ヘテロなクラスタ環境における並列 LINPACK の最適化

笹生 健[†] 松岡 聡[†] 建部修見[††]

本研究では LINPACK Benchmark のひとつである HPL をヘテロなクラスタ環境向けに最適化して実装し，CPU が異なるノードの混在する CPU ヘテロなクラスタ上での評価を行なった．用いた最適化手法は，各ノードの性能差に応じて割り当てるデータサイズを変えることによってロードバランスを取るというものである．それにより，ピーク性能において理論性能の 57.1%，通常の HPL と比べ最大 1.49 倍の性能を達成した．

The Optimization of The LINPACK Benchmark for Heterogeneous Clusters

Takeru Sasou ,[†] Satoshi Matsuoka [†] and Osamu Tatebe[††]

In this study, we implemented the optimization of HPL, which is one of the LINPACK Benchmark, for a heterogeneous cluster system and evaluated on the CPU heterogeneous cluster. We used the technique of optimization that load sharing by changing data size correspoonding to a performance of each nodes. From the experimental results, we attains 57.1% efficiency to theoretical peak performance and 1.49 times at maximum as much as best performance of HPL.

1. はじめに

近年，コモディティハードウェアを使用する事によるコストパフォーマンスの良さなどの理由から超並列計算機に代わってパーソナルコンピュータやワークステーションをネットワークで繋いだクラスタ型並列計算機が注目されている．最近ではギガビット級のネットワークを利用できるようになったことから，世界の高性能コンピュータの上位 500 位をリストアップしている Top500[1] においても多くのクラスタ型並列計算機がランクインするようになり，現在も普及の一途を辿っている．

クラスタ型計算機はコモディティハードウェアによって構築されていることから，本質的に多様性をもっている．その多様性は CPU 性能やネットワークハードウェアの種類，メモリ容量の差異，場合によっては OS や CPU アーキテクチャまで及ぶことがあり，今後クラスタの普及に伴い，この様な不均質 (ヘテロ) なクラスタ型並列計算機の増加は必至である．

† 東京工業大学
Tokyo Institute of Technology
†† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology

性能ヘテロなクラスタ環境においてはノード間の性能差を考慮して負荷分散をすることが性能向上に必要であるが，既存の NAS Parallel Benchmark[2] や LINPACK Benchmark[3] の参考用実装などは各ノードの性能が均質 (ホモ) な環境で動くことを前提としており，各ノードに対して同量の負荷分散を行なっているため，ヘテロなクラスタ環境についてポータブルに性能評価をする事ができず，クラスタシステムの性能評価という要求に対し充分に応える事ができない．

そこで，本研究ではヘテロなクラスタシステムの性能評価をポータブルに行なえるベンチマークソフトウェアの構築を目指し，Tennessee 大学の J.Dongarra らによって開発された LINPACK Benchmark プログラムのひとつである HPL[4] を，ヘテロなクラスタ環境向けに最適化した．HPL は非対称密行列連立 1 次方程式を解く際の演算性能を評価するソフトウェアであり，実行時に様々なパラメータをシステムの構成に合わせて設定する事で，より性能を引き出した性能評価ができるという特徴がある．

本研究で採った最適化手法は，割り当てるデータサイズを各 PE の性能に応じて変えることでロードバランスをとるというもので，ヘテロ向けに最適化した HPL を CPU ヘテロなクラスタ上で実装し評価を行なった．

2. HPL

ここではHPLの概要と，ヘテロなクラスタ環境上でHPLを使用する上で起きる問題について述べる．

2.1 HPLの概要

HPLは，フリーのオープンソースとして提供されているLINPACK Benchmarkプログラムである。

LINPACK Benchmarkは非対称密行列連立1次方程式を解く際の演算性能を評価するもので，主に浮動少数点演算のためのベンチマークであり，規定では解法に関する制約は無いが，演算量は$\frac{2}{3}N^3 + O(N^2)$で評価する事が決められている．LINPACK Benchmarkは現在

- LINPACK Fortran $n = 100$ Benchmark
- LINPACK $n = 1000$ Benchmark (Toward Peak Performance)
- LINPACK's “Highly Parallel Computing” Benchmark

の3種類があり，LINPACK Fortran $n = 100$ Benchmarkは問題サイズを$n = 100$で固定，規定によりソースコードの改変も認められていないもので，LINPACK $n = 1000$ Benchmarkは問題サイズは$n = 1000$で固定だが，ユーザーがソースコードを改変する事が認められている．Highly Parallel ComputingはTop500レポートで使用されており，数値解にある一定以上の精度が要求されるが問題サイズや解法は規定されない．HPLもHighly Parallel Computingのひとつである．

HPLは問題サイズの他，ブロックサイズや通信トポロジー，パネルLU分解のアルゴリズム等の様々なパラメータを計算機の特性に合わせて設定する事で，より高い性能が発揮できるようになっている．

HPLでは，まず係数行列を複数の正方形のブロックパネルに分割して2次元プロセス格子上にブロックサイクリックに分割する．例えば2×3のプロセス格子にデータ分割すると図1の様になる．

連立1次方程式を解く際に使用するアルゴリズムは外積形式ブロック*LU*分解アルゴリズムであり，主にPanel Factorization，Panel Broadcast，Update，Backward Substitutionというフェイズからなっている．それぞれにおいてはパネル列LU分解，分解済パネルの送信，未分解小行列の更新計算，後退代入演算による$Ux = y$の求解を行なう．

HPLではPanel Factorizationにおいてrow partial pivotingのためにパネル列内のプロセス間でallgather通信を行なう．そのため縦方向の同期通信が頻繁に起こる．

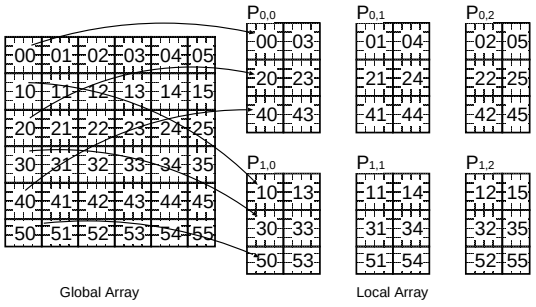

図1 ブロックサイクリックなデータ分割

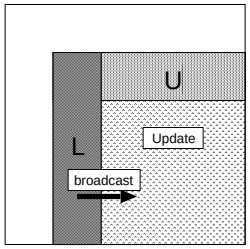

図2 ブロックLU分解

2.2 ヘテロなクラスタ上でのHPLの問題点

HPLではブロックサイクリックにデータ分割されるため，各PE(プロセッサエレメント)にはほぼ同量のデータが割り当てられ，性能ホモなクラスタにおいてはロードバランスがとれる．

しかし，各ノードの演算性能に差異があるようなヘテロな構成のクラスタ上でHPLを実行した場合，全てのPEにかかる負荷が同量なので演算性能の高いPEが性能の低いPEの処理が終わるのを待つ時間が生じてしまい，全体の性能が落ちてしまう．実際，800MHz×3 + 400MHz×1というCPUヘテロな構成のクラスタ上でHPLを実行した場合，図3のように性能の低いPEに足を引っ張られて充分な性能が出ない事がわかる．

3. ヘテロなクラスタ向けHPL

3.1 ロードバランス

ヘテロなクラスタ環境においては，CPU性能，メモリ容量，ネットワーク性能などハードウェア構成が異なるノードが混在する．そのため，ヘテロなクラスタ上で並列計算を行なう場合，各ノードの性能に応じた負荷分散やスケジューリングが性能向上のために必須となる．例えば，ノード間でCPU性能に差がある場合，CPU性能が高い方が単位時間内での演算量は多いので，性能が高い方により多く負荷をかけた方がクラスタ全体の性能が上がる．メモリ容量に差がある場合は，同じサイズでデータ分散するよりもメモリ利用効率が高くなるよう

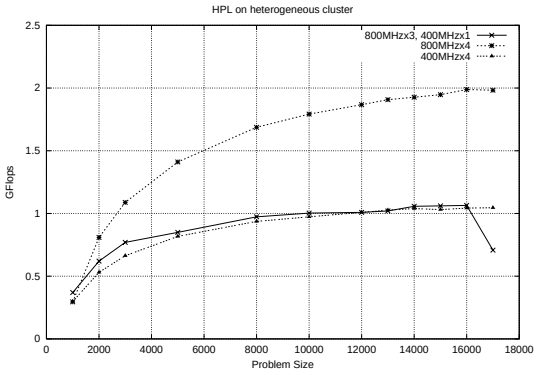

図 3 CPU ヘテロなクラスタ上での HPL の性能

P_0	P_1	P_3
P_0	P_2	P_4

図 4 ヘテロなプロセス格子割り当て

にサイズを変えてデータ分散した方がより大規模なサイズのデータを処理できる．

上で述べた様な特性のために，2 章で示したように HPL はヘテロなクラスタ環境上では性能が低下してしまうので，ヘテロなクラスタの構成を考慮してロードバランスを取れるようにする必要がある．

そこで，本研究では HPL をヘテロなクラスタ向けにするため，各 PE の性能に応じて割り当てるデータサイズを変えることでロードバランスをとるようにした．具体的には，一つの PE がプロセス格子上の複数のブロックを受け持つようにする．

例えば，P_0 が他の PE の 2 倍の演算性能をもっている場合，図 4の様に割り当てると全ての PE の処理時間が同じになり，演算性能の高い PE が性能の低い PE の処理を待つ事が無くなる．

また，P_0 が他の PE の 2 倍メモリ領域を使用できる場合も，図 4の様に割り当てることでメモリをより有効に利用できる．

3.2 実装の方法

最適化手法を実現する方法は主に以下の 2 つである．

- 1PE 上でプロセスを複数起動する
- 1 プロセスで複数のパネルを処理できるようにする

前者はプログラムに変更を加えなくてもいいので容易に実現できるが，1PE 上で複数のプロセスが動くため，単純に通信回数が増大し，また必ずスリープ状態のプロセスが存在するためプロセス同士の同期を取るのが困難である．HPL では縦方向の同期通信が頻繁に起こるので，うまく同期が取れないと通信のオーバーヘッドが増大してしまう．

後者はアルゴリズムなどを変更しなければならないが，うまくロードバランスをとれば各プロセスが同期的に動く事が可能である．

なお，4 章でこの 2 種類の方法の性能比較を行なう．

1 プロセスで複数のパネルを処理できるようにした HPL(以下ヘテロ版 HPL) では LU 分解の計算手順の変更はしない．ただし，各プロセスが複数のブロックパネルを処理するようになったので，各パネルの属する列を調べて各フェイズにおいて処理するべきものを全部まとめて処理するようにする．また，横方向で隣接するプロセスが 2 つ以上であったり，パネルのデータ通信相手が自プロセス内の異なるパネルになる場合があるので通信に関わる部分に変更が必要である．

また，通信相手の特定や，通信相手の保持するパネルの情報を取得するためにプロセス格子への各プロセスの割り当てのテーブルを保持するようにした．

次に各計算フェイズにおける変更を述べる．

3.2.1 パネル LU 分解

パネル LU 分解では，列の部分ピボッティングのためにプロセス列中のプロセス同士で binary-exchenge 形式の allgather 通信を行う．この時，処理するべきいくつかのパネルのうちから 1 列だけ選んで交換すればよいので，次のような手順で処理を行うようにした．

(1) プロセス中の処理するべき全てのパネルの最大要素を調べ，最大要素を含む行をもつパネルを選択する．
(2) プロセス間で binary-exchenge 通信を行い，軸として選択された行配列と対角要素を含む行配列を取得する．
(3) 選択軸とローカルな行配列のスワップを行ない，選択軸のバッファへのコピーを行う．

なお，現在は部分軸選択つき LU 分解，再帰的 LU 分解ともに right-looking 形式のアルゴリズムのみ実装している．

3.2.2 Panel Broadcast

HPL では，各プロセスが一つのパネルを一つのプロセスに送信すればよかったが，ヘテロ版 HPL では通信相手が 2 つ以上になることもあり，その場合図 5(1) の様に通信ステップが増えるために他のプロセスが Update 処理に入るまでに待ち時間が生じてしまう．

また，同じパネル列内に同プロセスが保持するパネルがある場合も図 5(2) の様に他のプロセスとの足並みが乱れる事となる．

現在の実装では，保持するパネルを一つずつ Broadcast し，同じパネル列に属する同プロセス内のパネル同

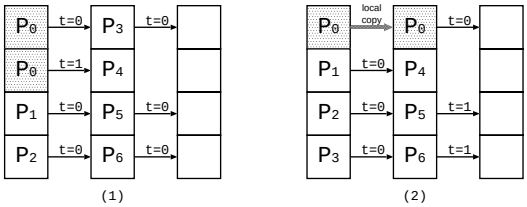

図 5 ヘテロな Broadcast

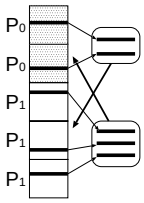

図 6 Update フェイズでの行交換通信

士ではローカルにコピーするようになっている．

3.2.3 Update

Update では，まず最初に縦方向に U の Broadcast と行交換のための allgather 通信を行い，本来 U とスワップするべき行配列と U パネルを送受信しあう．ヘテロ版 HPL ではこれを次のような手順で行う．

(1) 同じパネル列に属するパネルがそれぞれ持っている U とスワップするべき行配列を一つの作業領域に集める．

(2) 作業領域に集めたデータを用いて allgather 通信を行う．U パネルを受信したときは，全てのパネルで行交換を行う．

(3) allgather 通信が終わったら，U パネルを全てのパネル構造体にコピーする．

また，プロセスが複数のプロセス内に属している場合，あるパネル列の allgather 通信を終わるまで異なるパネル列の allgather 通信が行えないが，この通信が最初に 1 回だけ行われた後は各プロセスは独立して処理ができる．そこで，保持するパネルの allgather 通信を全て最初に済ませてしまい，その後で消去計算を行う．

4. 性能評価

ここでは CPU がヘテロなクラスタ環境において HPL を 1PE1 プロセスずつで実行した場合， HPL を 1PE 複数プロセスで実行した場合，ヘテロ版 HPL を実行した場合の 3 通りで性能比較を行い，結果についての考察を述べる．

評価実験には松岡研究室の Prospero クラスタを使用した．表 1に Prospero の個々のノードの仕様を示す．

Prospero をヘテロな環境とするために， CPU を

CPU	Intel PentiumIII 800MHz×2
Motherboard	MSI MS-6120N
Memory	640MB
HDD	28.8GB
Network Card 1	DEC 21140AF(Planex)
Network Card 2	ADMtek Centaur-P(AN983)
Network Card 3	ADMtek Centaur-P(AN983)
Operating System	Redhat Linux release6.2(Zoot)
	Kernel 2.2.16

表 1 Prospero クラスタの仕様

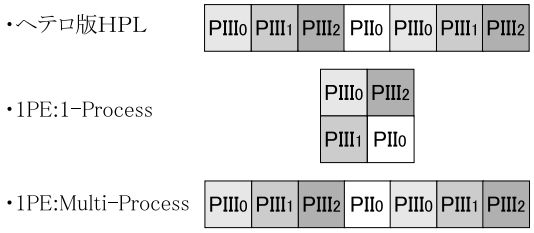

図 7 $PIII_{0,1,2}$=800MHz, PII_0=400MHz

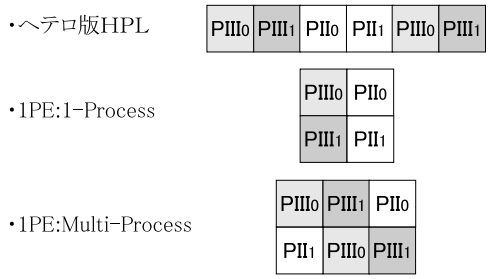

図 8 $PIII_{0,1}$=800MHz, $PII_{0,1}$=400MHz

PentiumII 400MHz に交換したノードを混ぜて， CPU がヘテロな環境を構築した．

現在の実装では Update における binary-exchange 通信の部分に不具合があり， 1×Q という形状のプロセス格子でしか実行できないため，ノード数を増やした際の並列化効率が良くないので， 4 ノード使用の小規模な構成で実験を行なう事にした．

評価は 800MHz×3 + 400MHz×1， 800MHz×2 + 400MHz×2 の 2 つの場合で行ない，問題サイズは N=1000 〜 17000 の間で計測した．

各 PE のプロセス格子への割り当てを図 7， 8に示す．

HPL をコンパイルする際のライブラリや設定は次のようになっている．

- MPI : mpich-1.2.0[5)]
- BLAS : ATLAS 3.2.0[6)]
- コンパイルオプション

```
-O6 -DAdd_ -DF77_INTEGER=int
-DStringSunStyle -DHPL_CALL_CBLAS
```

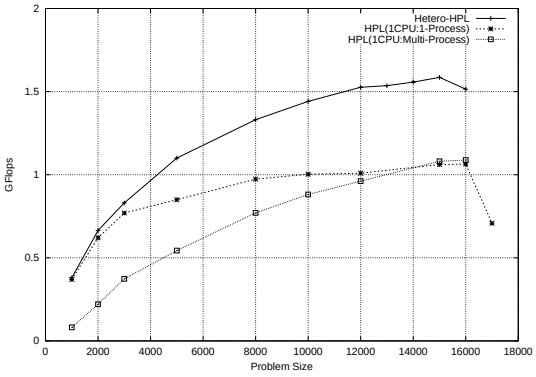

図 9 HPL とヘテロ版 HPL の比較 (800MHz×3, 400MHz×1)

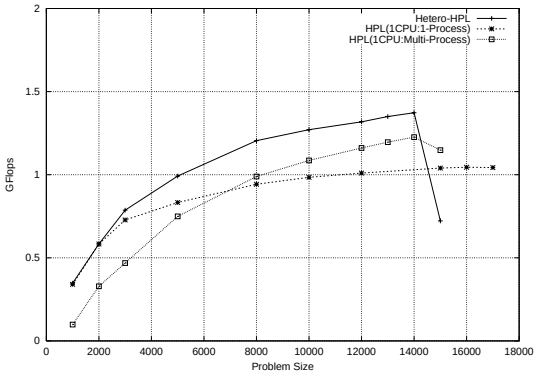

図 10 HPL とヘテロ版 HPL の比較 (800MHz×2, 400MHz×2)

N	ヘテロ版 HPL	HPL 1PE:1-Process	HPL 1PE:Multi-Process
1,000	0.3798	0.3698	0.08186
2,000	0.6642	0.6202	0.2204
3,000	0.8300	0.7696	0.3738
5,000	1.100	0.8495	0.5438
8,000	1.331	0.9734	0.7706
10,000	1.441	1.003	0.8808
12,000	1.526	1.009	0.9612
13,000	1.536	1.020	1.000
14,000	1.557	1.057	1.050
15,000	1.586	1.061	1.081
16,000	1.515	1.064	1.088
17,000	未測定	0.7079	未測定

表 2 HPL とヘテロ版 HPL の比較 (800MHz×3, 400MHz×1)

N	ヘテロ版 HPL	HPL 1PE:1-Process	HPL 1PE:Multi-Process
1,000	0.3474	0.3407	0.09841
2,000	0.5836	0.5833	0.3288
3,000	0.7868	0.7277	0.4684
5,000	0.9920	0.8330	0.7494
8,000	1.205	0.9427	0.9891
10,000	1.271	0.9844	1.086
12,000	1.318	1.010	1.161
13,000	1.350	1.020	1.196
14,000	1.373	1.031	1.226
15,000	0.7219	1.040	1.148
16,000	未測定	1.045	未測定
17,000	未測定	1.043	未測定

表 3 HPL とヘテロ版 HPL の比較 (800MHz×2, 400MHz×2)

4.1 ロードバランスの最適化

各 PE の CPU の違いによる演算性能の差を埋めてそれぞれの処理の足並みをそろえるためには，実行時間の逆比となるサイズでブロックを割り当てれば良い．しかし，各計算フェイズにおけるオーダーは異なるため，どれか一つについて最適化することになる．

次元数 N の行列を $P \times Q$ のプロセス格子上に分割した時の各計算フェイズのオーダーは以下のようになる．

- Panel Factorization

$$\frac{3}{2P} \cdot N^2 + O(N)$$

- Update

$$\frac{2N^3}{3PQ} + \frac{P+Q}{PQ} \cdot O(N^2) + O(N)$$

- Backword substitution

$$\frac{1}{PQ} \cdot O(N^2)$$

これに見れるように，処理の大部分は Update に費される．そのため，割り当てるデータサイズの比を Update フェイズの処理にかかる時間の逆比とすることでロードバランスを取ることにした．

適当な問題サイズで 1CPU での実行時間を測定したところ，400MHz と 800MHz それぞれの Update 処理時間の逆比はおよそ 1 : 1.988 となったので，データサイズの比は 1 : 2 とした．

4.2 評価結果

N=1000～17000でそれぞれ計測した結果を図9,10及び表2,3に示す．表2,3において未測定となっている箇所は，実行中にメモリのスワップが起きてしまうので最後まで測定していないものである．

ヘテロ版 HPL は 400MHz×1 においてピーク性能 1.586GFlops であり，理論性能 2.8GFlops の 56.6%，HPL のピーク性能 1.064GFlops の 1.49 倍を達成している．400MHz×2 においてもピーク性能 1.373GFlops であり，理論性能 2.4GFlops の 57.1%，HPL のピーク性能 1.045GFlops の 1.31 倍を達成している．

これはヘテロ向けの最適化が有効である事を示している．

また，性能ホモな状態では Prospero 上での HPL の対理論性能対の割合はおよそ 61.2% である事から考えると，ヘテロ版 HPL は性能効率が若干低下しているが，これは通信トポロジーが (P,Q)=1 × 4 のプロセス格子と同じであることによる性能効率の低下である．実際，(P,Q)=1 × 4 では，性能ホモな Prospero 上での HPL の対理論性能比は 58% 程度である．

一方，1CPU 上で複数プロセスを起動した場合は，400MHz×1 においてピーク性能 1.088GFlops，400MHz×2 においては 1.226GFlops というようにヘテロ版 HPL と比べてだいぶ性能が劣る．これは，同じ PE 上で複数のプロセスを起動している場合，あるプロセスが動いている間は同 PE 上の他プロセスはスリープ状態となるため，プロセス間で通信を行なおうとしてもどちらかのプロセスがスリープしているという状態が生じて，通信部分がオーバーヘッドとなったからであると考えられる．

なお，ヘテロ版 HPL は HPL よりも問題サイズが小さいところでピーク性能に達しているが，これは CPU400MHz の PE のメモリ利用効率が悪いためである．

5. おわりに

本稿では分散メモリ型並列計算機用のベンチマークソフトウェアである HPL をヘテロなクラスタ環境向けに最適化し，実装してこれを評価した．性能評価では，CPU がヘテロな環境において HPL，HPL をナイーブな最適化の実装をしたものとの比較を行なった．

ヘテロ向けの最適化により，CPU が PentiumIII 800MHz のノードと PentiumII 400MHz ノードが混在するヘテロなクラスタ環境においてヘテロ版 HPL は HPL と比べ最大 1.49 倍の性能を達成し，これによりヘテロなクラスタ向けの最適化手法が有効であることがわかった．

今後の課題としては，次のようなものが挙げられる．

- プロセス格子への割り当て方による性能差の検証
- メモリサイズがヘテロな場合での評価
- ネットワークがヘテロな場合での評価
- 割り当てるデータサイズなどパラメータの自動チューニング

プロセス格子への割り当て方を変えると，通信トポロジーの違いから最適な割り当て方があると考えられる．

また，演算速度に応じて割り当てるデータサイズを変えるという手法では，演算性能の低い PE のメモリ利用効率が悪くなるため，クラスタの構成次第では逆に性能が遅くなることも考えられ，これについても検証が必要である．

データサイズを決定するための要因は CPU 性能やメモリサイズ，ネットワーク性能など，実行時に静的に与えられるものであり，一度クラスタ構成に関する情報を取得しておくことで最適なデータサイズの自動セッティング機能は実現でき，今後実装する予定である．

参 考 文 献

1) TOP500 Supercomputer Sites:TOP500 Supercomputer Sites, http://www.top500.org/
2) The NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB/
3) http://www.netlib.org/benchmark/top500/lists/linpack.html
4) A.Petitet, R.C.Whaley, J.Dongarra, A.Cleary: HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, http://netlib.org/benchmark/hpl/index.html
5) http://www-unix.mcs.anl.gov/mpi/mpich/
6) Automatically Tuned Linear Algebra Software (ATLAS), http://www.netlib.org/atlas/
7) 小国力 編著, 村田健郎, 三好俊郎, ドンガラ.J.J, 長谷川秀彦 著: 行列計算ソフトウェア WS，スーパーコン，並列計算機.
8) S. Toledo: Locality of Reference in LU Decomposition with partial pivoting, SIAM Journal on Matrix. Anal. Appl., Vol. 18, No. 4, 1997.
9) J. Choi, J. J. Dongarra, and D. W. Walker: Parallel Matrix Transpose Algorithms on Distributed Memory Concurrent Computers, Parallel Computing, Vol. 21, pages 1387-1405, 1995.
10) J. Dongarra, R. van de Geijn and D. Walker: Scalability Issues in the Design of a Library for Dense Linear Algebra, Journal of Parallel and Distributed Computing, Vol. 22, No. 3, pp. 523-537, 1994.
11) F. Gustavson: Recursion Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms, IBM Journal of Research and Development, Vol. 41, No. 6, pp. 737-755, 1997.