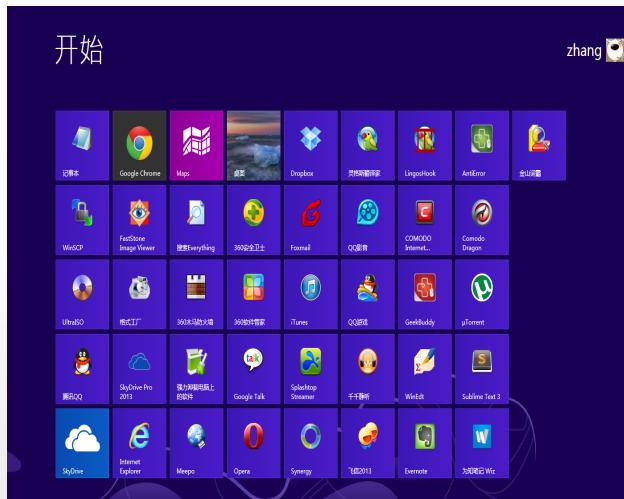
Assembly Language:
mov eax,A
mul B
add eax,C
call WriteInt

Intel Machine Language:
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000

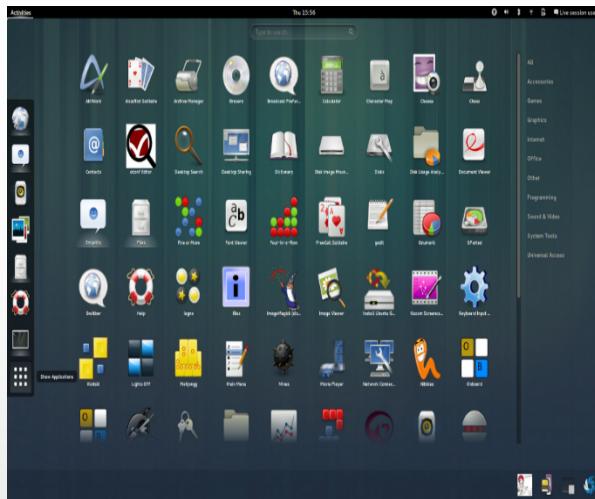


The World of OS (operating system)

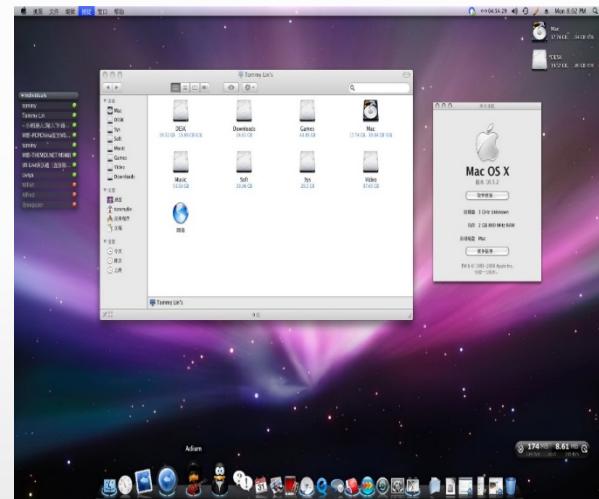
Windows



Ubuntu(Linux)



Mac OS



那么问题来了：为什么supercomputers, 服务器等都要使用linux系统？？

Linux提供了一个完整的操作系统当中最底层的硬件控制与资源管理的完整架构，其稳定与完善的资源分配功能，可以支持大型应用程序（如地球系统模式等）又快又稳的运行。



Linux的历史

Linux创始人: Linus Torvalds (林纳斯·托瓦兹)

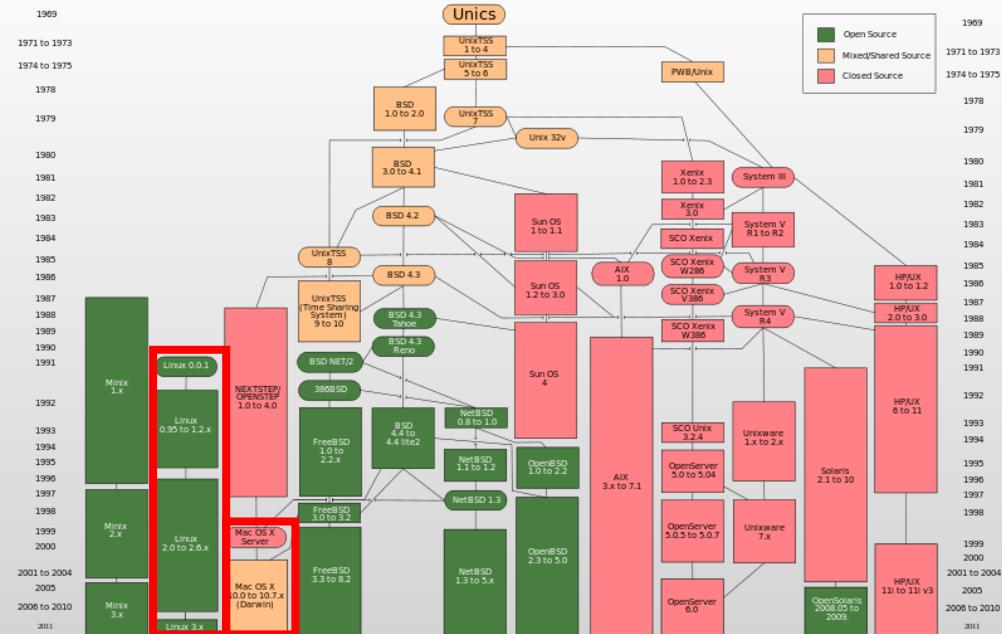
Linux官方生日: 1991年的10月5日



Linux的前辈: UNIX操作系统 (1973)

Linux常见桌面发行版本: RedHat、Debian, Ubuntu, Fedora, Gentoo

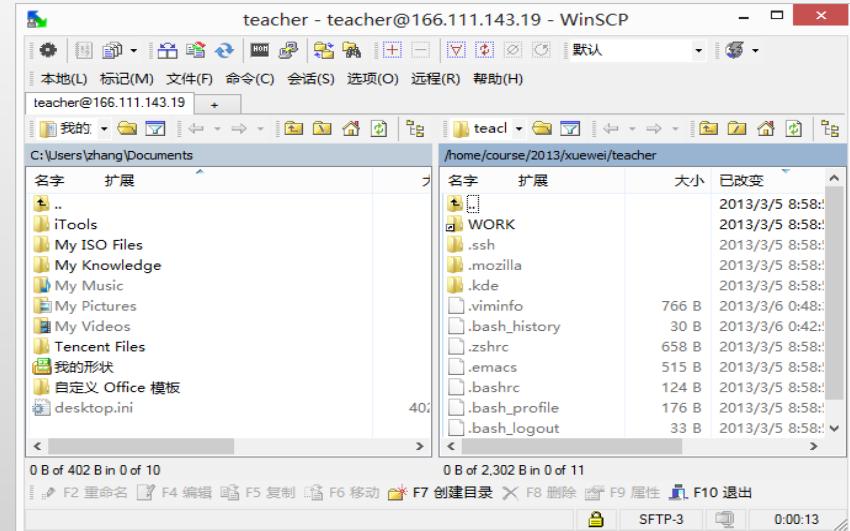
Linux的五大支柱: Linux操作系统的诞生、发展和成长过程始终依赖着以下五个重要支柱: UNIX操作系统、MINIX操作系统、GNU计划、POSIX标准和Internet网络。





实验环境

- 请用ssh协议登陆，用putty或者Secure SSH或相应客户端软件:username@IP
- IP: 待定
- 在各自目录下进行测试和作业
 - 及时在本机做好备份。
- 编译链接和小规模练习和测试使用虚拟机
- 与本机通信使用WinSCP





虚拟机使用

Oracle virtualbox虚拟机实例course_vm

- 占用<20GB空间
- Intel compiler, 支持OpenMP
- Intel mpi
- Intel Vtune

基于VirtualBox-4.3.2-90405-Win

- 需要安装Oracle_VM_VirtualBox_Extension_Pack-4.3.2-90405
方法：管理—全局设定—扩展
- 虚拟机内外交换数据：先在virtualbox中设置共享文件夹。
例如F:/sharing；再在虚拟机系统中挂载; mount -t vboxsf sharing /mnt/share

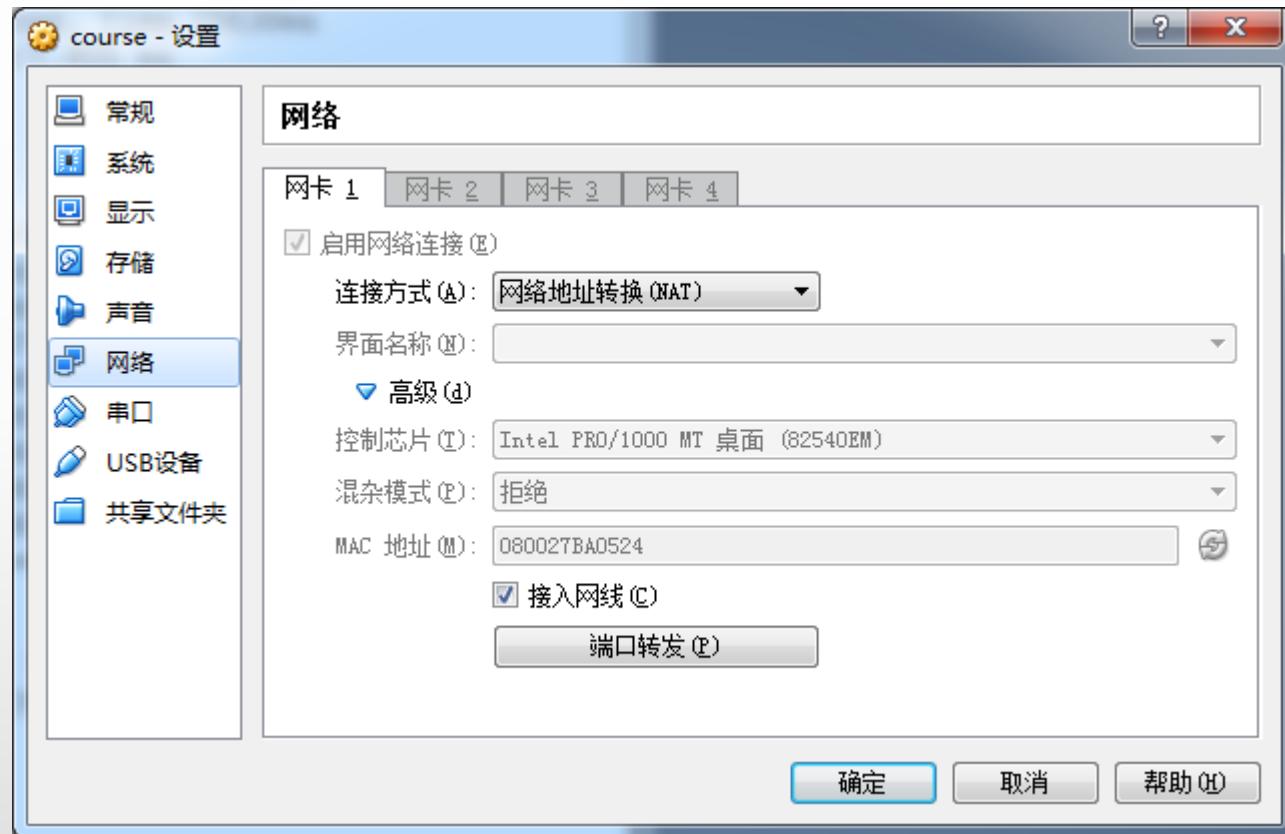
Linux虚拟机：

root: parallel

用户thuhpc: 123456



网络配置





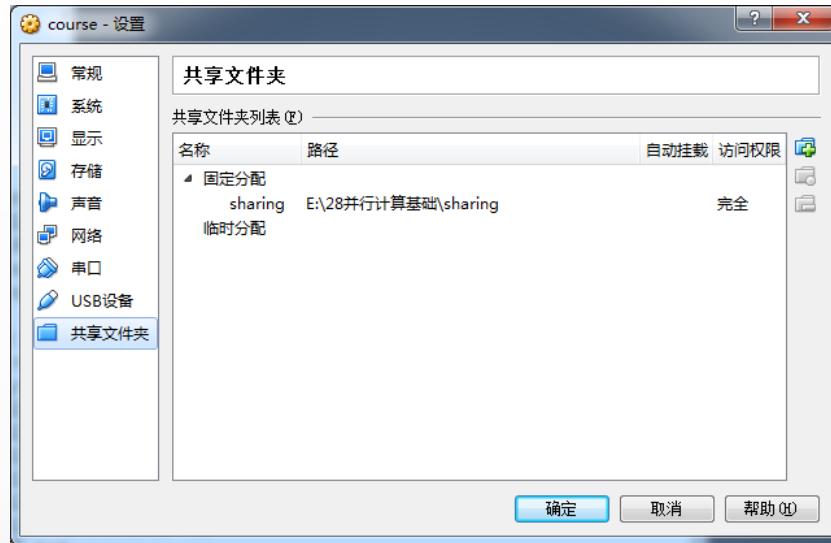
虚拟机与本机通信配置

一、共享文件

1、设置共享文件夹：sharing

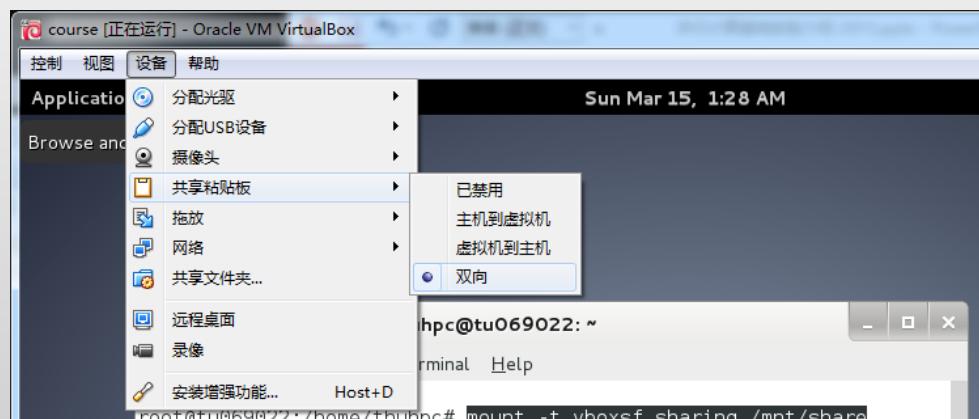
2、虚拟机中挂载

`mount -t vboxsf sharing /mnt/share`



二、虚拟机内外复制粘贴

设备----共享粘贴板----双向



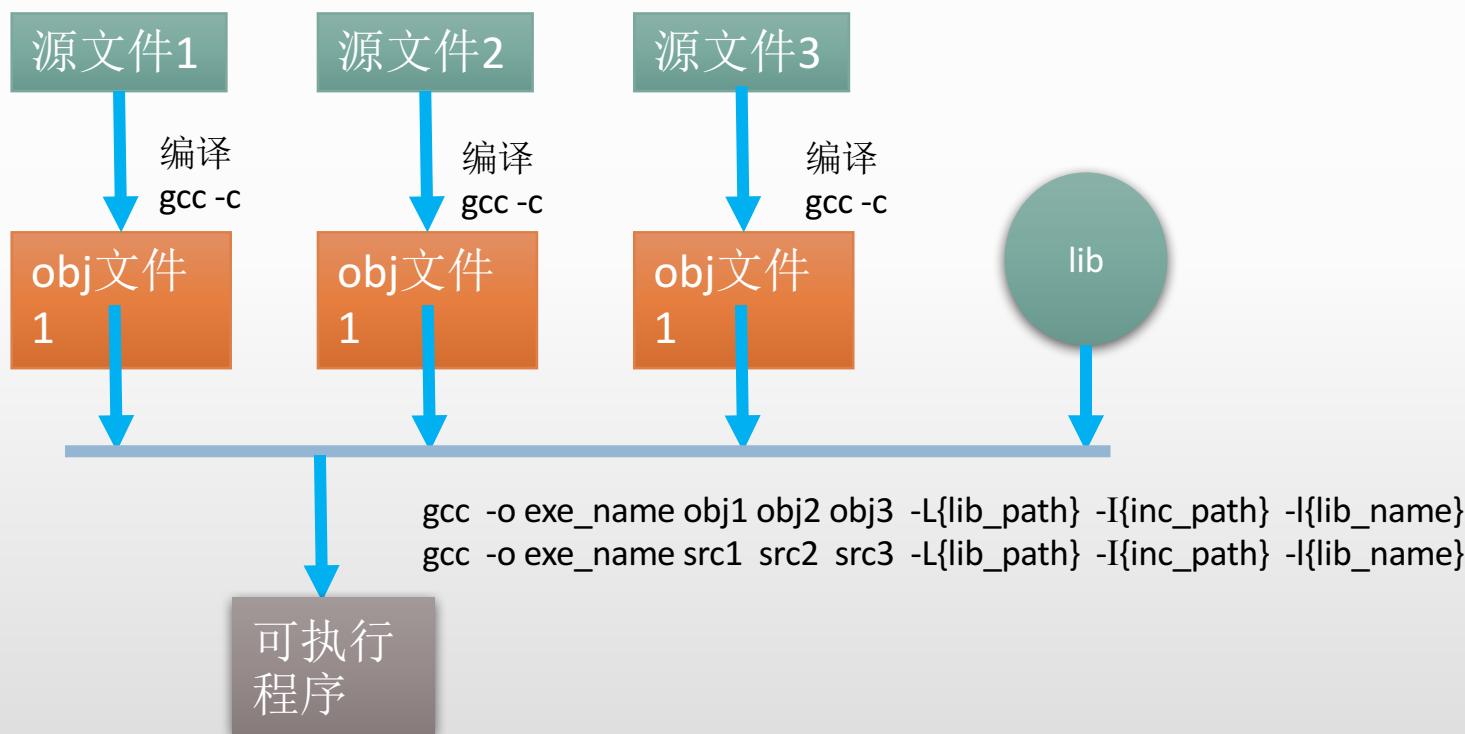


工具链

- 编辑器 (editor): vim, emacs
- 编译器 (compiler) 和链接器 (linker): gcc, icc, mpicc, mpiicc
- 调试器 (debugger) : gdb, Totalview, “print”
- 编译系统: makefile
- 性能测试工具
 - Intel® Vtune
 - gprof
 - mpiP



源码编译过程





服务器基本使用环境配置

- which mpiicc 查看mpi路径
- Which icc 查看intel编译器路径

- vi ~/.bashrc
- source /opt/intel/composer_xe_2015/bin/compilervars.sh intel64
- source /opt/intel/impi/5.0.2.044/bin64/mpivars.sh
- source ~/.bashrc

- 编译: mpiicc example.c -o example
- 启动mpd守护进程: mpdboot
- 执行: mpiexec -n 进程数 ./example/example

- 性能工具: /home/dingnan/example/softwar



Vi

- 模式
 - 普通模式
 - 编辑模式
 - 命令模式
 - 可视模式



普通模式

- 移动光标: h/j/k/l, Ctrl-F, Ctrl-B, ^, \$, w
- x、d\$、dd——删除一个字符、删除光标所在处到行尾的所有字符以及删除整行的命令
- yy 复制一行, p粘贴
- 进入命令模式: : (冒号)
- 进入编辑模式: i/I/a/A/o/O/s/S, Ctrl+p 补全
- 进入可视模式: v
- 进入查找状态: / 或者 ?



命令模式

- 替换: 类似Sed中的替换
- 文件读写: w, wq, q!, x
- 撤销: u
- 行号: set nonu; set nu
- 退出命令模式至普通模式: Esc



可视模式

- 进入可视模式 (visual mode): 普通模式+v
- 作用:
 - 选取文本
 - 配合拷贝/删除操作: y/d
- 退出此模式: Esc



Linux常用命令

- 系统相关 uname hostname free cpuinfo
- 文件系统 ls mkdir mv rm cp ln du df tail
- 进程管理 top ps kill
- 时间相关 date time
- 远程访问 ping ssh
- 文件压缩 tar bzip2 gzip
- 其它工具 man grep diff screen ...
- 用户和权限管理: w who whoami chmod



压缩与解压缩

- 打包+解包: tar
- 压缩: gzip, bzip2
- 打包+压缩
 - tar -c jvf newtar.tar.bz2 files_list
 - tar -czvf newtar.tgz files_list
- 解压缩+展开
 - tar -x jvf newtar.tar.bz2
 - tar -xzvf newtar.tgz



Makefile

```
1 CC =gcc
2 OPT=-O3
3 #CFLAGS = -DDEBUG -I/cm/shared/apps/intel-cluster-tools/impi/3.2.0.011/include64
4 CFLAGS = -I/cm/shared/apps/intel-cluster-tools/impi/3.2.0.011/include64
5 DEF= -DPERF_PROFILE -DATM -DLND -DICE -DOCN
6 AR = ar
7 ARTAG = rcv
8
9 libmpit : profile.o timer.o mpiwrapper.o mpiwrapper_f.o
10      $(AR) $(ARTAG) libmpit.a mpiwrapper.o mpiwrapper_f.o timer.o profile.o
11      $(RM) -f *.o
12 profile.o : profile.c
13      $(CC) $(CFLAGS) $(DEF) -c profile.c
14 mpiwrapper.o: mpiwrapper.c
15      $(CC) $(CFLAGS) $(DEF) -c mpiwrapper.c
16 mpiwrapper_f.o: mpiwrapper_f.c
17      $(CC) $(CFLAGS) $(DEF) -c mpiwrapper_f.c
18 timer.o : timer.c
19      $(CC) $(CFLAGS) -c timer.c
20 clean :
21      rm -f *.o      rm -f libmpit.a
~
```



查看机器状态

- top:
- sar
- vmstat
- ps
- iostat

Top命令

Linux下常用的性能分析工具，能够实时显示系统中各个进程的资源占用状况。



• 序号	列名	含义
• a	PID	进程id
• H	PR	优先级
• i	NI	nice值。负值表示高优先级，正值表示低优先级
• m	TIME+	进程使用的CPU时间总计，单位1/100秒
• n	%MEM	进程使用的物理内存百分比
• o	VIRT	进程使用的虚拟内存总量，单位kb。VIRT=SWAP+RES
• p	SWAP	进程使用的虚拟内存中，被换出的大小，单位kb。
• q	RES	进程使用的、未被换出的物理内存大小，单位kb。RES=CODE+DATA
• r	CODE	可执行代码占用的物理内存大小，单位kb
• s	DATA	可执行代码以外的部分(数据段+栈)占用的物理内存大小，单位kb
• t	SHR	共享内存大小，单位kb
• u	nFLT	页面错误次数
• v	nDRT	最后一次写入到现在，被修改过的页面数。

通过 **f** 键可以选择显示的内容。按 **f** 键之后会显示列的列表，按 **a-z** 即可显示或隐藏对应的列，最后按回车键确定。



Sar 命令

帮助掌握系统资源的使用情况

例：sar -u -o test.log 60 5

表示间隔60秒进行一次采样，采集5次，并将数据以二进制形式保存到test.log

查看二进制文件test.log中的内容，则需键入如下sar命令：

```
sar -u -f test.log
```

- A: 所有报告的总和。
- u: CPU利用率
- v: 进程、I节点、文件和锁表状态。
- d: 硬盘使用报告。
- r: 没有使用的内存页面和硬盘块。
- g: 串口I/O的情况。



vmstat 命令

对操作系统的虚拟内存、
进程、CPU活动进行监控

例： vmstat 5 5 表示在5秒时间内进行5次采样

(1) 进程procs:

r: 在运行队列中等待的进程数。
b: 在等待io的进程数。

(2) Linux 内存监控内存memoy:

swpd: 现时可用的交换内存（单位KB）。
free: 空闲的内存（单位KB）。
buff: 缓冲去中的内存数（单位： KB）。
cache: 被用来做为高速缓存的内存数（单位： KB）。

(3) Linux 内存监控swap交换页面

si: 从磁盘交换到内存的交换页数量，单位： KB/秒。
so: 从内存交换到磁盘的交换页数量，单位： KB/秒。

vmstate

(4) Linux 内存监控 io块设备:

bi: 发送到块设备的块数，单位： 块/秒。
bo: 从块设备接收到的块数，单位： 块/秒。

(5) Linux 内存监控system系统:

in: 每秒的中断数，包括时钟中断。
cs: 每秒的环境（上下文）转换次数。

(6) Linux 内存监控cpu中央处理器:

us: 用户进程使用的时间。以百分比表示。
sy: 系统进程使用的时间。以百分比表示。
id: 中央处理器的空闲时间。以百分比表示。



ps命令

列出系统出当前正在运行的进程的瞬间状态

linux上进程有5种状态:

1. 运行(正在运行或在运行队列中等待)
2. 中断(休眠中, 受阻, 在等待某个条件的形成或接收到信号)
3. 不可中断(收到信号不唤醒和不可运行, 进程必须等待直到有中断发生)
4. 僵死(进程已终止, 但进程描述符存在, 直到父进程调用wait4()系统调用后释放)
5. 停止(进程收到SIGSTOP, SIGSTP, SIGTIN, SIGTOU信号后停止运行)

ps工具标识进程的5种状态码:

D 不可中断 uninterruptible sleep (usually IO)

R 运行 runnable (on run queue)

S 中断 sleeping

T 停止 traced or stopped

Z 僵死 a defunct ("zombie") process

常用命令:

- 1、`ps -A` 显示所有进程信息
- 2、`ps -u username` 显示指定用户信息
- 3、`ps -ef` 显示所有进程信息, 连同命令行
- 4、`ps -ef|grep ssh` ps 与grep 常用组合用法, 查找特定进程
- 5、`ps aux` 列出目前所有的正在内存当中的程序



iostat命令

对操作系统的的磁盘操作活动进行监视

例： iostat 5 5 每隔5秒进行一次刷新，刷新5次采样

常见用法

iostat -d -k 1 10

#查看吞吐量信息

iostat -d -x -k 1 10

#查看设备使用率（%util）、响应时间（await）

iostat -c 1 10

#查看cpu状态



性能测试工具

- Linux time 工具
- gprof: 串行
- Mpip: 并行-进程级
- intel@VTune: 并行-线程级、体系结构行为



Linux time工具

不需要插装在程序中的: time

eg: time ./test

运行结束后得到三个时间: real user 和sys

real: 从程序开始到程序执行结束时所消耗的时间，包括CPU的用时和所有延迟程序执行的因素的总和

CPU用时被划分为**user**和**sys**两块：

user 程序本身，以及它所调用的库中的子例程使用的时间

sys 由程序直接或间接调用的系统调用执行的时间

real = cpu用时+其他因素时间

cpu 用时= user + sys

所以: real > user + sys (单核情况)



插装到程序中的：

`clock()`: 返回的是程序运行过程中的CPU

`Getrusage()`: 获得进程的相关资源信息。如： 用户开销时间， 系统开销时间， 接收的信号量等等

`Clock_gettime(clockid_t which_clock, struct timespec *tp):`

`which_clock`常用3种类型：

`CLOCK_REALTIME`: 系统实时时间, 随系统实时时间改变而改变

`CLOCK_PROCESS_CPUTIME_ID`: 进程级, CPU用时

`CLOCK_THREAD_CPUTIME_ID`: 线程级, CPU用时

`Gettimeofday():`

更多参考可见：<http://linux.die.net/man/>



gprof:

- 编译加-pg选项: `gcc -o example_gprof -pg example_gprof.c`
- 运行程序: `./example_gprof`
- 运行后出现文件: `gmon.out`
- 查看`gmon.out`文件: `gprof -b example_gprof gmon.out |less`



```
[dingnan@cn001 gprof]$ ./example_gprof
main()  function()
    +--- call b()      function
        +---call a()  function
        +---call c()  function
[dingnan@cn001 gprof]$
```

```
1 Flat profile:
2
3 Each sample counts as 0.01 seconds.
4 no time accumulated
5
6   % cumulative   self           self      total
7   time    seconds   seconds  calls  Ts/call  Ts/call  name
8   0.00     0.00     0.00     1     0.00     0.00  a
9   0.00     0.00     0.00     1     0.00     0.00  b
10  0.00     0.00     0.00     1     0.00     0.00  c
11 ^L
12
13          Call graph
14
15 granularity: each sample hit covers 2 byte(s) no time propagated
16
17 index % time   self  children   called      name
18
19 [1]   0.0   0.00   0.00     1/1          b [2]
20 -----
21
22 [2]   0.0   0.00   0.00     1          a [1]
23 -----
24
25
26 [3]   0.0   0.00   0.00     1/1          main [11]
27
28 [2]   0.0   0.00   0.00     1          b [2]
29
30 [3]   0.0   0.00   0.00     1          c [3]
31
32 [1] a
33 [2] b
34 [3] c
```



mpiP

- mpiP: Lightweight, Scalable MPI Profiling
- Open source MPI profiling library
 - LLNL&ORNL
 - Works with any MPI library
- Easy to use
 - PMPI instrumentation
 - No additional tool(libunwind)
 - Text file output
 - One file



To know

- Aggregate statistics over time
 - Number of invocations
 - Data volume
 - Time spent during function execution
- Multiple aggregations options/granularity
 - By function name or type
 - By source code location (call stack)
 - By process rank



Use mpiP

- Add “-L/mpiP-Path/lib -lmpiP -lbfd -liberty –lunwind”
- Use –g option when compiled recommended
 - To know which subroutines calls MPI functions

libbfd：对二进制文件进行分析，在**mpip**中实现共享对象的来源查找

libiberty：为GNU程序提供标准C接口子程序调用集合，在**mpip**中为解码符号信息提供支持

Libunwind：提供了基本的堆栈辗转开解功能，包括输出程序的堆栈信息



Example

- 首先是不同进程的执行时间，分别是应

@--- MPI Time (seconds) ---			
Task	AppTime	MPITime	MPI%
0	10	0.000243	0.00
1	10	10	99.92
2	10	10	99.92
3	10	10	99.92
*	40	30	74.94



Example

- 下面会给出对每个mpi函数的调用点

@--- Callsites: 2 ---					
ID	Lev	File/Address	Line	Parent_Funct	MPI_Call
1	0	9-test-mpip-time.c	52	main	Barrier
2	0	9-test-mpip-time.c	61	main	Barrier



Example

- 下面会给出每个MPI函数的运行时间信息

```
-----  
@--- Aggregate Time (top twenty, descending, milliseconds) ---  
-----  
Call           Site     Time    App%    MPI%    COV  
Barrier        2       3e+04   75.00   100.00   0.67  
Barrier        1       0.405   0.00    0.00    0.59
```



Example

- 下面会给出消息大小的信息：

@--- Aggregate Sent Message Size (top twenty, descending, bytes) -----					
Call	Site	Count	Total	Avrg	MPI%
Send	7	320	1.92e+06	6e+03	99.96
Bcast	1	12	336	28	0.02



Example

- 下面会给出每个进程的时间信息：

@--- Callsite Time statistics (all, milliseconds): 8 ---								
Name	Site	Rank	Count	Max	Mean	Min	App%	MPI%
Barrier	1	0	1	0.107	0.107	0.107	0.00	44.03
Barrier	1	*	4	0.174	0.137	0.107	0.00	0.00
Barrier	2	0	1	0.136	0.136	0.136	0.00	55.97
Barrier	2	1	1	1e+04	1e+04	1e+04	99.92	100.00
Barrier	2	2	1	1e+04	1e+04	1e+04	99.92	100.00
Barrier	2	3	1	1e+04	1e+04	1e+04	99.92	100.00
Barrier	2	*	4	1e+04	7.5e+03	0.136	74.94	100.00



Example

- 下面会给出每个进程的每个mpi函数的消息大小的相关信息：

@--- Aggregate Sent Message Size (top twenty, descending, bytes) -----					
Call	Site	Count	Total	Avg	MPI%
Send	7	320	1.92e+06	6e+03	99.96
Bcast	1	12	336	28	0.02



mpiP安装与使用提示

1、首先安装Binutils、libbfd、liunwind三个外部库：

使用命令：./configure –prefix=安装路径
make; make install

2、安装mpiP

使用命令：./configure
--prefix=
--with-cc=mpicc
--with-binutils-dir=
LDFLAGS="-L/usr/lib64 -L/vol-th/home/soft/libunwind/lib"
LIBS="-lbfd -liberty"
--with-libunwind=
CFLAGS=-I/vol-th/home/xuewei/soft/libunwind/include
make; make install

3、使用命令，在编译时：

```
mpiicc-g hello.c -o hello -L/vol-th/home/dingnan/mpiP-3.4.1-install/lib -L/vol-th/home/soft/libunwind/lib -L/vol-th/home/soft/binutils/lib -lmpiP -lunwind -lbfd -liberty -lm -lz
```



一些提示：

1、并行程序运行时提示链接不到.so动态库：

在bashrc中，`export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:库路径`

2、慎用 `rm` 删除操作

3、把输出到屏幕的log，同时输出到文件：

例如：原命令 `./test`

利用管道串联输出到文件：`./test | & tee test.log`

4、及时把服务器端的文件和数据在本机做好备份



谢谢~