**Linpack测试与性能优化**

**目录：**

**一、Linpack介绍**

**二、计算峰值**

**三、Linpack安装与测试**

**四、影响Linpack性能的因素**

**五、Linpack性能的初步调优**

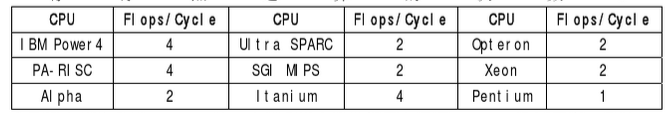
**一、Linpack介绍**

Linpack是国际上最流行的用于测试高性能计算机系统浮点性能的benchmark。通过对高性能计算机采用高斯消元法求解一元N次稠密线性代数方程组的测试，评价高性能计算机的浮点性能。  
Linpack测试包括三类，Linpack100、Linpack1000和HPL。Linpack100求解规模为100阶的稠密线性代数方程组，它只允许采用编译优化选项进行优化，不得更改代码，甚至代码中的注释也不得修改。Linpack1000要求求解1000阶的线性代数方程组，达到指定的精度要求，可以在不改变计算量的前提下做算法和代码上做优化。

HPL即High Performance Linpack，也叫高度并行计算基准测试，它对数组大小N没有限制，求解问题的规模可以改变，除基本算法（计算量）不可改变外，可以采用其它任何优化方法。前两种测试运行规模较小，已不是很适合现代计算机的发展。  
HPL是针对现代并行计算机提出的测试方式。用户在不修改任意测试程序的基础上，可以调节问题规模大小、使用CPU数目、使用各种优化方法等等来执行该测试程序，以获取最佳的性能。HPL采用高斯消元法求解线性方程组。求解问题规模为N时，浮点运算次数为(2/3 \* N^3－2\*N^2)。因此，只要给出问题规模N，测得系统计算时间T，峰值=计算量(2/3 \* N^3－2\*N^2)/计算时间T，测试结果以浮点运算每秒（Flops）给出。HPL测试结果是TOP500排名的重要依据。

**二、计算峰值**

衡量计算机性能的一个重要指标就是计算峰值或者浮点计算峰值，它是指计算机每秒钟能完成的浮点计算最大次数。包括理论浮点峰值和实测浮点峰值。  
理论浮点峰值是该计算机理论上能达到的每秒钟能完成浮点计算最大次数，它主要是由CPU的主频决定的。  
理论浮点峰值＝CPU主频×CPU每个时钟周期执行浮点运算的次数×系统中CPU数  
CPU每个时钟周期执行浮点运算的次数是由处理器中浮点运算单元的个数及每个浮点运算单元在每个时钟周期能处理几条浮点运算来决定的，下表是各种CPU的每个时钟周期执行浮点运算的次数。



实测浮点峰值是指Linpack值，也就是说在这台机器上运行Linpack测试程序，通过各种调优方法得到的最优的测试结果。在实际程序运行中，几乎不可能达到实测浮点峰值，更不用说理论浮点峰值了。这两个值只是作为衡量机器性能的一个指标。

**三、Linpack安装与测试**

**1．Linpack安装条件**

在安装HPL之前，系统中必须具备以下三个条件：

**1）编译器**

系统必须安装了支持C语言和Fortran 77语言的编译器。

**2）并行环境**

并行环境是指MPI。在以太网环境下，一般采用MPICH，当然也可以是其它版本的MPI，如LAM－MPI。在Myrinet网下，采用MPICH-GM。若是其它网络环境就采用相应的MPI。

**3）BLAS库**

BLAS库及基本线性代数库，采用BLAS库的性能对最终测得的Linpack性能有密切的关系。常用的BLAS库有GOTO、Atlas、ACML、ESSL、MKL等。

**2．安装与编译**

**1）**第一步，从[www.netlib.org/benchmark/hpl](http://www.netlib.org/benchmark/hpl) 网站上下载HPL源代码包hpl.tar.gz并解包。

**2）**第二步，编写Make文件。从hpl/setup目录下选择合适的Make.<arch>文件copy到hpl/目录下。修改make.test文件，主要修改的变量有：

ARCH：必须与文件名Make.<arch>中的<arch>一致

TOPdir：指明hpl程序所在的目录

MPdir： MPI所在的目录

MPlib： MPI库文件

LAdir： BLAS库或VSIPL库所在的目录

LAinc、LAlib：BLAS库或VSIPL库头文件、库文件

HPL\_OPTS：包含采用什么库、是否打印详细的时间、是否在L广播之前拷贝L

CC： C语言编译器

CCFLAGS： C编译选项

LINKER： Fortran 77编译器

LINKFLAGS：Fortran 77编译选项

**3）**第三步，编译。在hpl/目录下执行make arch=<arch>，<arch>即为Make.<arch>文件的后缀，生成可执行文件xhpl(在hpl/<arch>/bin目录下)

**3．运行**

运行hpl之前，需要修改配置文件hpl.dat(在hpl/<arch>/bin目录下)，次配置文件每一项代表的意思在文档第三部分说明。  
HPL的运行方式和MPI密切相关，不同的MPI在运行方面有一定的差别。对于MPICH来说主要有两种运行方法：  
1） 在hpl/<arch>/bin目录下执行：mpirun–np <N> xhpl。这种运行方式读取$(MPICH安装目录)/share/machines.LINUX配置文件  
2） 在hpl/<arch>/bin目录下执行：mpirun –p4pg <p4file> xhpl。这种运行方式需要自己编写配置文件<p4file>，以指定每个进程在哪个节点上运行  
MPICH要求至少有一个MPI进程在递交任务的节点上运行，但GM(MPI for Myrinet)、Infi－MPI(MPI for Infiniband)、ScaMPI(MPI for SCI)、BCL等MPI来说，没有这个要求。

对于GM来说，可以采用mpirun –machinefile <machinefile> -np <N> xhpl。这也是很多MPI所支持的一种运行方式，这种运行方式也需要自己编写<machinefile>以指定每个进程在哪个节点上运行。  
测试结果输出到指定文件中(在配置文件hpl.dat中定义)，缺省文件名为HPL.out。

**4．运行结果**

HPL允许一次顺序做多个不同配置测试，所以结果输出文件（缺省文件名为HPL.out）可能同时有多项测试结果。  
在文件的第一部分为配置文件hpl.dat的配置。在下面的部分  
使用基准测试一般需要和收集的信息包括：  
R: 它是系统的最大的理论峰值性能，按GFLOPS表示。

N: 给出有最高GFLOPS值的矩阵规模或问题规模。正如拇指规则，对于最好的性能，此数一般不高于总内存的80%。  
Rmax: 在Nmax规定的问题规模下，达到的最大GFLOPS。  
NB: 对于数据分配和计算粒度，HPL使用的块尺度NB。小心选择NB尺度。从数据分配的角度看，最小的NB应是理想的；但太小的NB值也可以限制计算性能。虽然最好值取决于系统的计算/通信性能比，但有代表性的良好块规模是32到256个间隔。

**四、影响Linpack性能的因素**

Linpack性能主要受三个因素的影响。分别是硬件因素、软件因素和HPL执行参数。

1）硬件因素

硬件因素主要包括cache大小和存储系统结构、访存速度、处理器性能、 计算机系统的结构以及互连网络的性能等，这些因素都会影响机器的Linpack性能。

2）软件因素

软件因素主要指的是MPI和BLAS对HPL性能的影响。MPI常用的有LAMMPI、MPICH和OpenMPI，这三种MPI的性能不一样，有些针对一些特殊的结构（如

SMP）会进行优化。

编译器的选择也有很大关系。

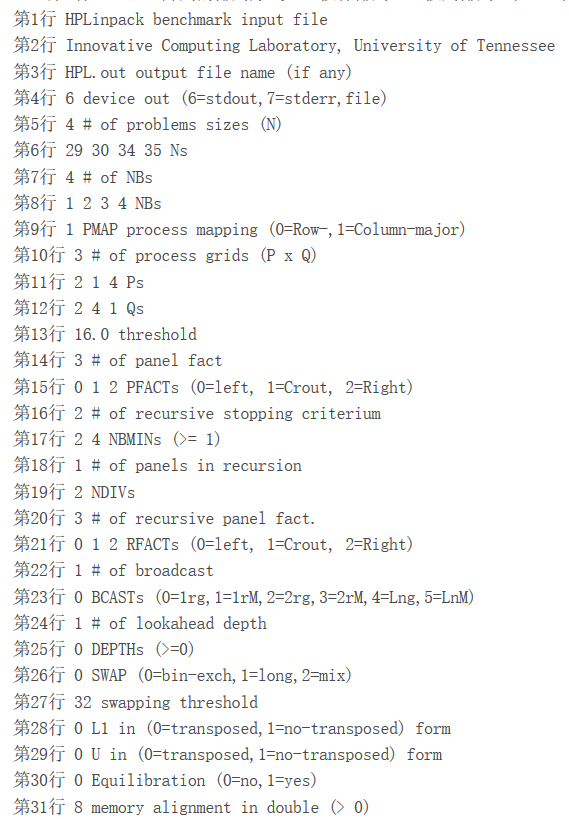
3）HPL执行参数

HPL执行参数很多， 在文件“HPL.dat” 中进行设置，其中对Linpack性能影响比较大的是Ｐ×Ｑ、Ｎ和NB。（PXQ是处理器网格的排列形式，Ｎ是矩阵规模，NB是矩阵分块大小）

由于硬件一般比较固定， 所以我们在做Linpack测试时主要调整的是软件以及HPL执行参数。

**五、Linpack性能的初步调优**

**1.对HPL配置文件（hpl.dat）进行调整：**



下面逐个简要说明每个参数的含义，及一般配置。  
**1) 第1、2行为注释说明行，不需要作修改**  
**2) 第3、4行说明输出结果文件的形式**

“device out”为“6”时，测试结果输出至标准输出（stdout）  
“device out”为“7”时，测试结果输出至标准错误输出（stderr）  
“device out”为其它值时，测试结果输出至第三行所指定的文件中  
**3) 第5、6行说明求解矩阵的大小N**矩阵的规模N越大，有效计算所占的比例也越大，系统浮点处理性能也就越高；但与此同时，矩阵规模N的增加会导致内存消耗量的增加，一旦系统实际内存空间不足，使用缓存，性能会大幅度降低。因此，对于一般系统而言，要尽量增大矩阵规模N的同时，又要保证不使用系统缓存。  
考虑到操作系统本身需要占用一定的内存，除了矩阵A（N×N）之外，HPL还有其它的内存开销，另外通信也需要占用一些缓存（具体占用的大小视不同的MPI而定）。一般来说，矩阵A占用系统总内存的80％左右为最佳，即N×N×8＝系统总内存×80％。  
这只是一个参考值，具体N最优选择还跟实际的软硬件环境密切相关。当整个系统规模较小、节点数较少、每个节点的内存较大时，N可以选择大一点。当整个系统规模较大、节点数较多、每个节点的内存较小时是，N可以选择大一点。

**4) 第7、8行说明求解矩阵分块的大小NB**  
为提高数据的局部性，从而提高整体性能，HPL采用分块矩阵的算法。分块的大小对性能有很大的影响，NB的选择和软硬件许多因数密切相关。NB大小的选择还跟通信方式、矩阵规模、网络、处理器速度等有关系。一般通过单节点或单CPU测试可以得到几个较好的NB值，但当系统规模增加、问题规模变大，有些NB取值所得性能会下降。所以最好在小规模测试是选择三个左右性能不错的NB，在通过大规模测试检验这些选择。  
**5) 第9行是HPL 1.0a的新增项，是选择处理器阵列是按列的排列方式还是按行的排列方式。**  
按HPL文档中介绍，按列的排列方式适用于节点数较多、每个节点内CPU数较少的瘦系统；而按行的排列方式适用于节点数较少、每个节点内CPU数较多的胖系统。在机群系统上，按列的排列方式的性能远好于按行的排列方式。

**6) 第10～12行说明二维处理器网格（P×Q）。二维处理器网格（P×Q）的要遵循以下几个要求：**

P×Q＝进程数。这是HPL的硬性规定。

P×Q＝系统CPU数＝进程数。一般来说一个进程对于一个CPU可以得到最佳性能。

当Q/4≤P≤Q时，性能较优。

当P＝2n，即P2的幂时，性能较优.

**7) 第13行说明阈值。**

这个值就是在做完线性方程组的求解以后，检测求解结果是否正确。若误差在这个值以内就是正确，否则错误。一般而言，若是求解错误，其误差非常大；若正确则很小。所以没有必要修改此值。

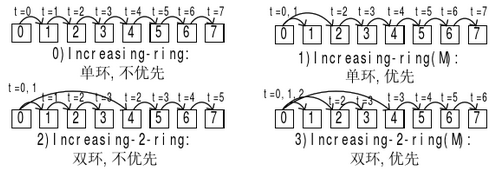
**8) 第14～21行指明L分解的方式**  
在消元过程中，HPL采用每次完成NB列的消元，然后更新后面的矩阵。这NB的消元就是L的分解。每次L的分解只在一列处理器中完成。  
对每一个小矩阵作消元时，都有三种算法：L、R、C，分别代表Left、Right和Crout。在LU分解中，具体的算法很多，HPL就采用了这三种。对这三种算法的具体描述可参考相关LU分解的资料，也可参加HPL的源代码。  
HPL中，L分解采用递归的方法，其伪代码如下：



**9) 第22、23行说明L的横向广播方式，HPL中共提供六种广播方式。**

其中前四种适合于快速网络；后两种采用将数据切割后传送的方式，主要适合于速度较慢的网络。目前，机群系统一般采用千兆以太网甚至Myrinet、Infiniband、SCI等高速网络，所以一般不采用后两种方式。

前四种算法如图所示，分别采用单环/双环、第一列处理器不优先/优先。



对于系统规模较小、处理器数（进程数）较少的系统来说，这四个选择对性能影响很小。  
对于横向处理器数Q较大的网络来来说，选择双环可以减少横向通信宽度，较小横向通信延迟。另外，第一列处理器优先算法也可以确保下一次L分解的尽早开始。

在小规模系统中，一般选择0或1；对于大规模系统，3是个不错的选择。

**10) 第24、25行说明横向通信的通信深度。**  
L的分解过程是一个相对比较耗时的过程，为了提高性能，其采用先作一部分分解，然后将这一部分结果广播出去。“DEPTHs”值就是说明将L分几次广播。DEPTHs＝0表明将L一次性广播出去，也就是将整个L分解完成以后在一次性广播；DEPTHs＝1表示将L分两次广播；依此类推。  
L分为多次广播可以使得下一列处理器尽早得到数据，尽早开始下一步分解。但这样会带来额外的系统开销和内存开销。DEPTHs的值每增加1，每个进程需要多申请约（N/Q＋N/P＋NB＋1）×NB×8的内存。这对HPL的开销是很大的，因为增加DEPTHs以后，为了保证不使用缓冲区，不得不减小问题规模N的值，所以在N和DEPTHs需要作一个权衡。在小规模系统中，DEPTHs一般选择1或2；对于大规模系统，选择2～5之间。

**11) 第26、27行说明U的广播算法。**  
U的广播为列向广播，HPL共提供了三种U的广播算法：二元交换法、Long法和二者混合法。SWAP=“0”，采用二元交换法；SWAP=“1”，采用Long法；SWAP=“2”，采用混合法。  
二元交换法的通信开销为㏒2P×（Latency＋NB×LocQ（N）/Bandwith），适用于通信量较小的情况；Long法的通信开销为（㏒2P＋P－1）×Latency＋K×NB×LocQ（N）/Bandwith适用于通信量较大的情况。其中P为列向处理器数，Latency为网络延迟，Bandwith为网络带宽，K为常数，其经验值约为2.4。LocQ（N）＝NB×NN为通信量，NN随着求解过程的进行逐步减少。  
由于NN在求解过程中在不断的变化，为了充分发挥两种算法的优势，HPL提供了混合法，当NN≤swapping threshold（第27行指定）时，采用二元交换；否则采用Long法。  
一般来说，我们选择混合法，阈值可通过公式求得一个大概值。对于小规模系统来说，此值性能影响不大，采用其缺省值即可。  
**12) 第28、29行分别说明L和U的数据存放格式。**C语言矩阵在内存是按行存放的，Fortran语言是按列存放的。由于HPL采用C语言书写，而调用的BLAS库有可能采用C语言，也有可能采用Fortran语言编写。  
若选择“transposed”，则采用按列存放，否则按行存放。我没有对这两个选项进行性能测试，而选择按列存放。因为感觉按列存放是，性能会好一些。  
**13) 第30行主要是在回代中用到。**

**14) 第31行的值主要为内存地址对齐而设置，主要是在内存分配中作地址对齐而用。**

**2.其他性能优化：**

除了对HPL配置文件进行调整外，还有其它很多的优化方法。

**1） MPI**  
对于常用的MPICH来说，安装编译MPICH时，使其节点内采用共享内存进行通信可以提升一部分性能，在configure时，设置“—with-comm=shared”。  
对于GM来说，在找到路由以后，将每个节点的gm\_mapper进程kill掉，大概有一个百分点的性能提高。当然也可以采用指定路由表的方式启动GM。

**2）编译优化**

采用编译优化可以在很大程度上提高CPU密集型应用程序的性能，如采用超长指令可以较大程度的提高浮点数处理性能。编译优化在不修改程序的条件下主要有两种方法：采用性能较好的编译器和增加编译优化选项。在X86平台上，主要编译器有GNU、Intel编译器、PGI编译器等，在一些专门的平台上有专门的编译器，如IBM的P系列机器上有其专有的编译器xlc和xlf。编译优化选项和编译器密切相关，不同的编译器编译优化选项不尽相同。  
在HPL测试中，编译优化对其性能影响不大。原因是HPL将计算最密集部分的都通过调用BLAS库完成，在HPL本身的程序中，作数值计算的几乎没有。