第36卷 第19期 计 算 机 工 程 2010年10月
Vol.36 No.19 Computer Engineering October 2010
·软件技术与数据库· 文章编号：1000—3428(2010)19—0047—03 文献标识码：A 中图分类号：TP393

基于混合精度算法的改进 HPL 软件包

王 磊[1,2,3]，张云泉[1,2]，刘芳芳[1]，张先轶[1]
(1. 中国科学院软件所并行计算实验室，北京 100190；2. 中国科学院计算机科学国家重点实验室，北京 100190；
3. 中国科学院研究生院，北京 100190)

摘 要：利用求解线性方程组的混合精度算法，对 HPL 软件包进行改进。从性能与加速比、迭代时间与迭代次数以及误差分析 3 个方面，在四路 AMD Opteron870 双核处理器平台上，对原 HPL 与改进的 HPL 软件包进行对比测试。实验结果表明，改进的 HPL 软件包在保证双精度浮点精度要求的前提下，计算性能大约提高 1 倍，并具有良好的可扩展性。

关键词：混合精度算法；HPL 软件包；加速比

Improved HPL Software Package Based on Mixed Precision Algorithm

WANG Lei[1,2,3], ZHANG Yun-quan[1,2], LIU Fang-fang[1], ZHANG Xian-yi[1]
(1. Lab of Parallel Computing, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China;
2. State Key Lab of Computer Science, Chinese Academy of Sciences, Beijing 100190, China;
3. Graduate University of Chinese Academy of Sciences, Beijing 100190, China)

【Abstract】This paper improves a High Performance Linpack(HPL) software package by using mixed precision algorithm to solve linear equations set. The performance test which including performance and speedup, the number and time of iteration and error analysis for original HPL and improved HPL software package are conducted on the platform of four AMD Opteron870 dual-core processors. Experimental results show the computing performance of improved software package enhances almost twice compared with the original HPL while keeping double floating point precision and it also has good scalability.

【Key words】mixed precision algorithm; High Performance Linpack(HPL) software package; speedup ratio

1 概述

HPL(High Performance Linpack)[1]是第 1 个标准的并行 Linpack 测试软件包。全球 TOP500 排名以及国内的 TOP100 排名，都是根据 Linpack 的测试结果制定的。随着计算机体系结构的不断发展，在当代计算机体系结构上，32 位单精度浮点操作的性能至少是 64 位双精度浮点操作性能的 2 倍[2]。虽然单精度浮点操作性能突出，但其可以表示的精度范围有限，通常不能满足应用程序的需求。混合精度算法通过混合利用单精度浮点操作和双精度浮点操作，能够在满足双精度浮点操作精度要求的前提下，提高程序的性能。本文利用混合精度算法对 HPL 软件包进行改进，获得了很好的性能提升。国际上对于修改后的 Linpack 测试程序是否能作为测试标准还处于探讨阶段，因此本文工作对于 Linpack 测试标准是一次探索。

2 相关工作

混合精度算法的思想为：在多数情况下，一个利用单精度浮点操作得到的结果，通过不断求精通常可以达到双精度浮点操作的精度范围。该思想很容易应用于求解线性方程组问题，下文为文献[2-4]介绍的利用直接法求解稠密线性方程组的混合精度算法。

混合精度算法首先利用单精度操作的直接法求出一个初步解 x_0，然后利用双精度浮点操作对初步解进行不断迭代求精 $(x_1, x_2, \cdots, x_k)$，直到满足收敛性的要求。其算法过程如下：

(1)利用单精度的 LU 分解算法计算 $\boldsymbol{LU} \leftarrow \boldsymbol{PA}$；

(2)利用单精度算法求解 $\boldsymbol{Ly} = \boldsymbol{Pb}$；

(3)利用单精度算法求解 $\boldsymbol{U}x_0 = \boldsymbol{y}$，得到初步解 x_0；

(4)利用双精度浮点操作计算 $r_k \leftarrow \boldsymbol{b} - \boldsymbol{A}x_{k-1}$ $(k=1,2,\cdots,n)$；

(5)利用单精度算法求解 $\boldsymbol{Ly} = \boldsymbol{P}r_k$；

(6)利用单精度算法求解 $\boldsymbol{U}z_k = \boldsymbol{y}$；

(7)利用双精度浮点操作更新当前解 $x_k \leftarrow x_{k-1} + z_k$，计算收敛性判断结果，如果已达到收敛范围，则结束，否则转(4)继续计算。

由于 HPL 只提供利用双精度操作求解方程组的实现，并且为了性能考虑有很多地方不能直接被混合精度算法利用，因此本文主要工作是分析 HPL 的实现，并且利用上述混合精度算法进行改进，从而提高性能。

基金项目：国家自然科学基金资助项目(60303020)；国家自然科学基金资助重点项目(60533020)；国家“863”计划基金资助项目(2006AA01A102, 2006AA01A125)

作者简介：王 磊(1984－)，男，硕士研究生，主研方向：并行算法，并行软件开发；张云泉，研究员、博士；刘芳芳，助理研究员、硕士；张先轶，助理研究员、硕士

收稿日期：2010-04-11 **E-mail：**lei.beststones@gmail.com

3 HPL 软件包实现

HPL 软件包中实现了 Left-Looking、Right-Looking、Crout 3 种 LU 分解算法[5]，并且用户可以选择是否利用 Look-ahead 技术和递归分块技术执行不同例程。图 1 为 HPL 软件包流程。

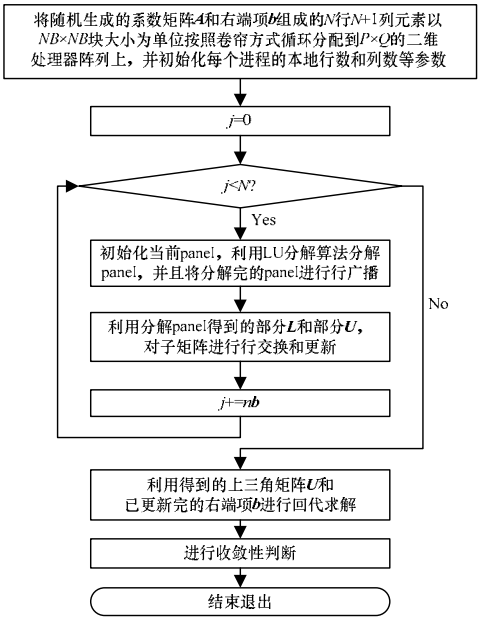

图 1 HPL 软件包流程

以 Right-Looking LU 分解算法为例进行说明。HPL 中 Right-Looking LU 分解算法的实现虽然完成了 LU 分解的功能，但是出于性能考虑其实现没有完全遵从 Right-Looking LU 分解算法进行，在此主要有 2 点说明：(1)HPL 在每次分解完 panel 后，由于不对 1~k×NB(设块大小为 NB)列进行行交换而不会影响后面的分解，因此出于性能考虑，HPL 程序没有对全局矩阵的前 k×NB 列进行行交换，这样最终得到的 $\boldsymbol{L}$ 下三角矩阵并不是正常 LU 分解得到的 $\boldsymbol{L}$。由于每次分解得到的行交换信息会被下次分解的 panel 信息冲掉，因此得不到最终的行交换信息 $\boldsymbol{P}$。因为在混合精度算法的(5)中需要用到 $\boldsymbol{L}$ 和 $\boldsymbol{P}$，所以在后面的混合精度算法实现中对此进行了改进。(2)HPL 在 LU 分解的过程中同时将 $\boldsymbol{b}$ 进行了更新，即 HPL 用一个函数完成了混合精度算法中的(1)和(2)，而(2)在混合精度算法的(5)中还要使用，所以，本文混合精度算法中实现了一个完成(5)功能的函数 HPL_pdtrsv_L。

4 基于混合精度算法的改进 HPL 软件包

4.1 初步实现

由于混合精度算法首先需要利用单精度浮点操作求解线性方程组，因此首先要将 HPL 中双精度浮点操作转换成单精度浮点操作，主要是将 HPL 中的数据结构和变量等转换成单精度类型。对底层 BLAS 双精度子程序的调用改成调用单精度子程序。用于收敛性判断的相关数据结构和函数不需要修改。由于后面的迭代求精过程中需要用到相关的双精度浮点操作，因此一些数据结构和子程序需要提供双精度版本。本文自定义了存储双精度浮点矩阵的 HPL_T_dpmat 数据结构，提供的双精度子程序包括：随机生成双精度浮点矩阵的 HPL_pddmatgen，调用 BLAS 中双精度操作的 HPL_ddgemv、HPL_ddgemm、HPL_ddtrsm 等函数以及通信函数 HPL_recv_D、HPL_send_D、HPL_broadcast_D、HPL_all_reduce_D、HPL_sum_D、HPL_max_D 等。

4.2 $\boldsymbol{L}$ 矩阵修正和行交换信息的保存

线性方程组求解算法中提到 HPL 中得到的 $\boldsymbol{L}$ 不是真正 LU 分解得到的 $\boldsymbol{L}$，并且分解过程中没有保留行交换数组 P，由于这 2 个信息要被混合精度算法的(5)所用，因此本文对其进行了以下改进：

(1)为每个进程增加了一个数组 ipiv 存储行交换数组 $\boldsymbol{P}$；

(2)修改 HPL_pdgesv0 和 HPL_pdgesvk2 2 个函数，在每次分解完 panel 后将相应的行交换信息存储到进程的 ipiv 数组中，然后利用该信息对 $\boldsymbol{L}$ 进行行交换更新。

4.3 $\boldsymbol{Ly=Pb}$ 函数求解

由于 HPL 线性方程组求解算法中的 HPL 没有提供求解 $\boldsymbol{L}y = Pb$ 的函数，因此本文首先利用 4.2 节中求得的 ipiv 对 $\boldsymbol{b}$ 进行行交换，然后利用 $\boldsymbol{L}$ 实现完成余下操作的 HPL_pdtrsv_L 函数。

如图 2 所示，该操作的过程类似于分解 panel 的过程，以 NB 为步长进行操作。假设以当前对角线上的块 L_{kk} 进行操作，调用 HPL_dtrsv 函数求解 L_{kk} 所在行对应的右端项 $\boldsymbol{b}$ 的部分。之后将得到的部分 $\boldsymbol{b}$ 元素传递给当前列下面各块所在的进程，当前列下面各块所在的进程调用 HPL_dgemv 函数更新其所在行对应的部分 b 元素。然后再将更新完的部分 b 元素进行行广播，这时下一列的各块可以执行同样的更新操作。这样依次类推直到 L_{nn}。

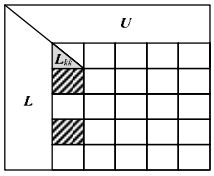

图 2 pdtrsv_L 函数执行过程

为性能考虑，本文采用 Look-ahead 技术，首先将对角线块 L_{kk} 计算完后，将得到的结果先传递给当前列下一个 NB 大小的块所在的进程(即含有斜纹块的进程)，而不是一次传给当前列的其他所有进程。这个进程先更新其部分 b 然后广播给右侧列所在的进程，这时 $L_{(k+1)(k+1)}$ 块就可以更新了，同时 L_{kk} 块所在进程将之前得到的结果传递给当前列其余进程进行更新，这样就能同时进行 2 个部分的操作，从而提高性能。

4.4 迭代过程和收敛性判断

混合精度算法的迭代过程按照混合精度算法的(4)~(7)来实现，需要利用上述的改进部分，收敛性判断采用 HPL 中的收敛性判断标准，即判断：

$$\|\boldsymbol{A}X-b\|_\infty/(eps\times(\|X\|_\infty\times\|\boldsymbol{A}\|_\infty+\|b\|_\infty)\times N)<16.0$$

是否成立，收敛性判断的所有操作都使用双精度浮点操作完成。其中，eps=1.110 223e^{-16}。

5 实验结果与分析

5.1 实验环境

本文实验平台为曙光天阔服务器 S4800A1，机器配置为四路 AMD Opteron870 双核处理器，2.0 GHz 主频、16 GB DDR 内存、2.1 TB SCSI 硬盘，操作系统为 Turbo Linux 3.4.3、编译器、MPI 和 HPL 的版本分别为 gcc3.4.3、mpich-1.2.7 和 HPL2.0，BLAS 选择标准 BLAS 软件包。

5.2 实验

本文在目标平台上使用 HPL 内部测试程序进行测试，设定矩阵规模的范围为 1 000~10 000 之间，步长为 1 000，块大小 NB 为 64，随机生成的系数矩阵元素在-0.5~0.5 之间。为

了叙述方便本文将改进后的 HPL 命名为 MHPL。

5.2.1　性能与加速比

图 3 为 *NB*=64、进程阵列为 2×4 时，不同矩阵规模时 MHPL 与 HPL 的加速比比较。图 4 为 *NB*=64、矩阵规模为 8 000 时，不同进程阵列时 MHPL 与 HPL 的时间比较。图 5 为 *NB*=64、矩阵规模为 8 000 时，不同进程阵列时 MHPL 与 HPL 的加速比比较。

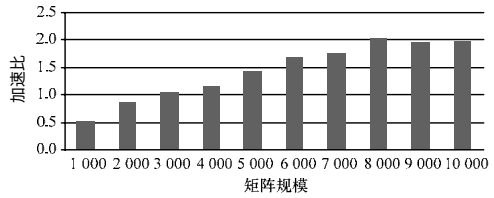

图 3　不同矩阵规模时的加速比比较

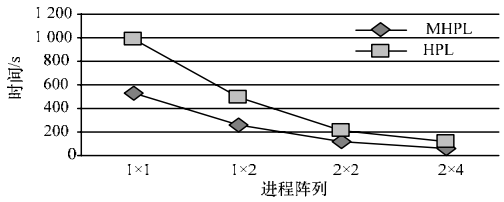

图 4　不同进程阵列时 MHPL 与 HPL 的时间比较

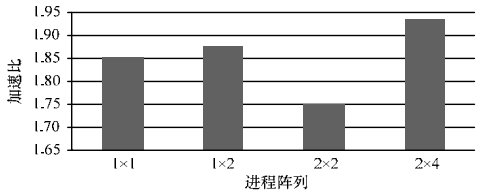

图 5　不同进程阵列时的加速比比较

从图 3 可以看出，MHPL 随着矩阵规模的增大加速比近似增大，但是在矩阵规模较小时加速比会出现低于 1 的情况，其原因是在矩阵规模较小时，迭代求精所占用的时间与矩阵分解的时间基本相当，所以，在矩阵规模较小时使用混合精度算法反而会降低程序的性能。图 4 和图 5 是在不同进程阵列下，MHPL 与 HPL 之间的性能比较和加速比，可以看出 MHPL 具有良好的可扩展性。

5.2.2　迭代时间与迭代次数

当 *NB*=64、进程阵列为 2×4 时，图 6 为 MHPL 迭代求精时间占整体时间的比率，图 7 为 MHPL 迭代次数。

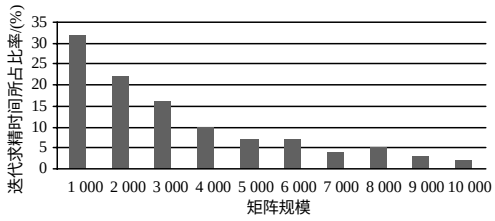

图 6　MHPL 迭代求精时间占整体时间的比率

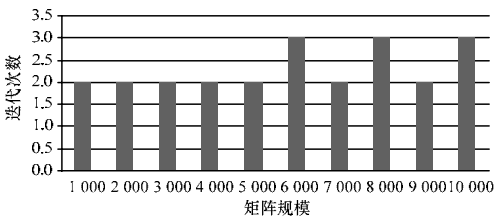

图 7　MHPL 迭代次数

从图 7 可以看出，在矩阵规模为 1 000~10 000 之间时，迭代求精的平均次数为 2 次，并且由于矩阵分解的计算量为 $O(n^3)$ 量级，而迭代求精的计算量为 $O(n^2)$ 量级，因此在图 6 中随着矩阵规模的增大迭代求精所用的时间占整体时间的百分比会逐渐下降，当矩阵规模达到一定程度时基本可以忽略不计。

5.2.3　误差分析

文献[6-7]对混合精度求解线性方程组算法的舍入误差进行了分析，并得出了如下结论：只要系数矩阵的条件数不大于单精度运算条件的倒数，迭代求精过程就会在有限步后收敛。表 1 为 *NB*=64、矩阵规模为 8 000 时，MHPL 和 HPL 收敛性判断结果比较，从中可以看出 MHPL 的误差明显大于 HPL。由此可知，在某些情况下，混合精度算法并不适用，此时需要使用双精度浮点操作完成全部运算。

表 1　MHPL 和 HPL 的收敛性判断结果比较

进程阵列($P\times Q$)	MHPL	HPL
1×1	0.463 625 3	0.005 361 7
1×2	0.456 813 4	0.005 778 1
2×2	0.456 813 4	0.005 778 1
2×4	0.380 446 2	0.004 913 5

6　结束语

本文利用混合精度算法对 HPL 软件包进行改进，在 AMD Opteron870 处理器平台上，改进后的 HPL 在矩阵规模为 1 000~10 000 之间时平均迭代次数为 2 次，与 HPL 的平均加速比为 2，其性能近似于只使用单精度浮点操作的性能，同时具有良好的可扩展性。然而通过误差分析可知，改进后的 HPL 虽然达到了双精度的收敛要求，但其误差明显大于只使用双精度浮点操作的 HPL，所以，在某些情况下混合精度算法并不适用。混合精度 HPL 的性能主要取决于机器单精度浮点计算的性能，对于目前主流的 GPU，其单精度浮点峰值是双精度浮点峰值的 10 倍多。下一步工作为：将混合精度算法的 HPL 移植到 GPU 上提升其性能。

参考文献

[1] Petitet A, Whaley R C, Dongarra J. HPL — — A Portable Implementation of the High-performance Linpack Benchmark for Distributed-memory Computers[EB/OL]. (2008-09-10). http://www.netlib.org/benchmark/hpl/.

[2] Baboulin M, Buttari A, Dongarra J, et al. Accerlating Scientific Computations with Mixed Precision Algorithms[J]. Computer Physics Communications, 2009, 180(12): 2526-2533.

[3] Demmel J W. Applied Numerical Linear Algebra[M]. 1st ed. [S. l.]: SIAM, 1997.

[4] Wilkinson J H. Rounding Errors in Algebraic Processes[M]. [S. l.]: Prentice Hall Press, 1963.

[5] Dongarra J, Luszczek P. How Elegant Code Evolves with Hardware: the Case of Gaussian Elimination[M]. [S. l.]: O'Reilly & Associates Inc., 2007.

[6] Langou J, Luszczek P, Kurzak J. Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy[C]// Proc. of 2006 ACM/IEEE Conference on Supercomputing. Tampa, Florida, USA: [s. n.], 2006.

[7] Arioli M, Duff I S, Using FGMRES to Obtain Backward Stability in Mixed Precision[R]. [S. l.]: Rutherford Appleton Laboratory. Technical Report: RAL-TR-2008-006, 2008.

编辑　陆燕菲