IBM JS22 刀片服务器用户使用指南

中国科学技术大学 超级运算中心 李会民

2008 年 4 月

目录

1 IBM BladeCenter JS22 刀片服务器概述

中国科学技术大学超级运算中心的峰值计算能力为每秒 1 万亿次 IBM Blade Center JS22 刀片服务器（以下简称 JS22）于 2008 年 3 月建成，由两个刀片中心构成，每个刀片中心含有八把刀片，刀片中心之间利用 Infiniband 高速互联。每把刀片含有两颗主频为 4.0 GHz 的IBM POWER6 双核 64 位处理器和一块 10000 转容量为 146 GB 的 SAS 硬盘，四个核之间为 SMP 架构，共享 16 GB 667 MHz DDR 内存。操作系统为 IBM AIX 5.3L，安装有 IBM 企业版的 C/C++、Fortran 编译器和 PE 并行环境，支持 C/C++ 和 Fortran77/90/95/2003 程序及 OpenMP（只能在每把刀片内部的四个核之间）和 MPI 并行，安装有数学加速子系统 MASS、基本线性代数函数库 BLAS、工程与科学子程序库 ESSL 和其并行版本 PESSL 等，作业管理系统为 LoadLeveler。

本指南主要将对在此系统上运行作业做一基本介绍，详细信息请参看相应的指南，超算中心也将为用户需要提供必要的技术支持。

2 用户登录与文件传输

JS22 操作系统为 IBM AIX 5L 5.3，用户需以 ssh 方式登录上此节点后进行编译、运行等操作（不支持 telnet 协议，用户在 MS Windows 下可利用 putty[1] 等支持 ssh 协议的软件进行登录），用户数据则可以利用 ftp和 sftp 协议进行传输（建议在客户端设置使用安全的 sftp 协议）。

用户 ssh 登录进来后默认的 shell 为 ksh，可以利用 chsh 命令修改为自己喜欢的 shell，比如 bash，用户修改密码可以在主节点（master）上运行 passwd 命令。

IBM AIX 为 UNIX 系统的一种，用户所需要的常用命令与 HP UX 和 Linux 类似，可以用 **man** 或者命令加 -h 或 –help 参数来查看命令信息，详细信息请参考 AIX 手册。

[1]putty 下载：http://scc.ustc.edu.cn/download/putty.exe

3 串行及 OpenMP 程序编译

JS22 支持 C/C++、Fortran 的串行程序，以及与 OpenMP 和 MPI 结合的并行程序，用户只需要在登录节点上以相应的编译命令和参数进行编译即可。当前安装的编译环境为：

- C/C++ 编译器：IBM XL C/C++ Enterprise Edition V9.0
- Fortran 编译器：IBM XL Fortran Enterprise Edition V11.1
- MPI 并行环境：IBM Parallel Environment(PE) V4.3

3.1 C/C++ 程序的编译

IBM XL C/C++ Enterprise Edition for AIX 是一种高性能的高级编译器，可用于开发复杂且要进行大量计算的程序，也可对 Fortran 程序进行语言间调用。其编译命令主要为 **xlc**、**xlc++**、**xlC**、**cc**、**c89**、**c99**、**xlCcore**、**xlc++core** 及相关命令：

- **xlc**、**xlc_r7**、**xlc128**、**xlc128_r**、**xlc128_r4** 和 **xlc128_r7**：对 C 源文件进行编译。（ANSI C89、ISO C99 和 IBM 语言扩展）
- **xlc++**、**xlc++_r**、**xlc++_r4**、**xlc++_r7**、**xlc++128**、**xlc++128_r**、**xlc++128_r4**、**xlc++128_r7**、**xlC**、**xlC_r**、**xlC_r4**、**xlC_r7**、**xlC128**、**xlC128_r**、**xlC128_r4** 和 **xlC128_r7**：对 C++ 源文件进行编译。
- **cc**、**cc_r**、**cc_r4**、**cc_r7**、**cc128**、**cc128_r**、**cc128_r4** 和 **cc128_r7**：对不符合标准 C 的旧代码进行编译。（pre-ANSI C）
- **c89**、**c89_r**、**c89_r4**、**c89_r7**、**c89_128**、**c89_128_r**、**c89_128_r4** 和 **c89_128_r7**：对完全符合 C89 标准的 C 源文件进行编译。（ANSI C89）
- **c99**、**c99_r**、**c99_r4**、**c99_r7**、**c99_128**、**c99_128_r**、**c99_128_r4** 和 c99_128_r7：对完全符合 C99 标准的 C 源文件进行编译。（ISO 99）
- **xlc++core**、**xlc++core_r**、**xlc++core_r7**、**xlc++core128**、**xlc++core128_r**、**xlc++core128_r7**、**xlCcore**、**xlCcore_r**、**xlCcore_r7**、**xlC128core**、**xlC128core_r** 和 **xlC128core_r7**：对 C++ 源文件进行编译，但将仅链接至运行时库的核心。

- **gxlc**：接受 GNU C 选项并将它们映射至等价的 XL C 选项，然后调用 **xlc**。
- **gxlc++** 和 **gxlC**：接受 GNU C/C++ 选项并将它们映射至等价的 XL C++ 选项，然后调用 **xlc++**。

所有带后缀 _r 的调用命令考虑了线程安全（r4 和 r7 分别表示 DCE 和 53 POSIX Draft 7 Threads 标准），主要用于编译多线程和 OpenMP 程序时使用。128 表示将 64 位双精度浮点扩展到 128 位。

这些命令间的主要差别在于使用的缺省选项不同（由配置文件 /etc/vac.cfg.53 设置）。请参阅 Compiler Reference，以了解有关这些调用命令的更多信息。

与一般 C/C++ 编译器的不同的几个重要参数（主要以 q 为前缀的）：

- -q32 或 -q64：选择以 32 位或 64 位编译方式。将 -q32 和 -q64 选项与 -qarch 和 -qtune 选项一起使用，可针对编译器输出的目标体系结构进行优化。缺省值：-q32。
- -qsmp=omp：启用“严格遵守 OpenMP”。只能识别 OpenMP 并行编译指令。
- -qarch=<suboption>：指定该生成代码（指令）的一般处理器体系结构以提高性能。通常，-qarch 选项允许您将特定体系结构作为编译的目标。对于任何给定的 -qarch 设置，编译器将缺省为特定匹配 -qtune 设置，这可以提供额外的性能改进。针对 JS22一般设置为 auto、pwr6 或 pwr6e。
- -qtune=<suboption>：指定优化可执行程序的体系结构系统。<suboption> 针对 JS22 一般设置为 auto、pwr6 或 pwr6e。
- -qipa=<suboption>：执行过程间分析优化。
- -qpdf1=<suboption>、-qpdf2=<suboption>：通过概要定向反馈（PDF）调整优化。
- -qlanglvl=<suboptions_list>：选择编译的语言级别和语言选项。<suboptions_list> 可以为{ classic |extended|saa|saa12|stdc89|stdc99|extc89|extc99}:{ucs|noucs} 中的某一种。

C/C++ 程序编译举例：

- **xlc_r -o yourprog yourprog.c**
  将 C 程序 yourprog.c 编译为 32 位可执行程序 yourprog

- **xlc++ -o yourprog -q64 yourprog.C**
  将 C++ 程序 yourprog.C 编译为 64 位可执行程序 yourprog

- **xlc_r -o yourprog-omp -qsmp=omp -qarch=auto -qtune=auto -q64 yourprog.c**
  将 OpenMP 的 C 程序 yourprog-omp.c 编译为 64 位的针对硬件平台自动优化的 yourprog-omp

XL C/C++ 附带提供了下列库：

- SMP 运行时库，支持显式并行处理和自动并行处理。

- 数学加速子系统（MASS）库，含有 32 和 64 位的优化过的数学内部函数。在某些优化级别时会自动调用，或者利用编译参数进行显式调用。请参阅 XL C/C++ Programming Guide 中的 Using the Mathematical Acceleration Subsystem。

- 优化的基本线性代数子系统（BLAS）库，可以计算一般矩阵或其转置矩阵的矩阵向量乘积，对一般矩阵或其转置矩阵执行组合的矩阵乘法和加法。请参阅 XL C/C++ Programming Guide 中的 Using the Basic Linear Algebra Subprograms。

关于 IBM XL C/C++ Enterprise Edition V9.0 for AIX 编译器的详细信息，请参考以下资料：

- IBM XL C/C++ Enterprise Edition V9.0 for AIX: Installation guide：包含有关 XL C/C++ 的安装和配置的信息。

- IBM XL C/C++ Enterprise Edition V9.0 for AIX: Getting Started with XL C/C++：包含 XL C/C++ 产品简介，提供了有关设置和配置环境、编译链接程序以及对编译错误进行故障诊断的信息。

- IBM XL C/C++ Enterprise Edition V9.0 for AIX: Compiler Reference：包含有关各种编译选项、编译指示、宏、环境变量和内置函数（包括那些用于并行处理的函数）的信息。

- IBM XL C/C++ Enterprise Edition V9.0 for AIX: Language Reference：包含有关 IBM 对C 和 C++ 编程语言支持（包括为了获取对非专有标准的可移植性和一致性的语言扩展）的信息。

- IBM XL C/C++ Enterprise Edition V9.0 for AIX: Programming Guide：包含有关应用程序的移植、使用 Fortran 代码的语言间调用、库开发、应用程序优化和并行化以及 XL C/C++ 高性能库等高级编程主题的信息。

- IBM XL C/C++ Enterprise Edition V9.0 for AIX: Standard C++ Library Reference：包含标准 C++ 运行时库和头的参考信息。

- C/C++ Legacy Class Libraries Reference：包含关于 USL I/O 流库和复数数学函数库的参考信息。

与 XL C/C++ 相关的更多文档，如红皮书、白皮书、教程和其他文章，都可以在 Web 上获得，网址为：http://www-306.ibm.com/software/awdtools/xlcpp/library/。

同时也可以从 XL C/C++ 的技术支持页面获得其他技术支持，网址为：http://www-306.ibm.com/software/awdtools/xlcpp/support/，此页面提供一个具有搜索能力的门户网站，可以在该网站找到大量的技术支持和 FAQ 文档。

3.2 Fortran 程序的编译

IBM XL Fortran Enterprise Edition for AIX 是一种高性能的高级编译器，可以用于开发复杂且要进行大量计算的程序，包括对 C/C++ 程序进行语言间调用。其命令主要为 **xlf**、**xlf_r**、**xlf_r7**、**f77**、**fort77**、**f90**、**xlf90**、**xlf90_r**、**xlf90_r7**、**f95**、**xlf95**、**xlf95_r**、**xlf95_r7**、**f2003**、**xlf2003**、**xlf2003_r** 及相关命令：

- **xlf**、**xlf_r**、**xlf_r7**、**f77**、**fort77**：编译 Fortran77 程序

- **xlf90**、**xlf90_r**、**xlf90_r7**、**f90**：编译 Fortran90 程序

- **xlf95**、**xlf95_r**、**xlf95_r7**、**f95**：编译 Fortran95 程序

- **xlf2003**、**xlf2003_r**、**f2003**：编译 Fortran2003 程序

所有带后缀 _r 的调用命令考虑了线程安全（r7 表示 53 POSIX Draft 7 Threads 标准），主要用于编译多线程和 OpenMP 程序。

这些命令间的主要差别在于使用的缺省选项不同（这些选项在配置文件 /etc/xlf.cfg.53 中设置）。请参阅 Compiler Reference，以了解有关这些调用命令的更多信息。

与一般 Fortran 编译器的不同的几个重要参数（主要以 q 为前缀的）：

- -q32 或 -q64：选择 32 位或 64 位编译方式。将 -q32 和 -q64 选项与 -qarch 和 -qtune 编译器选项一起使用，可针对编译器输出的目标体系结构对其进行优化。缺省值：-q32
- -qsmp=omp：启用“严格遵守 OpenMP”。只能识别 OpenMP 并行编译指令。
- -qextname=<names>：在指定函数名后加 _，如调用 flush 函数时需要此选项。
- -qarch=<suboption>：指定该生成代码（指令）的一般处理器体系结构以提高性能。通常，-qarch 选项允许您将特定体系结构作为编译的目标。对于任何给定的 -qarch 设置，编译器将缺省为特定匹配 -qtune 设置，这可以提供额外的性能改进。针对 JS22一般设置为 auto、pwr6 或 pwr6e。
- -qtune=<suboption>：指定优化可执行程序的体系结构系统。<suboption> 针对 JS22 一般设置为 auto、pwr6 或 pwr6e。
- -qipa=<suboption>：执行过程间分析优化。
- -qpdf1=<suboption>、-qpdf2=<suboption>：通过概要定向反馈（PDF）调整优化。
- -qlanglvl=<suboptions_list>：选择编译的语言级别和语言选项。<suboptions_list> 可以为{77std|90std|90pure|90ext|95std|95pure|2003std|2003pure|extended} 中的某一种，缺省为 extended。

XL Fortran 编译举例：

- **xlf_r -o yourprog yourprog.f**
  将 Fortran 77 程序 yourprog.f 编译为 32 位可执行程序 yourprog
- **xlf90_r：xlf90_r -o yourprog -q64 yourprog.f90**
  将 Fortran 90 程序 yourprog.f90 编译为 64 位可执行程序 yourprog

- **xlf95_r -o yourprog-omp -qsmp=omp -qarch=auto -qtune=auto -q64 yourprog.f95**
  将 OpenMP 的 Fortran 95 程序 yourprog-omp.f95 编译为 64 位的针对硬件平台优化的 yourprog-omp

有关 IBM XL Fortran Enterprise Edition V11.1 for AIX 的详细信息，请参考以下资料：

- IBM XL Fortran Enterprise Edition V11.1 for AIX: Installation guide：包含有关 XL Fortran 的安装和配置的信息。
- IBM XL Fortran Enterprise Edition V11.1 for AIX: Getting Started with XL Fortran：包含 XL Fortran 产品简介，提供了有关设置和配置环境、编译链接程序以及对编译错误进行故障诊断的信息。
- IBM XL Fortran Enterprise Edition V11.1 for AIX: Compiler Reference：包含有关各种编译器选项、编译指示、宏、环境变量和内置函数（包括那些用于并行处理的函数）的信息。
- IBM XL Fortran Enterprise Edition V11.1 for AIX: Language Reference：包含有关 IBM 对 Fortran 编程语言支持（包括为了获取对非专有标准的可移植性和一致性的语言扩展）的信息。
- IBM XL Fortran Enterprise Edition V11.1 for AIX: Optimization and Programming Guide：包含有关应用程序的移植、使用 Fortran 代码的语言间调用、库开发、应用程序优化和并行化以及 XL Fortran 高性能库等高级编程主题的信息。

XL Fortran 附带提供了下列库：

- SMP 运行时库，它支持显世并行处理和自动并行处理。
- 数学加速子系统（MASS），它含有 32 和 64 位的优化过的数学内部函数。在某些优化级别时会自动调用，或者也可指定编译参数进行显式调用。请参阅 XL Fortran Optimization and Programming Guide 中的 Using the Mathematical Acceleration Subsystem。

- 优化的基本线性代数子系统（BLAS）库，可以计算一般矩阵或其转置矩阵的矩阵向量乘积，对一般矩阵或其转置矩阵执行组合的矩阵乘法和加法。详情请参阅 XL Fortran Optimization and Programming Guide 中的 Using the Basic Linear Algebra Subprograms。

与 XL Fortran 相关的更多文档，如红皮书、白皮书、教程和其他文章，都可以在 Web 上获得，网址为：http://www-306.ibm.com/software/awdtools/fortran/xlfortran/library/。

同时也可以从 XL Fortran 的技术支持页面获得其他技术支持，网址为：http://www-306.ibm.com/software/awdtools/fortran/xlfortran/support/，此页面提供一个具有搜索能力的门户网站，可以在该网站找到大量的技术支持和 FAQ 文档。

3.3 OpenMP 程序的编译与运行

如前面所述，XL C/C++ 和 XL Fortran 编译器支持 OpenMP 并行，只需要利用线程安全的编译命令（带 _r 的），采用 -qsmp=omp 编译参数进行编译即可：

- **xlc_r -o yourprog-omp -qsmp=omp -qarch=auto -qtune=auto -q64 yourprog.c**
  将 OpenMP 的 C 程序 yourprog-omp.c 编译为 64 位的针对硬件平台自动优化的 yourprog-omp

- **xlf95_r -o yourprog-omp -qsmp=omp -qarch=auto -qtune=auto -q64 yourprog.f95**
  将 OpenMP 的 Fortran 95 程序 yourprog-omp.f95 编译为 64 位的针对硬件平台优化的 yourprog-omp

OpenMP 的运行一般是在运行前设置环境变量 **OMP_NUM_THREADS** 来控制进程数，比如在 C shell 中利用 **setenv OMP_NUM_THREADS 4** 设置或在 bash 中利用 **export OMP_NUM_THREADS=4** 设置来表示用 4 个进程运行。

4 MPI 并行环境：PE

JS22 上安装的 MPI 的并行环境为 IBM Parallel Environment for AIX 5L V4.3，主要包括以下内容：

- 并行操作环境（Parallel Operating Environment-POE）：编译、提交和管理
- MPI 库：支持 MPI 程序
- 并行调试器（pdbx）：用于调试并行程序

4.1 MPI 并行程序的编译

PE 含有的编译命令主要为：**mpCC**、**mpCC_r**、**mpcc**、**mpcc_r**、**mpxlf**、**mpxlf_r**、**mpxlf90**、**mpxlf90_r**、**mpxlf95**、**mpxlf95_r** 和 **mpxlf2003_r**。

对于并行程序，源文件类型和编译命令的对应关系如下：

- **mpcc -o yourprog-mpi yourprog-mpi.c**
  将 C 语言的 MPI 程序 yourprog-mpi.c 编译为 32 位可执行程序 yourprog-mpi
- **mpCC -o yourprog-mpi yourprog-mpi.C**
  将 C++ 语言的 MPI 程序 yourprog-mpi.C 编译为 32 位可执行程序 yourprog-mpi
- **mpxlf -o yourprog-mpi yourprog-mpi.f**
  将 Fortran 77 语言的 MPI 程序 yourprog-mpi.f 编译为 32 位可执行程序 yourprog-mpi
- **mpxlf90 -o yourprog-mpi -q64 yourprog-mpi.f90**
  将 Fortran 90 语言的 MPI 程序 yourprog-mpi.f90 编译为 64 位可执行程序 yourprog-mpi

4.2 MPI 并行程序的运行

MPI 并行程序的运行命令为 **poe**，其格式为：**poe program program_options**，比如 **poe yourprog-mpi -procs 4** 表示以 4 个进程运行 yourprog-mpi。在 JS22 上，请务必用作业管理软件 LoadLeveler 进行提交（否则作业将被杀掉），因此无需再指定 -procs 参数，具体将在作业管理系统中介绍。

4.3 MPI 并行程序调试

PE 提供并行调试器 **pdbx** ，可对并行程序进行调试，基本用法如下：

对并行程序进行调试，需要在编译时添加 -g 参数以编译成可调试的程序：

mpif90_r -g -o yourprog-mpi-dbg yourprog-mpi.f90

以 **pdbx** 启动 2 个进程以进行调试：

pdbx ./yourprog-mpi-dbg -procs 2

进入后 **pdbx**，运行 help 可以看到各种命令，一般来说运行 cont 后可以自动停到出错的地方或者用户设置的断点那里。

利用 **pdbx -a pid** 可以调试正在运行中的进程，其中 pid 为正运行的程序的进程号，可利用 **ps -ef | grep poe** 得到。

有关利用 **pdbx** 并行调试具体信息，请参考 Operation and Use Volume 2：Tools Reference 的 Using the pdbx debugger。

4.4 PE 的相关文档

- Parallel Environment for AIX 5L V4.3: Installation：有关 PE 安装设置的信息。
- Parallel Environment for AIX 5L V4.3: Introduction：有关 PE 的简要介绍。
- Parallel Environment for AIX 5L V4.3: Messages：有关 PE 运行期间正常和错误输出的消息的具体含义，可方便查找问题所在。
- Parallel Environment for AIX 5L V4.3: MPI Programming Guide：有关 MPI 的 IBM 实现的编程指导。
- Parallel Environment for AIX 5L V4.3: MPI Subroutine Reference：描述 MPI 的 IBM 实现的子程序的参考信息。
- Parallel Environment for AIX 5L V4.3: Operation and Use, Volume 1 Using the Parallel Operating Environment：有关利用 PE 对 MPI 并行程序进行编译、优化、运行、分析等信息。
- Parallel Environment for AIX 5L V4.3: Operation and Use, Volume 2 Tools Reference：有关利用 PE 对 MPI 并行程序调试以及分析等信息。

5 数学函数库

JS22 上目前安装的数学函数库主要有：

- 数学加速子系统：Mathematics Acceleration Subsystem(MASS)
- 基本线性代数子系统库：Basic Linear Algebra Subprograms(BLAS)
- 工程与科学子函数库：Engineering and Scientific Subroutine Library(ESSL), V4.2
- 并行工程与科学子函数库：Parallel Engineering and Scientific Subroutine Library(PESSL), V3.3

上面数学函数库集合了常用的 BLAS、LAPACK、ScaLapack、FFT 等数学函数库，用户可以直接调用，以提高性能、加快开发。

5.1 数学加速子系统：MASS

数学加速子系统（Mathematics Acceleration Subsystem-MASS），是优化的基本数学内部函数，包括 sin、exp 等函数，具有 32 和 64 位版本。编译时，在某些优化级别时会自动调用，或者也可设定编译参数以进行显式调用。

5.1.1 XL C/C++ 中调用 MASS 标量库

MASS 标量库 libmass.a 安装在 /usr/lib 下，包含一些优化过的常用数学内部函数，这些函数在用户使用如下参数编译程序时将自动调用：

- -qhot -qignerrno -qnostrict
- -qhot -O3
- -qsmp -O3
- -O4
- -O5

XL C/C++ 编译器自动对大多数数学函数调用更快的 MASS 函数，实际上，编译器首先尝试调用等价的 MASS 向量库进行向量调用，如果失败，那么将调用标量函

数。当编译器实现自动代替数学库函数时，它调用的库包含在系统库 libxlopt.a 中。调用时，用户无需在源码中添加任何特殊的调用，或者特别指定链接到 libxlopt 库。

如果用户未使用上述的任何优化选项，想明确指定调用 MASS 标量函数，可以采用下述方法：

- 在源码中包含 math.h 以提供函数原型（anint、cosisin、dnint、sincos 除外）。
- 在源码中包含 mass.h 以提供 anint、cosisin、dnint、sincos 函数的原型。
- 在链接时指定 libmass.a。

5.1.2 XL C/C++ 中调用 MASS 向量库

在用户使用如下参数编译程序时将自动调用 MASS 向量库：

- -qhot -qignerrno -qnostrict
- -qhot -O3
- -O4
- -O5

XL C/C++ 编译器会自动尝试通过调用等价的 MASS 向量函数来向量调用除 vdnint、vdint、vsincos、vssincos、vcosisin、vscosisin、vqdrt、vsqdrt、vrqdrt、vsrqdrt、vpopcnt4、vpopcnt8 之外的系统数学函数。编译器自动代替数学库函数时，它调用的库包含在系统库 libxlopt.a 中，用户无需在源码中添加任何特殊的调用，或者特别指定链接到 libxlopt 库。如果用户没使用上述的优化选项，想显式指定调用的库，可在源文件中包含 mass.h 以调用下面的库：

- libmassv.a：通用向量库
- libmassvp3.a：某些函数针对 POWER3 优化，其余等价于 libmassv.a 中的
- libmassvp4.a：某些函数针对 POWER4 优化，其余等价于 libmassv.a 中的
- libmassvp5.a：某些函数针对 POWER5 优化，其余等价于 libmassv.a 中的
- libmassvp6.a：某些函数针对 POWER6 优化，其余等价于 libmassv.a 中的，建议在 JS22 上使用

详情请参阅 XL C/C++ Programming Guide 中的 Using the Mathematical Acceleration Subsystem。

5.1.3 XL C/C++中调用 MASS 库的编译与链接

为了使程序链接时调用 MASS 库，请在链接参数中添加 mass 和 massv（或 massvp3、massvp4、massvp5、massvp6），比如用如下方式进行编译：

xlc progc.c -o progc -lmass -lmassv

如果用户想对某些函数调用 libmass.a 标量库，而对其余的函数调用通常的数学库 libm.a 中的，可按下面流程进行编译链接：

- 生成一包含想调用函数的列表（可为一纯文本文件）。例如在程序 sample.c 中只想从 libmass.a 调用快速的正切函数，那么可以生成一个文件 fasttan.exp，内部只需要有一行：tan。

- 利用 **ld** 命令链接 libmass.a 库，生成共享目标文件：

  ld -bexport:fasttan.exp -o fasttan.o -bnoentry -lmass -bmodtype:SRE

- 利用 **ar** 命令打包此共享库：

  ar -q libfasttan.a fasttan.o

- 利用 **xlc** 生成最终的可执行文件时，在标准数学库 libm.a 前指明包含 MASS 函数的目标文件。这将只链接在此目标文件中的函数（在上述例子中为 tan 函数），其余的将调用标准数学库中的函数：

  xlc sample.c -o sample -Ldir_containing_libfasttan -lfasttan -lm

注意：在调用 MASS 中，如果指明调用 cosisin 函数，那么 sincos 函数将自动被链接，如果指明调用 sin 函数，那么 cos 函数也将自动被链接，如果指明调用 atan 函数，那么 atan2 函数也将自动被链接。

5.1.4 XL Fortran 中调用 MASS 标量库

MASS 标量库 libmass.a 安装在 /usr/lib 下，包含一些优化过的常用数学内部函数，这些函数在用户使用如下参数编译程序时将自动调用：

- -qhot -qnostrict

- -qhot -O3
- -qsmp -O3
- -O4
- -O5

XL Fortran 编译器自动对大多数数学函数调用更快的 MASS 函数，实际上，编译器首先尝试调用等价的 MASS 向量库进行向量调用，如果失败，那么将调用标量函数。当编译器自动代替数学库函数时，它调用的库包含在系统库 libxlopt.a 中。用户无需在源码中添加任何特殊的调用，或者特别指定链接到 libxlopt 库。

如果用户未使用上述的任何优化选项，想明确指定调用 MASS 标量函数，可采用下述方法：

- 在用户的应用中调用 MASS 标量库进行链接
- 所有 MASS 标量程序，除了那些在下面列出的函数被 XL Fortran 重组作为内部函数外，无需明确指定接口块。如果需要调用如下函数，需明确指定接口块，并且要在源码中引用 mass.include：

  acosf、acosh、acoshf、asinf、asinh、asinhf、atan2f、atan、atanf、atanh、atanhf、cbrt、cbrtf、copysign、copysignf、cosf、coshf、cosisin、erff、erfcf、expf、expm1f、hypot、hypotf、lgammaf、logf、log10f、log1pf、rsqrt、sinf、sincos、sinhf、tanf、tanhf、x**y

5.1.5 XL Fortran 中调用 MASS 向量库

在用户使用如下参数编译程序时将自动调用 MASS 向量库：

- -qhot -qnostrict
- -qhot -O3
- -O4
- -O5

编译器自动尝试通过调用等价的 MASS 向量函数来向量调用除 vatan2 、vsatan2、vdnint、vdint、vsincos、vssincos、vcosisin、vscosisin、vqdrt、vsqdrt、vrqdrt、vsrqdrt、vpopcnt4、vpopcnt8 之外的系统数学函数。编译器自动代替数学库函数时，它使用的库包含在系统库 libxlopt.a 中，用户无需在代码中添加任何特殊的调用，或者特别指明链接到 libxlopt 库。向量库包含 32 位和 64 位的下列库：

- libmassv.a：通用向量库
- libmassvp3.a：某些函数针对 POWER3 优化，其余等价于 libmassv.a 中的
- libmassvp4.a：某些函数针对 POWER4 优化，其余等价于 libmassv.a 中的
- libmassvp5.a：某些函数针对 POWER5 优化，其余等价于 libmassv.a 中的
- libmassvp6.a：某些函数针对 POWER6 优化，其余等价于 libmassv.a 中的，建议在 JS22 上使用

详情请参阅 XL Fortran Optimization and Programming Guide 中的 Using the Mathematical Acceleration Subsystem。

5.1.6 XL Fortran 中 MASS 库的编译与链接

为了使程序调用 MASS 库，请在链接参数中添加 mass 和 massv（或 massvp3、massvp4、massvp5、massvp6，建议针对 JS22 平台使用 massvp6），可用如下方式进行编译：

xlf progf.f -o progf -lmass -lmassv

如果用户想对某些函数使用 libmass.a 变量库，而对其余的使用通常的数学库 libm.a，可按下面流程进行编译链接：

- 生成一包含想调用函数的列表（可以为一纯文本文件）。例如对 sample.f 只想从 libmass.a 调用快速的正切函数，那么可以生成一个文件 fasttan.exp，内部只需要有一行：tan
- 利用 **ld** 命令链接 libmass.a 库，生成一个共享目标文件：

  ld -bexport:fasttan.exp -o fasttan.o -bnoentry -lmass -bmodtype:SRE

- 利用 **ar** 命令打包此共享库：

  ar -q libfasttan.a fasttan.o

- 利用 **xlf** 生成最终的可执行文件时，在标准数学库 libm.a 前指明包含 MASS 函数的目标文件。这样将只链接在此目标文件中的函数（在此例子中为 tan 函数），其余的将使用标准数学库中的函数：

  xlf sample.f -o sample -Ldir_containing_libfasttan -lfasttan -lm

注意：在调用 MASS 中，如果指明调用 cosisin 函数，那么 sincos 函数将自动被链接，如果指明调用 sin 函数，那么 cos 函数也将自动被链接，如果指明调用 atan 函数，那么 atan2 函数也将自动被链接。

5.2 基本线性代数库：BLAS

5.2.1 BLAS 库的内容

基本线性代数库（Basic Linear Algebra Subprograms-BLAS）由 libxlopt 库提供，主要包含下面的内容：

- sgemv（单精度）和 dgemv（双精度）：计算普通矩阵或其转矩阵的矩阵-向量乘
- sgemm（单精度）和 dgemm（双精度）：计算普通矩阵或其转矩阵的乘与加

由于 BLAS 子程序是用 Fortran 写的，因此所有参数传递采用的是引用（reference）方式，所有数组是以列优先（column-major）方式存储。

注意：libxlopt 中一些出错处理代码已被移除，在调用这些函数的时候不会有出错信息给出。

5.2.2 链接 libxlopt 库中的 BLAS

默认的情况下，当用 XL C/C++ 编译时，libxlopt 库自动被链接到应用程序中，但是如果用户使用第三方的 BLAS 库，并且想利用 libxlopt 提供的 BLAS 函数，那么用户必须在其余 BLAS 库之前指明 libxlopt 库。例如，如第三方库的名字为 libblas，用户可以利用下面命令编译：

xlc app.c -lxlopt -lblas

与 C/C++ 情况下类似，Fortran 调用的时候，需要按照如下方式调用：

xlf app.f -lxlopt -lblas

此时编译器将从 libxlopt 库中调用 sgemv、dgemv、sgemm、dgemm 函数，而其余的 BLAS 函数则从 libblas 中调用。

详情请参阅 XL C/C++ Programming Guide 或 XL Fortran Optimization and Programming Guide 中的 Using the Basic Linear Algebra Subprograms。

5.3 工程与科学子函数库：ESSL

工程与科学子函数库（Engineering and Scientific Subroutine Library-ESSL）是一个优秀的子程序集合，包括广大针对不同科学和工程应用方面所需要的数学函数，它的主要特征为高性能、函数兼容性及易用性。

ESSL 主要针对以下计算领域对性能进行了优化：

- 线性代数子函数
- 矩阵运算
- 线性方程
- 本征系统分析
- 傅立叶变换、卷积和相关性、相关计算
- 排序与搜索
- 插值
- 数值积分
- 随机数产生

使用 ESSL 时需要注意以下几点：

- ESSL 的安装目录为 /usr/lib，编译的调用参数为 -lessl
- 对于 SMP 形式的 ESSL 库，可用 XL Fortran XLSMPOPTS 或 OMP_NUM_THREADS 环境变量来影响 SMP 下的执行
- 如果用户需要用 64 位的 ESSL，需要在编译的时候添加 -q64 参数。

- ESSL 支持 XL Fortran 的编译时选项 -qextname，以在函数后添加_，避免此类函数找不到。
- 利用 XL Fortran 编译时，-qessl 编译参数允许在 Fortran 90 内部过程中调用 ESSL 函数。
- 在 AIX 系统中，ESSL 只能采用动态方式链接，不能采用静态方式。

ESSL 的具体使用方法，请参阅：ESSL for AIX V4.2 & ESSL for Linux on POWER V4.2.2 Guide and Reference 和 http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.essl.doc/esslbooks.html，下面仅做简要介绍。

5.3.1 C 程序调用 ESSL 的基本编译方式

C/C++ 程序的 ESSL 的头文件为 essl.h，安装在 /usr/include 目录下。除非用户想指明自己定义的复杂数据，否则用户一般无需对现有的 C 编译过程进行修改。如果用户想定义自己的短精度和长精度复数数据，需要在编译参数中分别添加 -D_CMPLX 或 -D_DCMPLX，否则将自动使用 ESSL 头文件中定义的短精度和长精度复数数据。

当链接和运行 C 程序时，必须设置合适的参数，一般可采用表 1 中的编译方式。

5.3.2 C++ 程序调用 ESSL 的基本编译方式

C/C++ 程序的 PESSL 的头文件为 essl.h，安装在 /usr/include 目录下。与 C 程序不同，C++ 程序调用 ESSL 进行编译时必需指定 -qnocinc=/usr/include/essl。

如果用户使用 IBM Open Class Complex Mathematics 库，将自动使用 ESSL 头文件中指定的短精度和长精度复数数据。如果用户想定义自己的短精度和长精度复数数据，需要在编译链接参数中分别添加 -D_CMPLX，否则 ESSL 将用 IBM Open Class Complex Mathematics 库或标准数值库。

如用户想明确指定对复数计算使用标准数值库，需添加编译参数 -D_ESV_COMPLEX_。

EESSL 头文件支持两种描述标量输出参数，默认的参数被声明为类型引用（type reference）。如果用户想声明为指针引用，请在编译参数中添加 -D_ESVCPTR。

当链接和运行用户程序时，必须设置对应 ESSL 的库正确，可以采用表 2 中的方式进行编译。

表 1: C 程序调用 ESSL 的基本编译方式

ESSL 库名		命令
SMP	32-bit	cc_r -O xyz.c -lesslsmp cc_r -O -D_CMPLX -D_DCMPLX xyz.c -lesslsmp
	64-bit	cc_r -O -q64 xyz.c -lesslsmp cc_r -O -D_CMPLX -D_DCMPLX -q64 xyz.c -lesslsmp
串行	32-bit	cc_r -O xyz.c -lessl cc_r -O -D_CMPLX -D_DCMPLX xyz.c -lessl
	64-bit	cc_r -O -q64 xyz.c -lessl cc_r -O -D_CMPLX -D_DCMPLX -q64 xyz.c -lessl
串行	32-bit	cc_r -O xyz.c -lessl cc_r -O -D_CMPLX -D_DCMPLX xyz.c -lessl
	64-bit	cc -O -q64 xyz.c -lessl cc -O -D_CMPLX -D_DCMPLX -q64 xyz.c -lessl

表 2: C++ 程序调用 ESSL 的基本编译方式

ESSL 库名		命令
SMP	32-bit	xlC_r -O xyz.C -lesslsmp -qnocinc=/usr/include/essl
		xlC_r -O -D_CMPLX xyz.C -lesslsmp -qnocinc=/usr/include/essl
		xlC_r -O -D_ESV_COMPLEX_ xyz.C -lesslsmp -qnocinc=/usr/include/essl
		xlC_r -O -D_ESVCPTR xyz.C -lesslsmp -qnocinc=/usr/include/essl
	64-bit	xlC_r -O -q64 xyz.C -lesslsmp -qnocinc=/usr/include/essl
		xlC_r -O -D_CMPLX -q64 xyz.C -lesslsmp -qnocinc=/usr/include/essl
		xlC_r -O -D_ESV_COMPLEX_ -q64 xyz.C -lesslsmp -qnocinc=/usr/include/essl
		xlC_r -O -D_ESVCPTR -q64 xyz.C -lesslsmp -qnocinc=/usr/include/essl
串行	32-bit	xlC_r -O xyz.C -lessl -qnocinc=/usr/include/essl
		xlC_r -O -D_CMPLX xyz.C -lessl -qnocinc=/usr/include/essl
		xlC_r -O -D_ESV_COMPLEX_ xyz.C -lessl -qnocinc=/usr/include/essl
		xlC_r -O -D_ESVCPTR xyz.C -lessl -qnocinc=/usr/include/essl
	64-bit	xlC_r -O -q64 xyz.C -lessl -qnocinc=/usr/include/essl
		xlC_r -O -D_CMPLX -q64 xyz.C -lessl -qnocinc=/usr/include/essl
		xlC_r -O -D_ESV_COMPLEX_ -q64 xyz.C -lessl -qnocinc=/usr/include/essl
		xlC_r -O -D_ESVCPTR -q64 xyz.C -lessl -qnocinc=/usr/include/essl
串行	32-bit	xlC -O xyz.C -lessl -qnocinc=/usr/include/essl
		xlC -O -D_CMPLX xyz.C -lessl -qnocinc=/usr/include/essl
		xlC -O -D_ESV_COMPLEX_ xyz.C -lessl -qnocinc=/usr/include/essl
		xlC -O -D_ESVCPTR xyz.C -lessl -qnocinc=/usr/include/essl
	64-bit	xlC -O -q64 xyz.C -lessl -qnocinc=/usr/include/essl
		xlC -O -D_CMPLX -q64 xyz.C -lessl -qnocinc=/usr/include/essl
		xlC -O -D_ESV_COMPLEX_ -q64 xyz.C -lessl -qnocinc=/usr/include/essl
		xlC -O -D_ESVCPTR -q64 xyz.C -lessl -qnocinc=/usr/include/essl

5.3.3 Fortran 程序调用 ESSL 的基本编译方式

对于 Fortran 程序，用户无需修改现有的编译方式，只需要添加必要的参数即可，一般采用表 3 的方式进行编译。

表 3: Fortran 程序调用 ESSL 的基本编译方式

ESSL 库名		命令
SMP	32-bit	xlf_r -O -qnosave xyz.f -lesslsmp
	64-bit	xlf_r -O -qnosave -q64 xyz.f -lesslsmp
串行	32-bit	xlf_r -O -qnosave xyz.f -lessl
	64-bit	xlf_r -O -qnosave -q64 xyz.f -lessl
串行	32-bit	xlf_r -O xyz.f -lessl
	64-bit	xlf_r -O -q64 xyz.f -lessl

5.4 并行工程与科学子函数库：PESSL

并行工程与科学子函数库（Parallel Engineering and Scientific Subroutine Library-PESSL）是可缩放的数值子函数库，支持利用高性能交换机连接的集群上的处理器进行并行处理的应用。PESSL 支持在单程序多数据（SPMD）编程模型中采用 MPI 库。PESSL 提供下面方面的通信：

- 2 和 3 级别的并行基本线性代数子程序（PBLAS）
- 线性代数方程
- 本征系统分析和奇异值分析
- 傅立叶变换
- 随机数产生

对于通信，PESSL 包含利用 MPI 的基本线性代数通信子函数（BLACS），而计算则采用 ESSL 库，PESSL 支持 32 位和 64 位的 Fortran、C/C++ 程序的调用。

注意 PESSL SMP 库，是提供并行环境中的 MPI 线程库，用户不能同时在多线程中调用。

使用 PESSL 时需要注意以下几点：

- PESSL 的安装目录为 /usr/lib，编译的调用参数为 -lpessl。
- 对于 SMP 形式的 PESSL 库，可用 XLSMPOPTS 或 OMP_NUM_THREADS 环境变量来影响 SMP 下的执行。
- 如果用户需要用 64 位的 PESSL，需要在编译的时候添加 -q64 参数。
- PESSL 支持 XL Fortran 的编译时选项 -qextname，以在函数后添加_，避免此类函数找不到。
- ESSL 和 PESSL 是共享库，必须联合使用它们。具有相同名字的等价子程序（比如 libblas.a），尽管在编译参数中代替 ESSL 库的位置，也将不被使用。
- 在 AIX 系统中，PESSL 只能采用动态方式链接，不能采用静态方式。

PESSL 的具体使用方法，请参阅：Parallel ESSL for AIX V3.3 & Parallel ESSL for Linux on POWER V3.3 Guide and Reference 和 http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.essl.doc/esslbooks.html，这里仅做简要介绍。

5.4.1 C 程序调用 PESSL 的基本编译方式

C 和 C++ 程序的 PESSL 的头文件为 essl.h 和 Cblacs.h，安装在 /usr/include 目录下。用户一般无需对现有的 C 编译过程进行修改，除非用户想指明自己定义的复杂数据。如果用户想定义自己的短精度和长精度复数数据，需要在编译链接参数中分别添加 -D_CMPLX 或 -D_DCMPLX，否则将自动使用 ESSL 头文件中定义的短精度和长精度复数数据。

当链接和运行 C 程序时，必须设置合适的参数，一般可采用表 4 中的编译方式。

5.4.2 C++ 程序调用 PESSL 的基本编译方式

C 和 C++ 程序的 PESSL 的头文件为 essl.h 和 Cblacs.h，安装在 /usr/include 目录下。与 C 程序不同，C++ 程序调用 PESSL 进行编译时必需要指定 -qnocinc=/usr/include/essl 参数。

表 4: C 程序调用 PESSL 的基本编译方式

应用模式	命令
32-bit	mpcc_r -O xyz.c -lesslsmp -lpesslsmp -lblacssmp mpcc_r -O -D_CMPLX -D_DCMPLX xyz.c -lesslsmp -lpesslsmp -lblacssmp
64-bit	mpcc_r -O -q64 xyz.c -lesslsmp -lpesslsmp -lblacssmp mpcc_r -O -q64 -D_CMPLX -D_DCMPLX xyz.c -lesslsmp -lpesslsmp -lblacssmp

如果用户使用 IBM Open Class Complex Mathematics 库，将自动使用 ESSL 头文件中指定的短精度和长精度复数数据。如果用户想定义自己的短精度和长精度复数数据，需要在编译链接参数中分别添加 -D_CMPLX，否则 ESSL 将用 IBM Open Class Complex Mathematics 库或标准数值库。

如果想明确指定想对复数计算使用标准数值库，请添加编译参数 -D_ESV_COMPLEX_。

EESSL 头文件支持两种描述标量输出参数，默认的参数被声明为类型引用（type reference）。如果用户想声明为指针引用，请在编译参数中添加 -D_ESVCPTR。

当链接和运行用户程序时，必须设置对应 ESSL 的库正确，可采用表 5 中的方式进行编译。

表 5: C++ 程序调用 PESSL 的基本编译方式

模式	命令
32-bit	mpCC_r -O xyz.C -lesslsmp -lpesslsmp -lblacssmp -qnocinc=/usr/include/pessl mpCC_r -O -D_CMPLX xyz.C -lesslsmp -lpesslsmp -lblacssmp -qnocinc=/usr/include/pessl mpCC_r -O -D_ESV_COMPLEX_ xyz.C -lesslsmp -lpesslsmp -lblacssmp -qnocinc=/usr/include/pessl mpCC_r -O -D_ESVCPTR xyz.C -lesslsmp -lpesslsmp -lblacssmp -qnocinc=/usr/include/pessl
64-bit	mpCC_r -O -q64 xyz.C -lesslsmp -lpesslsmp -lblacssmp -qnocinc=/usr/include/pessl mpCC_r -O -D_CMPLX -q64 xyz.C -lesslsmp -lpesslsmp -lblacssmp -qnocinc=/usr/include/pessl mpCC_r -O -D_ESV_COMPLEX_ -q64 xyz.C -lesslsmp -lpesslsmp -lblacssmp -qnocinc=/usr/include/pessl mpCC_r -O -D_ESVCPTR -q64 xyz.C -lesslsmp -lpesslsmp -lblacssmp -qnocinc=/usr/include/pessl

5.4.3 Fortran 程序调用 PESSL 的基本编译方式

对于 Fortran 程序，用户一般无需修改现有的编译方式，但使用 64 位的 Fortran 90 离散线性代数方程子程序时，需在编译参数中加：-I/usr/lpp/pessl.rte.common/include/64。Fortran 程序调用 PESSL 一般采用表 6 的方式进行编译。

表 6: Fortran 程序调用 PESSL 的基本编译方式

应用模式	命令
32-bit	mpxlf_r -O xyz.f -lesslsmp -lpesslsmp -lblacssmp
64-bit	mpxlf_r -O -q64 xyz.f -lesslsmp -lpesslsmp -lblacssmp mpxlf_r -O -q64 xyz.f -lesslsmp -lpesslsmp -lblacssmp -I/usr/lpp/pessl.rte.common/include/64

6 作业管理系统

JS22 利用 Tivoli Workload Scheduler LoadLeveler 进行资源和作业管理，所有需要运行的作业均必须通过作业提交命令 **llsubmit** 提交，提交后可利用相关命令查询作业状态等。为了利用 **llsubmit** 提交作业，用户必须针对此作业创建提交脚本，在脚本里面设定需要运行的作业参数等。在这里，我们将分别给出串行和并行的简单脚本，用户请修改此脚本以适用于自己的作业，如需要高级功能等，请参考 Tivoli Workload Scheduler LoadLeveler: Using and Administering。

6.1 作业命令文件

作业命令文件包含一些 LoadLeveler 的关键词和注释文字，关键词在以 # @ 开始的行中设定，并紧跟 # @，一般来说一行一个关键词。例如为了指明一个可执行的二进制文件被执行，利用关键词 **executable** 来指定。指明次脚本被执行，可以利用 **executable** 关键词，也可以省略此关键词，此时系统假设这个作业的脚本文件本身为所需要执行的作业。作业命令文件包含下面内容：

- LoadLeveler 关键词声明：关键词指的是在一个作业命令文件中的具有特定含义的词，关键词声明是跟随 LoadLeveler 关键词的声明，请参阅 Tivoli Workload Scheduler LoadLeveler V3.4 Using and Administering 中第 336 页开始的 Job command file keyword descriptions。

- 注释声明：用户可以利用注释使得作业命令文件具有可读性，类似普通脚本文件中的用处。

- 脚本命令声明：若用户使用脚本作为执行命令，作业命令文件可包含脚本命令。

- LoadLeveler 变量：请参阅 Tivoli Workload Scheduler LoadLeveler V3.4 Using and Administering 中第 372 页开始的 Job command file variables。

作业命令文件的一般规定：

- LoadLeveler 关键词以 # @ 开始，在 # 和 @ 之间允许有任意多的空格。

- 注释以 # 开始，任何第一个非空格字符为 # ，并且不是 LoadLeveler 关键词的行被认为为注释。

- 注释以空格为分割，用户可以在其他分割符之前和之后使用空格来提高可读性。
- \ 是续行符，并且要求续行不得以 # @ 开始。如果用户的作业命令文件是需要被执行的脚本，用户必须在续行以 # 开始。请参看 Tivoli Workload Scheduler LoadLeveler V3.4 Using and Administering 中第 169、170 页的例子。
- LoadLeveler 关键词忽略大小写，用户可以使用大写、小写或混合方式。

6.2 串行作业

我们希望用户在 JS22 此平台上尽量运行并行作业，不要运行串行作业（串行作业将浪费此平台的并行优势，请在自己的机子上运行），但考虑到特殊需要，比如生成下一步并行作业所需的前期数据或作业需要大内存无法在其余机子运行等情况下，将不完全禁止串行程序的运行，希望用户自觉。

对于串行程序，用户可编写命名为 serial_job.cmd（此脚本名可以按照用户喜好命名）的串行作业命令文件，其内容如下：

```
# This job command file lists a job step called 'step1',
# which input file name is 'step1.in',
# screen output file name is 'step1.log',
# screen error output file name is 'step1.error',
# the cpu time for this job is 6000s, and if overtime, the job will be terminated.
# the class for this job is serial,
# the job's executable file name is executable1.
# @ step_name = step1
# @ input = step1.in
# @ output = step1.log
# @ error = step1.error
# @ wall_clock_limit = 6000
# @ class = serial
# @ executable = executable1
# @ queue
```

下面脚本与上面的功能一样，但由于没有用 **executable** 关键词指定需要运行的作业，此时将此作业命令文件作为作业，即将执行此作业命令文件的内容 **executable1**（一般为可执行程序名，如 /your/prog/name）。

```
# @ step_name = step1
# @ input = step1.in
# @ output = step1.log
# @ error = step1.error
# @ wall_clock_limit = 6000
# @ class = serial
# @ queue
/your/prog/name
```

上述作业命令文件的含义为：提交一个作业名为 job_name 的作业到 serial 队列（与 LSF 不同，在 LoadLeveler 中称为 class）以运行命令 /your/prog/name，/your/prog/name 输入文件为 step1.in，正常屏幕输出到 step1.log，错误信息输出 step1.error 中，此程序的最长运行时间为 6000 秒。上述脚本中以 # @ 开头的几行的 = 前的为 LoadLeveler 关键词，其余 # 后面的内容与一般脚本中的一样表示注释。关键词 queue 表示执行由此前信息所设置的作业。

作业命令文件编写完成后，可以按照下面命令提交作业：

llsubmit ser_job.cmd

如果成功，将有类似下面的输出：

```
llsubmit: The job "master.74" has been submitted.
```

其中 master.74 表示作业号，组成形式为 host.jobid，分别对应主机名、作业序号，之后可利用此作业号来进行查询、终止此作业等操作。

对于 32 位程序来说，若程序运行需要超过 256MB 内存，需在作业脚本中的执行命令（比如 /your/prog/name）前，添加设置环境变量的选项：

- **export LDR_CNTRL=MAXDATA=0x40000000** #可使用 1 GB 内存
- **export LDR_CNTRL=MAXDATA=0x80000000** #可使用 2 GB 内存

如果需要更大的内存，请在编译的时候添加 -q64 编译成 64 位的可执行文件。

6.2.1 多个作业

LoadLeveler 作业命令文件支持按顺序运行多个作业，既可设置为只有在前一个作业正常完成后才运行下面作业，也可设置为即使前面作业出错下面作业也将运行。

```
# This job command file lists two job steps called 'step1'
# and 'step2'. 'step2' only runs if 'step1' completes
# with exit status = 0. Each job step requires a new
# queue statement.
# @ step_name = step1
# @ executable = executable1
# @ input = step1.in
# @ output = step1.$(jobid).$(stepid).out
# @ error = step2.err
# @ queue
# @ dependency = (step1 == 0)
# @ step_name = step2
# @ executable = executable2
# @ input = step2.in
# @ output = step2.$(jobid).$(stepid).out
# @ error = step2.$(jobid).$(stepid).err
# @ queue
```

以上设置了两个作业，名字分别为 step1 和 step2，将分别执行 executable1 和 executable2，由于设置了关键词 **dependency = (step1 == 0)**，step2 只有在 step1 正常完成后才会运行。

如果在上面的作业命令文件中去掉 **dependency** 关键词，同时添加关键词 **coschedule = true**，那么只有在两个作业都获取到所需资源后，作业才开始同时运行，如果没必要要求两个作业必须同时运行，请不要设置此参数，否则会影响作业及时被运行。

如果几个作业直接无任何依赖关系，请不要添加 **dependency** 或 **coschedule** 关键词，以免影响运行。

上述作业的输出文件名将由于引用了作业运行时的 **jobid** 和 **stepid** 变量，将会与运行的作业号等相联系，这对提交多个作业来说非常有用，可以避免冲突。LoadLeveler 提供了非常多的变量供用户使用，如果需要详细的变量说明，请参看 Tivoli Workload Scheduler LoadLeveler V3.4: Using and Administering 第372 页的 Job command file variables。

6.3 并行作业

与串行作业类似，对于并行作业，需要编写类似下面脚本 par_job.cmd：

```
# A example for parallel job.
# @ job_type = parallel
# set to run parallel job
# @ environment = COPY_ALL
# set to copy all environment variable to node
# @ input = step1.in
# @ output = step1.log
# @ error = step1.error
# @ node = 1
# set to use 1 node to run.
# @ tasks_per_node = 8
# set to fork 8 threads for every node.
# @ wall_clock_limit = 6000
# @ notification = never
# @ class = medium
# @ queue
/usr/bin/poe /your/prog/name
```

与串行程序的作业脚本文件相比，主要不同之处在于通过关键词 **job_type** 指明为并行程序，利用 **environment** 设置将所有环境变量复制到运行节点，利用 **node** 设置节点数（现阶段只允许一个作业只能在一个节点上运行），利用 **tasks_per_node** 设置每个节点的进程数（每个节点上有四个核，而且 POWER6 支持同步多线程(SMT)，因此最大可设置为 8，请结合自己程序的特点，设置为 4 或 8，以获取最高性能）。另外对 MPI 程序需采用 poe 的命令格式提交并行可执行程序，而对于 OpenMP 程序不应该使用 poe，应该通过设置 OMP_NUM_THREADS=8 或 4 来设置进程数。另外系统默认作业完成后，将发送信件给用户，这里设置了关键词 **notification = never**，表示作业完成后将不发送信件，还可设置为 always、error、start、complete。。

与串行作业一样，可使用下面方式提交：

llsubmit par_job.cmd

6.4 常用作业管理命令

用户常用的与作业相关的 LoadLeveler 命令主要有：

- **llcancel**：取消已存在的作业
- **llclass**：查询队列信息
- **llhold**：挂起一个作业
- **llmodify**：修改作业的运行参数
- **llstatus**：显示节点信息
- **llsubmit**：提交作业
- **llprio**：修改作业的优先级
- **llq**：显示作业状态等详细信息

下面只对最常用的做简单介绍，更多的相关命令及详细用法，请参考 Tivoli Workload Scheduler LoadLeveler V3.4: Using and Administering。

6.4.1 终止作业：llcancel

用户如果想终止一个作业，可以利用 **llcancel**，比如下面命令将终止 master.84.0 作业的运行：

llcancel master.84.0

6.4.2 查询队列信息：llclass

用户作业需要提交到特定队列（class）才能运行，查看可以使用的队列信息，可以利用 **llclass** 命令，其输出类似：

```
Name        MaxJobCPU     MaxProcCPU     Free   Max  Description
          d+hh:mm:ss    d+hh:mm:ss  Slots  Slots
--------  ------------  ------------  -----  -----  ----------------------
serial      undefined     undefined       2      3  low priority serial queue
medium      undefined     undefined     120    128  normal parallel queue
```

上面显示有两种队列 serial 和 medium 可以使用，分别可以允许运行的最大作业数目（Max Slots）为 3 和 128，当前空闲的数目（Free Slots）为 2 和 120。

结合参数 -c classname 可显示某个队列的信息，结合 -l 可显示队列的详细信息。

6.4.3 挂起作业和取消挂起作业：llhold

qhold 命令可以挂起作业，被挂起的作业将暂停执行，以让其余作业优先得到资源运行，被挂起的作业在用 **llq** 命令查询时显示的状态标志为 H，下面命令将挂起作业号为 master.84.0 的作业：

llhold master.84.0

如果想释放已经被挂起的作业 master.84.0，重新进入排队，可用下面命令：

llhold -r master.84.0

6.4.4 修改作业参数：llmodify

利用 **llmodify** 可以修改作业的队列类型、时间界限等，比如

llmodify -W 30 master.110.0

将时间限制扩大 30 分钟。

6.4.5 修改作业优先级：llprio

如果作业在提交时没有特别设置优先级，且作业所需的资源一致，那么作业的优先级将相同，按照先提交先运行的原则进行调度。例如有两个作业 master.110.0 和 master.111.0，master.110.0 先于 master.111.0 提交，运行 **llq** 显示：

```
Id                  Owner    Submitted    ST  PRI  Class      Running on
------------------- -------- ------------ --  ---  ---------- ----------
master.110.0        hmli      3/30 19:01  I   50   medium
master.111.0        hmli      3/30 19:02  I   50   medium
```

上面显示两者的优先级都为 50（PRI 列），如想让 master.111.0 先于 master.110.0 运行，可以利用 **llprio** 降低 master.110.0 的优先级或升高 master.111.0 的优先级，比如将 master.111.0 优先级增加 10：

llprio +10 master.111.0

此时再运行 **llq** 将显示：

```
Id                  Owner    Submitted    ST PRI Class      Running on
------------------- -------- ------------ -- --- ---------- ----------
master.111.0        hmli      3/30 19:02  I   60  medium
master.110.0        hmli      3/30 19:01  I   50  medium
```

6.4.6 查看队列中的作业状态：llq

用户想查看现在作业的运行状态，可以利用 **llq**，将给出类似下面的输出：

```
Id                  Owner    Submitted    ST PRI Class      Running on
------------------- -------- ------------ -- --- ---------- ----------
master.83.0         hmli      3/30 15:06  R   50  medium     node14
master.84.0         hmli      3/30 15:06  H   50  medium
master.85.0         hmli      3/30 15:07  I   50  medium
```

上面几列的含义分别为：作业号、用户名、提交时间、作业状态、优先级、资源名、运行程序的节点，其中作业状态中的 R、H 和 I 分别表示作业处于运行、被挂起和排队中。

如想查看 master.85.0 尚没运行的原因，可利用 **llq -l master.85.0** 查看：

```
......
         Unix Group: nic
Negotiator Messages: User = hmli has reached the maximum number jobs
                           allowed running.
      Bulk Transfer: No
......
```

Negotiator Messages 一行显示用户 hmli 已经到达最大可运行的数目

或者利用 **llq -s master.84.0** 也可以查看挂起的原因：

```
......
==================== EVALUATIONS FOR JOB STEP master.84.0 ====================

The status of job step is : User Hold
Since job step status is not Idle, Not Queued, or Deferred, no attempt has
been made to determine why this job step has not been started.
......
```

上面 Status: User Hold 显示作业没有运行的原因是作业被用户自己挂起。

6.4.7 显示节点状态：llstatus

llstatus 可以显示当前各节点状态，其输出类似：

```
Name      Schedd  InQ  Act  Startd  Run  LdAvg  Idle  Arch    OpSys
master    Avail    3    2    Idle    0   0.03   9999  R6000   AIX53
node01    Avail    0    0    Idle    0   0.00   9999  R6000   AIX53
......    .....    .    .    ....    .   ....   ....  .....   .....
node14    Avail    0    0    Run     4   3.90   9999  R6000   AIX53
node15    Avail    0    0    Run     8   7.96   9999  R6000   AIX53

R6000/AIX53     16 machines       3  jobs      20  running tasks
Total Machines  16 machines       3  jobs      20  running tasks

The Central Manager is defined on master

The BACKFILL scheduler is in use
```

用户比较关心的是 LdAvg 一列，显示的是当前的负载，应该与用户指定的进程数差不多，如果严重偏小，说明利用率不高，此时最好查找原因，看看瓶颈在哪里。如果比指定的进程数大很多，有可能是有用户没通过作业管理系统而是直接到节点上运行作业，可以 **rsh** 节点名进入此节点，并运行 **topas**命令（AIX 系统下无 **top** 命令，对应的是 **topas**）看看是哪个进程，哪个用户在违规使用，如发现此问题，请联系我们。

7 相关文档

- POWER6 处理器：http://www-03.ibm.com/technology/ges/semiconductor/power/index.html
- 编译器：
  - XL C/C++：
    - http://www-306.ibm.com/software/awdtools/xlcpp/library/
    - http://www-306.ibm.com/software/awdtools/xlcpp/support/
  - XL Fortran：
    - http://www-306.ibm.com/software/awdtools/fortran/xlfortran/library/
    - http://www-306.ibm.com/software/awdtools/fortran/xlfortran/support/
- 并行环境：
  - PE：http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.pe.doc/pebooks.html
- 数学函数库：
  - ESSL 和 PESSL：http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.essl.doc/esslbooks.html
- 作业调度：
  - LoadLeveler：http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.loadl.doc/llbooks.html
- IBM P 系列和 AIX 信息中心：http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp

8 技术支持

中国科学技术大学超级运算中心主页为：http://scc.ustc.edu.cn。超级运算中心将为用户提供相应的技术支持，如有需要请联系：

- 李会民
  - 电话：0551-3602248
  - 电邮：hmli@ustc.edu.cn