EFFICIENT DETERMINATION OF BLOCK SIZE NB FOR PARALLEL LINPACK TEST

Zhang Wenli [1, 2] Fan Jianping [1] Chen Mingyu [1]
[1] National Research Center for Intelligent Computing Systems,
Institute of Computing Technology, Chinese Academy of Sciences
[2] Graduate School of the Chinese Academy of Sciences
NCIC, P.O. Box 2704, Beijing, P.R. China, 100080
Email: zhang_jlu@etang.com

ABSTRACT
HPL is a Linpack benchmark package widely used in massive cluster system performance test. Based on in-depth analysis of the blocked parallel solution algorithm of linear algebra equations and HPL implementation mechanics, the restriction factors of the HPL peak performance are probed. The influence of main parameters P×Q and NB on computing performance in LU factorization process is discussed especially. Theory analysis and experiment results indicate that, the efficiency factor, defined as the ratio of the matrix operation time and the amount of operations, is relevant to the matrix block size to a great extent, yet little correlate to the matrix size itself. According to this law, the author proposes to determine the block size NB of massively parallel test through scanning the operation efficiency of proper small-scale matrix. Thus it improves the status getting NB through trial-and-error experiment without definite aim. The author's idea has been verified by the real test results, as well as been applicable to the matrix parallel operations of other courses.

KEY WORDS
high performance Linpack, linear algebra equations, LU factorization, MPI

1. Introduction

Linpack is a prevailing performance test benchmark at present. Through testing the power of the high performance computer to solve dense linear algebra equations, Linpack evaluated the floating-point operation capability of the high performance computer system. It was put forward for the first time in 1979 by Jack Dongarra, mostly Fortran edition. It offers many kinds of programs and solves the linear equation problems, including solving dense matrix operation, belt linear equations, least squares issue and other various kinds of matrix operations, which are all based on principles of Gaussian Elimination, supported by other function libraries.

Linpack is divided into three magnitudes[1] of 100×100, 1000×1000 and n×n according to problem size and chosen optimization. HPL[2] (high performance Linpack) is the first open standard parallel edition Linpack testing package, MPI implementation of n × n test, easily portable to many systems, and used in Top500[3] test widely at present. The test is designed mainly aiming at large-scale distributed-memory parallel computing. For its loosest request in Linpack standard, users can use any number of CPUs to any problem size, use various kinds of optimization methods based on Gaussian Elimination to carry out the test to seek the best results.

Performance test is actually to calculate the floating-point operation rate. For the amount of floating-point operations ($2\ n^3/3\ +3\ n^2/2$) is certain, only relating to the problem size, i.e. the dimension of coefficient matrix, in solving scale n linear algebra equations by Gaussian Elimination. Thus based on the time t of factorization and backward substitution recorded during linear equations solving course, only the scale n is required to calculate the performance parameter of machine — Gflops:

$$(2\boldsymbol{n}^3/3+3\boldsymbol{n}^2/2\ )\ /\ \boldsymbol{t} \tag{1}$$

Generally, to obtain HPL peak value, the problem size should be as large as matching the memory size, that is to be probably close to 80% of the total capacity of the memory.

The author of this paper weighs the key adjustable parameters on the basis of the HPL source code analysis in detail, launches relevant experiments, analyzes results, and draws some useful conclusions, hoping to have some directive significance to follow-up linpack test and optimization. Section 2 of this paper briefly addresses principle of Gaussian Elimination and parallel thought of LU algorithm, section 3 discusses on thinking and attempt to the relevant problems in the course of analysis and testing, especially the discussion on the basis to determine

This research is supported by Chinese National High-tech Research and Development (863) Program (grants 2003AA 1Z2070) and by the foundation of Knowledge Innovation Program (grants 20036040), Chinese Academy of Sciences (CAS).

block size NB of matrix, then provides verified test results and does corresponding analysis. Finally section 4 concludes and presents further work.

2. Principle of Gaussian Elimination

The numeric solution of the linear algebra equations is divided into direct and iterative methods. To the dense coefficient matrix of linear algebra equations

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{2}$$

where $\mathbf{A}=(a_{ij})_{n\times n}$ is a non-singular matrix; $\mathbf{b}=(b_1, b_2, \ldots, b_n)^T$, $\mathbf{x}=(x_1, x_2, \ldots, x_n)^T$, A and b are given, and x is the n-vector to be solved, LU factorization[4, 5, 6] is a classical direct solution method. And Gaussian Elimination is a classical algorithm to realize LU factorization, through factorization and backward substitute course to set up concrete solving.

That is to say, LU factorization transforms issue (1) to solving two triangular equations

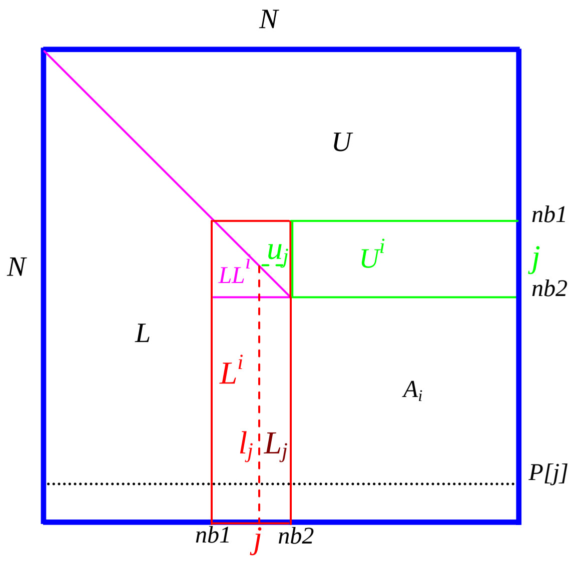
$$\boldsymbol{LUx} = \boldsymbol{Pb}.$$

Equivalent to

$$\boldsymbol{Ly} = \boldsymbol{Pb} \tag{3}$$

$$\boldsymbol{Ux} = \boldsymbol{y} \tag{4}$$

where P is a row permutation elementary matrix, equation (3) is operated simultaneously with LU factorization process among HPL, $\mathbf{y}=(y_1, y_2, \ldots, y_n)^T$ is corresponding part of b in augmented matrix after factorization, equation (4) shows the backward substitute solving course.



```
[1] nb1=i×NB, nb2=nb1+NB
[2] for(j=nb1; j<nb2; j++)
    [2.1] Find pivot, P[j] s.t. $|L^i_{P[j],j}| >= |L^i_{j:N,j}|$
    [2.2] $L^i_{j,\,nb1:nb2} \leftrightarrow L^i_{P[j],\,nb1:nb2}$
    [2.3] $l_j \leftarrow l_j / L^i_{j,j}$
    [2.4] $L_j \leftarrow L_j - l_j u_j$
[3] Broadcast $LL^i$, $L^i$ and row exchange information
      among processors in row direction
[4] for (j=nb1; j<nb2; j++)
    [4.1] $A^i_{j,\,nb2:N} \leftrightarrow A^i_{P[j],\,nb2:N}$ (i.e. columns after $Li$)
[5] $U^i \leftarrow (LL^i)^{-1} U^i$
[6] $A^i \leftarrow A^i - L^i U^i$
```

Fig. 1. LU factorization process of blocked matrix

According to the analysis of the above-mentioned algorithm course, it is apt to know that the operation amount in LU factorization involves A(i+1:n,i)/A(i,i) and A(i+1:n,i+1:n)-A(i+1:n,i)*A(i,i+1:n), that is

$$\Sigma_{\boldsymbol{i}=\mathbf{1}\sim(\boldsymbol{n}-\mathbf{1})} \left[(\boldsymbol{n}-\boldsymbol{i}) + 2\times(\boldsymbol{n}-\boldsymbol{i})^2 \right] = (2/3)\times \boldsymbol{n}^3 + \mathbf{O}(\boldsymbol{n}^2).$$

And the operation amount of solution (3) and (4) separately is $O(n^2)$, so the total operation amount remains $(2/3)\times n^3 + O(n^2)$.

Clearly, LU factorization is the main part of linear equations solution, accounting for $O(n^3)$. Moreover in serial factorization, the amount of computing is mainly at matrix updating, that is A[i][k]-A[i][j]×A[j][k]. So the key task for parallelism is to update different parts of matrix A as simultaneously as possible on multiprocessors. The two-dimensional block cyclic layout of dense matrix on processors is conformed after series of analysis and comparison[7]. On one hand, data blocking layout can take advantage of memory level on processor and get better operation efficiency in processor computing by BLAS3[8], on the other hand, cyclic distribution is to dispatch task among processors as rational as possible, balancing load. Thus the determination of NB becomes essential.

Dense matrix $A_{N\times(N+1)}$ created at random is divided into NB×NB blocks, cyclic distributing on p=P×Q two-dimensional processor array. Each process handles (N/P)×(N/Q) approximately blocks stored locally. For convenience, processors are numbered from 0 to p-1, and matrix columns (or rows) are denoted from 0 to N+1 (or N). Then to element A(m, n) of matrix A, it is distributed on p(i, j), shown with the equation as follows,

$$\boldsymbol{i} = (m / NB) \bmod P,\ \boldsymbol{j} = (n / NB) \bmod Q$$

That is, $p = p(\boldsymbol{i}, \boldsymbol{j}) = \boldsymbol{i} + \boldsymbol{j}\, P$.

To HPL, even assume that the load of each processor can guarantee the CPU power totally use, the unavoidable communication expenses and strategy redundancy computing restrict the peak parallel performance to some extent. Some quantitative analysis to the expenses in the course of factorization is done according to the pseudo code shown in Fig. 1.

Suppose α =Communication latency, β =1/bandwidth, L =Communication package size, communication time can be defined as

$$T = \alpha + \beta L.$$

Set efficiency factor of matrix operation as γ, namely the ratio of matrix operation time and corresponding operation amount. The smaller its value is, the higher operation efficiency is.

First of all, take one Panel, NB columns, as the research object to estimate the processor overhead of parallel operation. Suppose jj = j % NB to be column number local in the panel.

During factorization, each search of the pivot **[2.1]**, simultaneously completing row permutation of NB column in Panel **[2.2]**, costs $(\text{Log}_2P)(\alpha + \beta(2 \times NB + 4))$ communication amount. After permutation of the pivot to the diagonal position of matrix, utilizing redundant storage, each processor calculates *lj* **[2.3]** at the cost of (N – *j*) computing amount in all, updates the following columns in the panel **[2.4]** by the computing amount of 2(N-j)(NB-jj). Do row communication **[3]** by one of six communication models[2], which is mainly involved in the consideration for overcoming the network bandwidth restriction and processor load problem. Take increase ring modified method as an example, each time it need $\alpha + \beta((N - i \times NB) \times NB) / P$ communication amount, then the rest of matrix do row permutation **[4]** according to obtained information, that is to communicate pseudo U^i in column direction to other P-1 processors of the same column. Here take long method as an example, the $\alpha(\text{Log}_2P + P - 1) + 3\beta(N - (i + 1)NB) \times NB / Q$ communication is needed. Each processor does **[5]** to get the U^i needed by updating through redundancy storage at one time, costs $(N-(i+1)NB) \times NB^2/Q$. The communication latency of broadcast after computing for individual processor is avoided through redundancy computing. Do **[6]** to complete the rest part of matrix updating, which costs $2(N-(i+1)NB)^2 \times NB/(PQ)$ computing amount.

To sum up roughly, the overhead of handling Panel *i* is

$$\begin{aligned}T_{Factor} &= T_{Calc\ L} + T_{Find\ Pivot} + T_{Bcast\ L}\\ &= \gamma_f((N-i\times NB)/P-NB/3)\times NB^2\\ &\quad + NB(\text{Log}_2P)(\alpha + \beta(2\times NB+4))\\ &\quad + \alpha + \beta(N-i\times NB)NB/P\end{aligned}$$

$$\begin{aligned}T_{Update} &= T_{Calc\ U} + T_{Rest\ Update} + T_{Row\ Exchange}\\ &= \gamma_u((N-(i+1)NB)\times NB^2/Q\\ &\quad +2(N-(i+1)NB)^2\times NB/(PQ))\\ &\quad +\alpha(\text{Log}_2P+P-1) + 3\beta(N-(i+1)NB)\times NB/Q\end{aligned}$$

The backward substitute course of HPL is almost serial operation, and its proportion seldom exceeds 5% in general, so it is not usually considered too much in parallel optimization.

To sum up, it is easily found out that, in parallel algorithm there are impassable hindrance for improving multiprocessor efficiency, including P-1 time redundancy computing Rc for U and three parts of communication relevant expenses of $Tc = T_{Find\ Pivot} + T_{Bcast\ L} + T_{Row\ Exchange}$. The real performance of system should be revised as:

$$(2n^3/3 + 3n^2/2)+Rc) / (t- Tc)$$

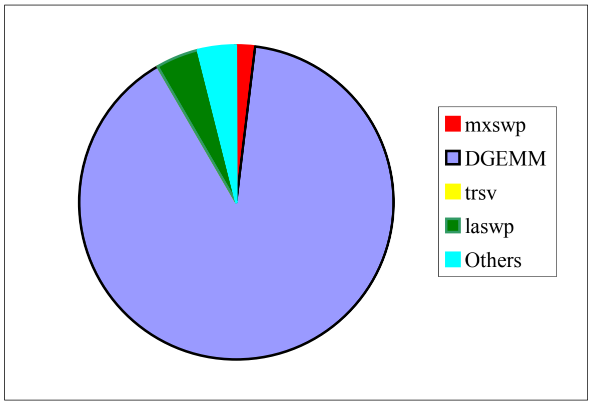
Related to the latency and bandwidth which are limited by interconnection network, the communication is hard to analyze. With the rapidly development of the network the communication is also not being a substantive performance bottleneck. Yet the computing performance is only related to the system frequency and problem size etc., apt to quantitate. Supported by the experience that for specific system the matrix size refers to the 80% memory capacity, the key parameters analysis is locked in the former two of main effect factors[9]: block size NB, processor array P × Q, matrix size N and communication method.

3. Observation and discussion

Through the analysis of HPL source code and calculation of above-mentioned algorithm complexity, it can be seen that, column expenses involves pivot search, though only (2×NB+4) communication each time, frequently as each column of matrix need to make at least $\log_2P$ time, and involves row permutation overhead $(N-(i+1)NB)\times NB/Q$ for U^i, which is time consuming for row permutation on column storage. The main overhead in row communication is to broadcast L, about $(N- i\times NB)\times NB/P$ each time. Based on such consideration, it is more close to rationalization to make P as power value of 2, P and Q approximate but P slightly smaller than Q in HPL test.

The determination of NB depends on experiential probing all the time, and there is no relatively qualitative method. Test experience[2] only provides: NB should be close to Cache line size as much as possible, generally between 32 and 256, which can display Cache performance in maximum and reduce Cache conflict, as well as utilize BLAS3 to carry on matrix operations as much as possible. For the data layout, the smaller NB is, the more balanced load is; For the computing, too small NB will restrict the computing performance, for it hardly has data reuse in senior level memory, and the quantity of news will increase too.

From the real test on a dual-CPU 1.6 GHz AMD 64-bit computer, see Fig. 2, clearly, matrix multiply-add operation DGEMM during update is key part of HPL process, i.e. the most time consuming part. So primary objective of NB determination is locked in obtaining peak computing performance on stand-alone computer.



Operation	Scale(%)
mxswp (search pivot & permute)	1.892345
DGEMM (matrix update)	89.63611
trsv (backward substiuttion)	0.063262
laswp (row exchange for U)	4.629095
Others	3.77919

Fig. 2. Time scale of main parts in HPL course

The algorithm and its operation course show that matrix multiply-add operation only relates to matrix dimension N and block size NB. The efficiency factor here is correspondingly defined as the ratio of the test time t_{DGEMM} and the mount of operations $2(N-(i+1)NB)^2 \times NB/(PQ)$, that is

$$\gamma = t_{DGEMM} / (2(N-(i+1)NB)^2 \times NB/(PQ))$$

Fig. 3. and Fig. 4. experiments based on Goto[10] library indicate that, the efficiency factor of matrix operation varies not more than several permillage with N gemination increase, but does the relevant change with NB to a great extent. With the increase of NB in a certain range, its value is obviously reduced. In addition, utilizing the turning flat curve point, the least scale of matrix displaying matrix operation efficiency can be determined roughly, further determine NB initial value.

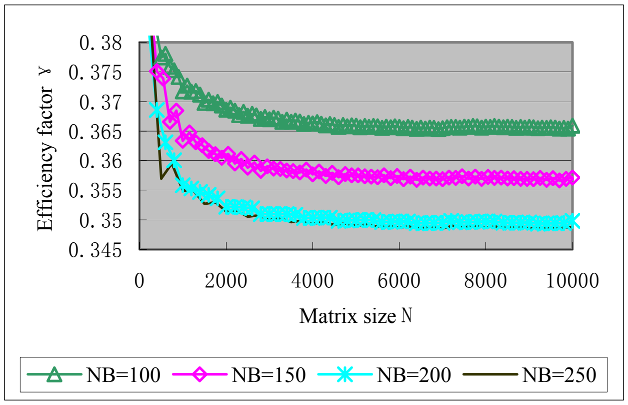


Fig. 3. γ under condition N=10000 and NB=100, 150, 200 and 250

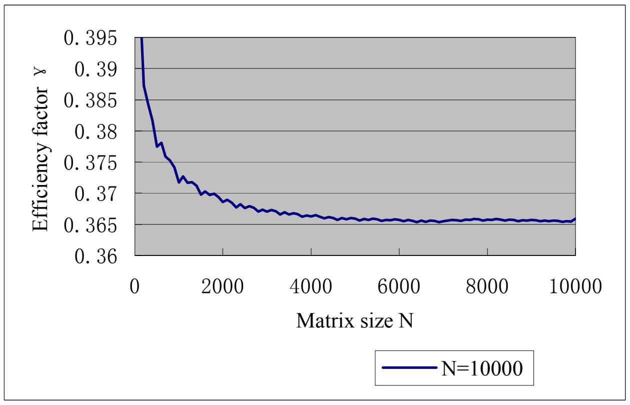
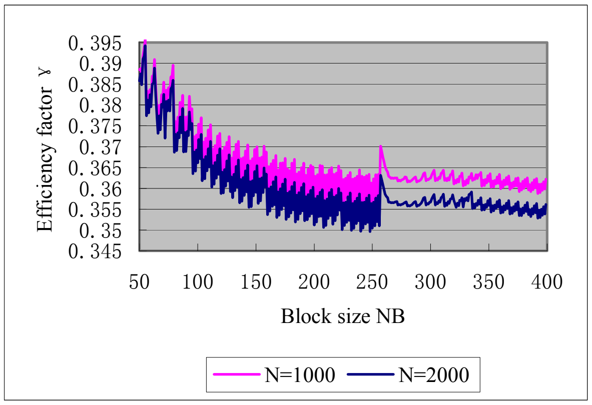


Fig. 4. γ under condition NB=100 and N ranging from 0 to 10000

According to the above-mentioned law, It makes possible for determine NB through multiply-add operation of proper scale matrix in quantity. In this way, so long as with small scale N, the change curve of γ can be got to the large range of NB with little time, after weighing other factors further, such as Cache etc., reasonable NB value could be quantitatively determined.

In order to verify this idea, the relevant experiments have been launched. Choosing representative high and low efficiency point separately from Fig. 5, a series of γ value got by small size matrix operation, the actual tests of real system to corresponding NB value are carried on. The test results in Table 1 reveal favorable verification to our inference. Just in the small-scale, the change of NB is not obvious in efficiency improvement of the system. As the scale of matrix increases, the advantage appears. Yet to HPL, to confirm NB finally should further take other systematic characteristics, such as Cache, etc. into account. Two scale γ curves in Fig. 5 is an effective response to the judgment that γ is nearly nothing to do with the size N.



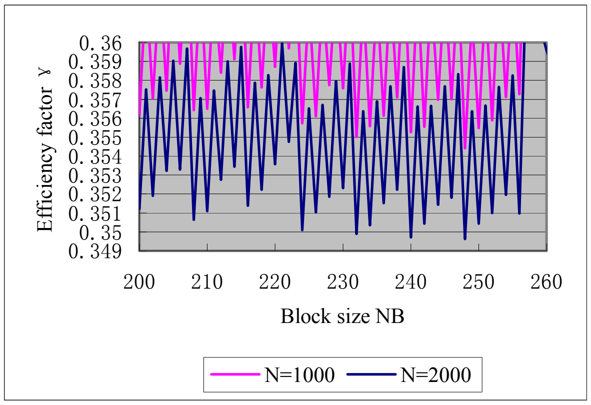


Fig. 5-1. Comparison of γ under condition N=1000 and 2000 **Fig. 5-2. Zoom in the high efficiency range of Fig. 5-1**

Table 1. Efficiency test results of representative NB on real system

N	NB	γ	Time(s)	Gflops	Efficiency (%)
1000	240	0.355267	0.30	2.221	69.4
	257	0.370144	0.32	2.112	66.0
2000	240	0.349712	2.16	2.467	77.1
	257	0.363151	2.25	2.373	74.2
10000	240	-	239.34	2.786	87.1
	257	-	247.77	2.691	84.1
14140	240	-	672.10	2.805	87.7
	257	-	696.76	2.705	84.5

4. Conclusion and further work

The experiments indicate, the efficiency of matrix operations is correlated with block size NB. Thus it is apt to determine suitable NB tentatively from the variation tendency of the operation efficiency curve of small-scale matrix. It provides a good basis for further confirming optimal NB according to Cache etc. to obtain peak performance finally, and greatly reducing the blindness of NB determination. This conclusion is equally suitable to parallel matrix operation of other courses as well.

Yet the above-mentioned test should be examined on a real large scale system and necessary to make further improvement on communication. Lots of experiments should be launched, such as overlapping computing and communication, hiding latency by turning synchronous communication into asynchronous one, especially weighing the memory size and the overhead of row permutation on column storage, further analyzing the feasibility that enlarge the matrix size auxiliary by hard disk prefetching or reduce matrix size with storing it both in row and column and so on, in order to improve the tested peak performance further. In addition, the compared study on thread and process will be done.

References

[1] Jack J. Dongarra and Piotr Luszczek and Antoine Petitet. The LINPACK Benchmark: Past, Present, and Future, *Concurrency and Computation: Practice and Experience*, 15, 2003.

[2] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary. HPL – A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, *http://www.netlib.org/benchmark/hpl/*.

[3] Hans W. Meuer, Erik Strohmaier, Jack J. Dongarra, and H.D. Simon. Top500 Supercomputer Sites, 17th edition, November 2 2001. (The report can be downloaded from http://www.netlib.org/benchmark/top500.html).

[4] BL Zhang, etc. *Theory and method of numeric parallel computing* (Beijing: National defense industry press, 1999,7).

[5] GL Chen. *Parallel computing: structure, algorithm, programming (modified version)* (Beijing: Advanced education press, 2003.8).

[6] ZZ Sun. *Numeric analysis (second edition)* (Nanjing: South-east university press, 2002.1).

[7] http://www.cs.utk.edu/~dongarra/WEB-PAGES/SPRI-NG-2000/lect08.pdf.

[8] R.P. Brent and P.E. Strazdins. Implementation of BLAS level 3 and LINPACK benchmark on the AP1000. *Fujitsu Scientific and Technical Journal*, 29(1): 61-70, 1993.

[9] H.-Y. Lin and P. Luszczek, "Tuning LINPACK N×N for PA-RISC Platforms", Presented at *High Performance Computing on Hewlett-Packard Systems Conference*, Bremen, Germany, October 7-9, 2001.

[10] http://www.cs.utexas.edu/users/kgoto/.