Linux 下程序的编译

AMAX测试工程师 刘耀卿

主要内容

1. 编译过程
2. 常用编译器介绍
3. gcc编译选项、连接和执行
4. mpi程序编译和执行
5. Makefile 介绍

编译

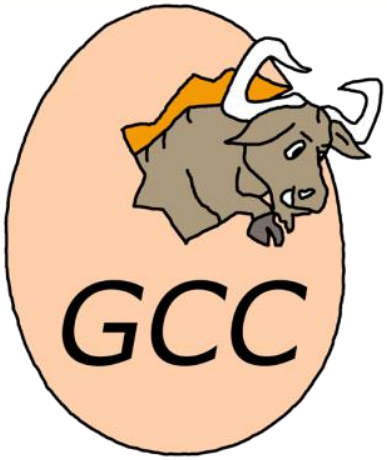
关键词：源程序 可执行程序

编译过程是将人可识别的源程序（文本文件）转换为机器可识别的二进制文件的过程

```
cat hello.c
#include <stdio.h>
void main(){
  printf("Hello World!\n");
};
```


```
./hello
Hello World!
```

Linux下常用的编译器

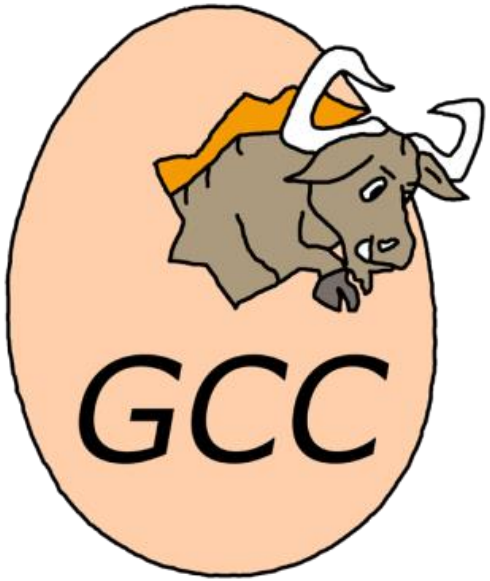


PGI®

intel®

AMD

Linux下常用的编译器



- GCC(GNU Compiler Collection)
- 目前Linux下最常用的C语言编译器
- 是GNU项目中符合ANSI C标准的编译系统，能够编译用C、C++和Object C等语言编写的程序；
- GCC不仅功能非常强大,结构也异常灵活，可以通过不同的前端模块来支持各种语言，如Java、 Fortran、Ada等.

Linux下常用的编译器

- Intel C/C++ Fortran编译器是一种主要针对Inetl平台的高性能编译器
- 在AMD Opteron平台上性能也不错
- 可用于开发复杂且要进行大量计算
- Intel Compiler 有单纯针对某种语言的编译器工具如C/C++/Fortran编译器，MKL函数库，也有如Intel Parallel Studio、 Intel Cluster Toolkit 等编译器套件

Linux下常用的编译器

- Open64是一套针对Itanium 及 x86-64架构最佳化的编译器
- GNU自由文档许可证所发行
- 支持的语言包括C语言、C++及Fortran 77/95以及OpenMP等
- AMD赞助并推广了Open64编译器
- 在AMD平台下可以使用Open64编译器

Linux下常用的编译器

PGI®

- Portland Group 为高性能并行计算市场开发并销售高性能、生产质量级编译器和软件开发工具
- 支持Intel，AMD CPU
- 支持Linux， MAC OS X，Windows操作系统
- 支持C/C++/Fotran语言
- 支持OpenMP，MPI并行
- 支持CUDA，OpenCL语言加速
- 支持OpenACC

gcc编译选项、连接和执行

- 编译：gcc [选项] hello.c

选项	功能
-o	指定编译后生成的可执行文件名.
-c	生成目标代码,不生成可执行文件
-g	在可执行文件中输出调试信息，通常是为调试器gdb所用
-On	代码优化，n随着系统的不同而不同
-Idir	指定头文件搜索目录
-Ldir	指定在目录中需要搜索的库(动态库.so和静态库.a)
-labc	连接指定的库libabc.so或libabc.a
-static	链接时使用静态链接
-shared	生成动态库
-Wall	打开警报信息开关

gcc编译选项、连接和执行

- gcc hello.c　　#生成可执行程序a.out
- gcc –o hello hello.c　#生成可执行程序hello

- ./a.out
  - Hello World!
- ./hello
  - Hello World!

- cat hello.c

```
#include <stdio.h>
main()
{
    printf("Hello World!\n");
}
```

gcc编译选项、连接和执行

- gcc –o hello hello.c hello11.c
- ./hello

Hello World!
hello11

- cat hello.c

```
#include <stdio.h>
Void hello11();
main()
{
   hello11();
   printf("Hello World!\n");
}
```

- cat hello11.c

```
#include <stdio.h>
void hello11()
{
   printf("hello11\n");
}
```

gcc编译选项、连接和执行

```
gcc –c hello.c                          //生成hello.o
gcc –c hello11.c                        //生成hello11.o
gcc –o hello hello.o hello11.o    //生成hello
```

```
cat Makefile
hello: hello.o hello11.o
        gcc –o hello hello.o hello11.o
hello.o:hello.c
        gcc –c hello.c
hello11.o:hello11.c
        gcc –c hello11.c
clean:
        rm –rf hello hello.o hello11.o
```

gcc编译选项、连接和执行

```
gcc –o hellosin hellosin.c  -lm  -L/home/hpc/ -lmy
```

```
cat hellosin.c
#include <stdio.h>
#include <math.h>
#define pi 3.1415927
Void main(){
  sin(pi/4);
   myfun(pi/2);      //假设myfun在 /home/hpc/libmy.a 中
}
```

gcc编译静态、动态库

●共享库的生成

使用-fPIC参数生成位置无关代码

使用-shared参数生成共享目标库

例如: gcc -fPIC -shared -o libfoo.so foo.c

●静态库的生成

gcc -c libfoo.c –o libfoo.o

ar –rcs libfoo.a libfoo.o

mpi并行程序的编译

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );
double f( double a )
{
   return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
   int done = 0, n, myid, numprocs, i;
   double PI25DT = 3.14159265358979323846264643;
   double mypi, pi, h, sum, x;
   double startwtime = 0.0, endwtime;
   int  namelen;
   char
processor_name[MPI_MAX_PROCESSOR_NAME];

MPI_Init(&argc,&argv);

MPI_Comm_size(MPI_COMM_WORLD,&numpr
ocs);

MPI_Comm_rank(MPI_COMM_WORLD,&myid);

MPI_Get_processor_name(processor_name,&n
amelen);

   fprintf(stderr,"Process %d on %s\n",
                  myid, processor_name);

   n = 0;
   while (!done)
   {
      if (myid == 0)
      {
/*
         printf("Enter the number of intervals: (0
quits) ");
```

mpi并行程序的编译

```
scanf("%d",&n);
*/
                if (n==0) n=100; else n=0;

                startwtime = MPI_Wtime();
      }
      MPI_Bcast(&n, 1, MPI_INT, 0,
MPI_COMM_WORLD);
      if (n == 0)
         done = 1;
      else

 {
         h   = 1.0 / (double) n;
         sum = 0.0;
         for (i = myid + 1; i <= n; i += numprocs)
         {
            x = h * ((double)i - 0.5);
            sum += f(x);
         }
```

```
 mypi = h * sum;

         MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE,
MPI_SUM, 0, MPI_COMM_WORLD);

         if (myid == 0)
                 {
            printf("pi is approximately %.16f, Error
is %.16f\n",
                  pi, fabs(pi - PI25DT));
                        endwtime = MPI_Wtime();
                        printf("wall clock time =
%f\n",
                              endwtime-startwtime);

                 }
      }
   }
   MPI_Finalize();

   return 0;
}
```

mpi并行程序的编译

编译： mpicc –o cpi mpi.c

单机运行： mpiexec –n 2 ./cpi

多机运行： mpiexec –n 2 -hostfile hf ./cpi

cat hf
compute-0-1
compute-0-2

Makefile介绍

Make的功能

- 使应用程序的编译和连接自动化
- 缩短编译可执行文件的时间
- 管理大型项目
- 按照代码之间的时间依赖关系维护文件

Make的规则文件：Makefile

- Makefile文件的内容是描述项目或软件（包）中的模块之间的相互依赖关系以及目标文件、可执行程序产生时要执行的命令等。
- Makefile文件主要含有一系列的规则
- 每条规则包含：一个目标，依赖，命令

```
目标：依赖1，依赖2
        命令                    #命令前有一TAB
```

Makefile介绍

Makefile 有几个常有用的变量：$@，$^，$<
$@：目标文件
$^：所有的依赖文件
$<：第一个依赖文件

```
cat Makefile
hello: hello.o hello11.o
        gcc –o hello hello.o hello11.o
hello.o:hello.c
        gcc –c hello.c
hello11.o:hello11.c
        gcc –c hello11.c
clean:
        rm –rf hello hello.o hello11.o
```


```
cat Makefile
hello: hello.o hello11.o
        gcc –o $@ $^
Hello.o: hello.c
        gcc –c $<
hello11.o:hello11.c
        gcc –c $<
clean:
        rm –rf hello hello.o hello11.o
```

Thanks