ハイパフォーマンス
コンピューティング 57−10
（1995. 8. 24）

分散メモリ型並列計算機による LU 分解

建部 修見*
tatebe@is.s.u-tokyo.ac.jp
東京大学大学院理学系研究科情報科学専攻
〒113 東京都文京区本郷 7-3-1

概要

分散メモリ型並列計算機で効率的に実行できる LU 分解のアルゴリズムの研究を行ない，その計算モデルの構築を行なった．効率的な実装としてはソフトウェアパイプライニング，非同期的消去があげられ，これらを行なうことで通信遅延，他プロセッサのピボット選択の時間の隠蔽を行なうことが可能となる．またこの隠蔽を行なえるための条件も示した．さらに富士通の並列計算機 AP+ を用いてこのモデルの評価を行ない，ほぼそのモデルの仮定が正しいことが確かめられた．また多段同時消去を行なった時の隠蔽しやすさからサイクリック - サイクリック分割が望ましいことが分かった．

LU Decomposition on Distributed Memory Machines

Osamu Tatebe
Department of Information Science, Faculty of Science, the University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, JAPAN

Abstract

I studied an efficient algorithm of the LU decomposition on distributed memory machines and constructed the computational model. Efficient techniques are software pipelining and asynchronous elimination, which result in hiding of communication latency and time of pivoting on other processors. I also showed the condition of hiding. Moreover, this computational model is evaluated by Fujitsu multicomputer AP+, and ascertained to be almost correct. On the other hand, cyclic-cyclic distribution is desirable since simultaneously multiple elimination does not make worse condition of hiding.

*日本学術振興会特別研究員 (JSPS Research Fellow)

1 はじめに

連立一次方程式の代表的な直接解法である LU 分解は非常に重要なアプリケーションであり，さまざまな分野で用いられている．本研究ではこの LU 分解の分散メモリ型並列計算機への効率的実装を行なうためのアルゴリズム，またそのアルゴリズムについてさまざまなデータ分散における計算モデルの構築を行なう．さらにこのモデルの評価を実機で行ない，ほぼ正しいことを示す．

2 LU 分解の計算モデル

2.1 アルゴリズム

$N \times N$ の連立一次方程式を部分ピボット選択付き LU 分解 (図 1) を使って解くことを考える．LU 分解は密行列の解法として非常に良く使われている解法で，行列 A を下三角行列 L と上三角行列 U の積に分解し，それら三角行列を解くことにより解を得る方法である．この解法は計算量が多いが ($O(N^3)$)，ほぼスケーラブルな台数効果が期待されるアルゴリズムである．

```
for (i = 0; i < N; i++) {
    piv_i = piv[i];      // pivot selection
    inv_p = A[piv_i][i] = 1 / A[piv_i][i];
    for (j = i + 1; j < N; j++) {
        c = A[piv[j]][i] *= inv_p;
        for (k = i + 1; k < N; k++)
            A[piv[j]][k] -= c * A[piv_i][k];
    }
}
```

図 1. LU 分解のプログラム．このプログラムではピボットがあらかじめ分かっているが，実際は i 列の最大値をピボットとする．

係数行列は 2 次元ブロックサイクリック分割[1]で分散させ，プロセッサ数を $P_{xy} = P_x \times P_y$ とする．図 1 を分散メモリ型並列計算機に実装するとピボット行，列の参照で通信が発生してしまう．この通信遅延を隠蔽するためにまず最外ループのソフトウェアパイプライニングを行なう．

図 2 では，先行的に次のピボットを決め，送信することにより通信と計算のオーバーラップを行ない，通信遅延を隠蔽している．また，プロセッサは列サイクリックにも分散され，部分ピボット選択に必要なプロセッサは部分的にしかないことを利用し，図 3 の様にピボット選択で暗黙の同期をとらず，非同期的な[2]実行を行なわせることができる．

[1] !HPF$ DISTRIBUTE (CYCLIC(NX), CYCLIC(NY))

```
pivot selection;
broad pivot;
for (i = 0; i < N; i++) {
    receive pivot;
    inv_p = A[piv_i][i] = 1 / A[piv_i][i];
    // delete only pivot column
    for (j = i + 1; j < N; j++) {
        c = A[piv[j]][i] *= inv_p;
        k = i + 1;
        if (k < N)
            A[piv[j]][k] -= c * A[piv_i][k];
    }
    pivot selection;
    update pivot column;
    broad pivot;
    // delete the rest columns
    for (j = i + 2; j < N; j++) {
        c = A[piv[j]][i];
        for (k = i + 2; k < N; k++)
            A[piv[j]][k] -= c * A[piv_i][k];
    }
}
```

図 2. ソフトウェアパイプライニングを行なった LU 分解のプログラム．

2.2 行サイクリック分割

まず簡単のために行サイクリック分割を考える．行サイクリック分割ではピボット選択は全てのプロセッサで行なう必要があり，図 2 のアルゴリズムで考えれば良い．この時，全てのプロセッサが (緩やかな) 同期をとりながら各段の消去を行なっていると考えられる．

いま $D_{p,i}$ をプロセッサ p が i 段の消去を終える時刻とし，P_i を i 段のピボット選択を終える時刻とする．この時，r を受信のメッセージハンドリングオーバーヘッド，p_i を i 段のピボット選択にかかる時間 (送信のオーバーヘッド含む)，t を通信遅延，$d_{p,i}$ をピボット列以外の列の消去にかかる時間とすると，P_i, $D_{p,i}$ は，

$$
\begin{aligned}
P_0 &= p_0 \\
D_{p,-1} &= P_0 \\
P_i &= \max_p D_{p,i-2} + r + p_i \\
D_{p,i} &= P_{i+1} + \max\{d_{p,i},\ t\}
\end{aligned}
$$

[2] 非同期的と書くと誤解を生んでしまうかも知れないが，プロセッサ全体で同期を行なわないという意味で使っている．

```
pivot selection;
broad pivot;
for (i = 0; i < N; i++) {
   receive pivot;
   inv_p = A[piv_i][i] = 1 / A[piv_i][i];
   if (we have the next pivot row) {
      // delete only pivot column
      for (j = i + 1; j < N; j++) {
         c = A[piv[j]][i] *= inv_p;
         k = i + 1;
         if (k < N)
            A[piv[j]][k] -= c * A[piv_i][k];
      }
      pivot selection;
      update pivot column;
      broad pivot;
      // delete the rest of columns
      for (j = i + 2; j < N; j++) {
         c = A[piv[j]][i];
         for (k = i + 2; k < N; k++)
            A[piv[j]][k] -= c * A[piv_i][k];
      }
   }
   else {
      for (j = i + 1; j < N; j++) {
         c = A[piv[j]][i] *= inv_p;
         for (k = i + 1; k < N; k++)
            A[piv[j]][k] -= c * A[piv_i][k];
      }
   }
}
```

図 3. 非同期の LU 分解のプログラム.

となる. この場合は,

$$d_{p,i} > t \tag{1}$$

が満たされれば, 通信遅延を隠蔽できることになる. また各段の消去の終了する時刻は $\max_p D_{p,i}$ である.

行サイクリック分割の場合は全てのプロセッサが各段でピボット選択をしなければならないが, その分ピボット選択を早く終えることができ, p_i は

$$p_i = a_p\lceil (N-i)/P_y \rceil + b_p \log P_y \tag{2}$$

となり, ほぼピボット選択の時間は逐次の場合の $1/P_y$ となる.

通信遅延が隠蔽できる場合, LU 分解の実行時間はそれぞれのプロセッサの計算量 (flop) で決まる. 部分ピボット選択を行なっているため, その選択のされ方によって計算量は異なるが, ピボットがもっとも均等に選ばれる最善ケースでは, もっとも計算量の多いプロセッサの計算量は,

$$\begin{aligned}\max_p \sum_{i=0}^{N-1} d_{p,i} &= 2\left(\sum_{i=0}^{N-1}(N-i)\lceil\frac{N-i}{P_y}\rceil - \frac{N^2}{P_y}\right)\\ &= 2\left(\sum_{i=1}^{N} i\lceil\frac{i}{P_y}\rceil - \frac{N^2}{P_y}\right)\\ &= \frac{2N^3}{3P_y} + \frac{N^2}{2} - \frac{3N^2}{2P_y} - \frac{NP_y}{6} + \frac{N}{2}\end{aligned}$$

となり, ピボット選択が偏って行なわれる最悪ケースでは

$$\begin{aligned}\max_p \sum_{i=0}^{N-1} d_{p,i} &= 2\left(\frac{N}{P_y}\sum_{i=N/P_y+1}^{N} i + \sum_{i=1}^{N/P_y} i^2 - \frac{N^2}{P_y}\right)\\ &= \frac{N^3}{P_y} - \frac{N^3}{3P_y^3} - \frac{N^2}{P_y} + \frac{N}{3P_y}\end{aligned}$$

となる.

2.3 列サイクリック分割

次に列サイクリック分割を考える. この場合, 図 3 のアルゴリズムで考えることになる. この時, 前節と同様に P_i, $D_{p,i}$ を定めると次のようになる.

$$\begin{aligned}P_0 &= p_0\\ D_{p,-1} &= 0\\ P_i &= \begin{cases}\max\{D_{\pi(i),i-2},\ P_{i-1}\} + p_i, & \text{if } \pi(i-1)=\pi(i)\\ \max\{D_{\pi(i),i-2},\ P_{i-1}+t\} + r + p_i, & \text{otherwise}\end{cases}\\ D_{p,i} &= \begin{cases}P_{i+1} + \max\{d_{p,i},\ t\}, & \text{if } p=\pi(i+1)\\ \max\{D_{p,i-1},\ P_i\} + d_{p,i}, & \text{if } p=\pi(i)\\ \max\{D_{p,i-1},\ P_i+t\} + r + d_{p,i}, & \text{otherwise}\end{cases}\end{aligned}$$

ただし $\pi(i)$ は i 段のピボット列を担当するプロセッサを表し,

$$\pi(i) = \mathrm{mod}(i,\ P_x)$$

である. i 段目のピボット選択が終了するのは, ピボット選択を行なうプロセッサが $i-2$ 段目の消去を終え, $i-1$ 段目のピボット選択が終了し, そのピボットを受信し, 次のピボット選択を行なうまでの時刻であり, プロセッサ p が i 段目の消去を開始するのは, 自プロセッサが $i-1$ 段目の消去を終了し, i 段目のピボット選択が終了した時刻である.

したがってこのモデルでは $D_{p,i}$ の max 計算において, $D_{p,i-1}$ が選択されれば, つまり各 i 段で次の条件

$$\min_p D_{p,i-1} > P_i + t \tag{3}$$

が満たされていれば, 通信遅延は完全に隠蔽されかつピボット選択を待つことなく完全に非同期に実行されることになる. いま簡単のためにロードバランスは均等 $(d_{0,i} = d_{1,i} = \cdots = d_{P_x-1,i})$ とすると, $D_{p,i}$ の計算式より一番最後に先行的にピボッティングをするプロセッサが一番早く消去を終了することになるため, $\min_p D_{p,i-1} = D_{0,i-1}$ となる. 従って, $D_{0,P_x-2} > P_{P_x-1} + t$ が満たされれば, つまりそのプロセッサがピボットを先行消去する時に, ピボットの受信を待たなければ, 通信遅延は完全に隠蔽されることになる. P_x が少ない場合は $d_{p,i} = d_{p,i+1} = \cdots = d_{p,i+P_x-1}$, $p_i = p_{i+1} = \cdots = p_{i+P_x-1}$ と考えて良く, この場合, 通信遅延が隠蔽され, 他のプロセッサのピボット選択を待たない条件は $d_{p,i} > p_i + t$ となる. しかしながらプロセッサの性能をあげるためには, ロード, ストアの回数を減らすため多段同時消去をどんどん行なう必要があり, m 段同時消去を行なう場合, ピボット選択の時間は m^2 倍に増えてしまうため, この条件を満たすのは厳しくなると考えられる.

列サイクリック分割の場合は, ピボット選択をするプロセッサは各段一つなので,

$$p_i = a_p(N - i) \tag{4}$$

であるが, 他のプロセッサで行なわれているピボット選択の時間は隠蔽できるため, 実質上自分の担当のピボット選択の時間しか実行時間には出てこない.

通信遅延, 他プロセッサのピボット選択が隠蔽される場合は, 分解の時間はそれぞれのプロセッサの計算量によって決まる. 列サイクリックの場合はピボット列がそれぞれのプロセッサから均等に減っていくため, 計算量最大のプロセッサの計算量は行サイクリック分割の最善ケースと等しくなり,

$$\max_p \sum_{i=0}^{N-1} d_{p,i} = \frac{2N^3}{3P_x} + \frac{N^2}{2} - \frac{3N^2}{2P_x} - \frac{NP_x}{6} + \frac{N}{2}$$

となる. ここで N と P の関係に現実的な条件 $\frac{N}{P_x} = N_p = \text{const.}$ をつける. この時,

$$\max_p \sum_{i=0}^{N-1} d_{p,i} = \left(\frac{2N_p}{3} + \frac{1}{2} - \frac{1}{6N_p}\right)N^2 + \left(\frac{1}{2} - \frac{3N_p}{2}\right)N$$

となり, 最大次 N^2 の係数は漸近的に $\frac{2N_p}{3} + \frac{1}{2}$ に一致する.

2.4 サイクリック - サイクリック分割

サイクリック - サイクリック分割は行サイクリック分割と, 列サイクリック分割を合わせたようなものとして考えることができ, この分割を扱うためにプロセッサ一列をプロセッサグループとして, そのプロセッサグループが列サイクリック分割されていると考える. この時 P_i, $D_{p,i}$ は次のように表せる.

$$\begin{aligned}
P_0 &= p_0 \\
D_{p,-1} &= 0 \\
P_i &= \max\{D_{\pi(i),i-2},\ P_{i-1} + t\} + r + p_i \\
D_{p,i} &= \begin{cases} P_{i+1} + \max\{d_{p,i},\ t\}, \\ \qquad\qquad \text{if } p = \pi(i+1) \\ \max\{D_{p,i-1},\ P_i + t\} + r + d_{p,i}, \\ \qquad\qquad \text{otherwise} \end{cases}
\end{aligned}$$

ただしここでプロセッサ p はプロセッサグループ番号である. 列サイクリック分割との違いは, ピボット列だけではなくピボット行の参照も通信が必要になるということである. したがってピボットを担当しているプロセッサグループ内でもピボット行の通信が必要になり, ピボットの後には必ず通信が必要になる.

この場合, 通信遅延が隠蔽される条件, 他のプロセッサのピボット選択の時間が隠蔽される条件は列サイクリックの場合と同様である. しかしながら列サイクリックと違い列方向にもサイクリックに分散させているため, 多段同時消去を行ないこれらの遅延が隠蔽される条件を満たさなくなっても, 列方向のプロセッサを多くすることで条件を満たすようにできる.

また通信遅延が完全に行なわれ, ピボット待ちがない場合, 分解が終了する時間はそれぞれのプロセッサの計算量によって決まるが, サイクリック - サイクリック分割の場合は, 行サイクリック分割の時と同様にピボット選択の仕方によってその計算量は変わる. 以下では簡単のために $P_x = P_y = \sqrt{P_{xy}}$ の場合を考える. 最善ケースでは計算量の最大値は,

$$\begin{aligned}
\max_p \sum_{i=0}^{N-1} d_{p,i} &= 2\left(\sqrt{P_{xy}} \sum_{i=1}^{N/\sqrt{P_{xy}}} i^2 - \frac{N^2}{P_{xy}}\right) \\
&= \frac{2N^3}{3P_{xy}} + \frac{N^2}{\sqrt{P_{xy}}} - \frac{2N^2}{P_{xy}} + \frac{N}{3}
\end{aligned}$$

となり, 最悪ケースでは

$$\max_p \sum_{i=0}^{N-1} d_{p,i} = 2\left(N \sum_{i=2}^{N/\sqrt{P_{xy}}} i + \sum_{i=1}^{N/\sqrt{P_{xy}}} i - \frac{N^2}{P_{xy}}\right)$$

$$= \frac{N^3}{P_{xy}} + \frac{N^2}{\sqrt{P_{xy}}} - \frac{N^2}{P_{xy}} - 2N + \frac{N}{\sqrt{P_{xy}}}$$

となる. 最善ケースでは列サイクリックと比べ N^3 の係数は同じで, N^2 の係数は必ず

$$\frac{1}{\sqrt{P_{xy}}} - \frac{2}{P_{xy}} \leq \frac{1}{2} - \frac{3}{2P_{xy}} \qquad (5)$$

が成り立つため (等号は $P_{xy} = 1$ の時), サイクリック - サイクリック分割の方が良いことになる. しかしながらこの分割は最悪ケースでは N^3 の係数が 1 となってしまい, 行サイクリックの最悪ケースと変わらない.

ここで列サイクリックの時と同じように $\frac{N}{P_{xy}} = N_p = \text{const.}$ という条件をつける. この時, 最善ケースでは,

$$\max_p \sum_{i=0}^{N-1} d_{p,i} = \frac{2N_p}{3}N^2 + \sqrt{N_p}N^{\frac{3}{2}} + \left(\frac{1}{3} - 2N_p\right)N$$

となり, 最悪ケースでは

$$\max_p \sum_{i=0}^{N-1} d_{p,i} = N_pN^2 + \sqrt{N_p}N^{\frac{3}{2}} - (N_p+1)N + \sqrt{N_p}N^{\frac{1}{2}}$$

となる. したがって最善ケースでは列サイクリックより最大次の係数が小さいことになり, また最悪ケースでも $N_p = 1$ では最大次の係数は一致する. このようにプロセッサ数が増えた場合にもサイクリック - サイクリック分割は有利である.

3 評価

LINPACK Benchmark[1] の問題を使い, 元数をいろいろ変えて LU 分解を行なった. まず始めの実験は係数行列を列サイクリックに分散させて行なった. ここで使った LU 分解のアルゴリズムはほぼ図 3 である. なおこの実験では富士通の並列計算機 AP+[3] 64 台を使用している.

表 1 より, 計算コアの平均は 5.0 MFLOPS/PU で実行されていることが分かる. このプログラムでは多段同時消去を行なっていないためプロセッサの性能は全然出ていない. しかしながら前節で考察した計算モデルを元に, 列サイクリック分割では N^3 の係数は純粋にこの計算モデルで決まることに注目し, 平均の MFLOPS 値を計算すると 6.0 MFLOPS/PU であった.

この 6.0 MFLOPS/PU の性能で計算モデルの理論値と実測値をともにプロットしたのが図 4 である. 理論値と実測値の差を調べてみると, わずかであるが $O(N^2)$ で広がっている. この理論値にはいっていない主な時間は, ピボット選択とメッセージハンドリングオーバーヘッドであるが, このうちピボット選択は $O(N)$, 通信オーバーヘッドは $O(N^2)$ となるため, 通信オーバーヘッドがきいていると考えられる. またこれらの項を計算モデルに加えればより精密な近似が可能だと考えられる.

次は係数行列をサイクリック - サイクリック分割[3]で分散させ, 8 段同時消去を行なう LU 分解で実験を行なった.

この表 2 より, 計算コアはおおよそ 14.7 MFLOPS/PU で実行されていることが分かる. サイクリック - サイクリック分割の場合はピボット選択のされ方により N^3 の

表 1. LU 分解とその計算コアの実行時間と性能 (列サイクリック分割)

元数	LU 分解		計算コア	
	時間	MFLOPS/PU	時間	MFLOPS/PU
1001	3.06	3.4	2.22	4.7
1201	4.81	3.8	3.70	4.9
1401	7.13	4.0	5.71	5.0
1601	10.11	4.2	8.33	5.1
1801	13.83	4.4	11.65	5.2
2001	18.37	4.5	15.74	5.3

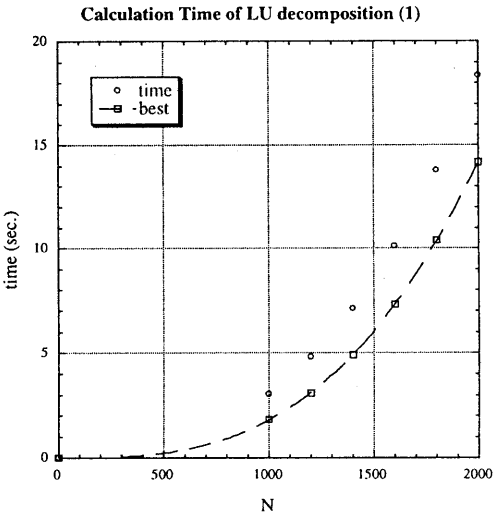

図 4. 列サイクリック分割した LU 分解の実行時間. 下の曲線は計算モデルによる理論値

[3] より正確には !HPF$ DISTRIBUTE (CYCLIC(8), CYCLIC)

表 2. LU 分解とその計算コアの実行時間と性能 (サイクリック - サイクリック分割)

元数	LU 分解		計算コア	
	時間	MFLOPS/PU	時間	MFLOPS/PU
1001	1.03	10.1	0.70	14.9
1201	1.68	10.7	1.24	14.6
1401	2.52	11.4	1.92	14.9
1601	3.71	11.5	2.94	14.5
1801	5.01	12.1	3.99	15.3
2001	6.87	12.1	5.67	14.7

係数が変わるため, 計算モデルから平均の MFLOPS を計算することができない. この平均値を使って, 前節で考察した計算モデルの最善ケース, 最悪ケースとともにグラフにしたのが図 5 である.

サイクリック - サイクリック分割の場合は問題によってピボット選択のされ方が異なり, そのたびに N^3 の係数までもが変わってしまうため, 列サイクリックの時のように簡単ではないが, 図 5 によると実測値は最善ケースと最悪ケースの間に収まっている. しかしながら, サイクリック - サイクリック分割についてもう少し精密な計算モデルを作るためには, 最善ケース, 最悪ケースだけではなく平均のケースについての計算モデルを作る必要があると思われる.

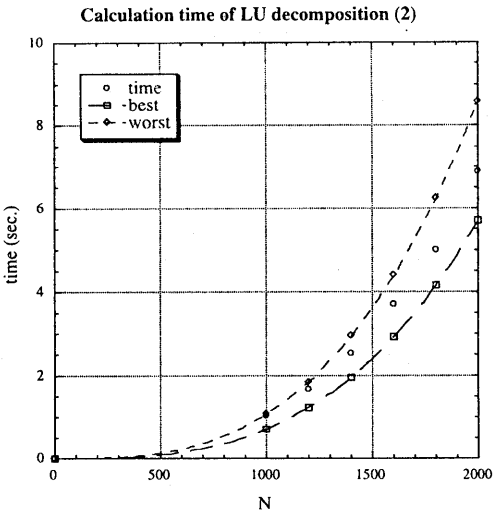

図 5. サイクリック - サイクリック分割した LU 分解の実行時間. 一番下の曲線は最善ケース, 一番上の曲線は最悪ケースのそれぞれの計算時間の予測

4 まとめ

この研究では分散メモリ型並列計算機で効率的な LU 分解のアルゴリズムと計算モデルの構築, そして実機での評価を行なった. 効率的な技法としては, ソフトウェアパイプライニングと非同期実行があり, それぞれピボット行, 列の通信遅延の隠蔽, 他プロセッサのピボット選択の時間の隠蔽を行なうことができる. また通信遅延隠蔽のための条件は大雑把には通信遅延が消去の時間より短いことであり, ピボット選択の時間の隠蔽の条件はこれも大雑把にピボット選択 (送信のオーバーヘッド含む) の時間が消去の時間より短いことである. もちろんその両方を行なう場合は消去の時間よりピボット選択と通信遅延を合わせた時間が短いことが十分条件である.

この条件が満たされる場合, 実行時間は計算量に比例するはずで, それぞれの分散について計算モデルを構築した. 実機で評価を行なったところ, 列サイクリック分割ではその仮定がかなり正しいことが分かった. サイクリック - サイクリック分割ではピボット選択のされ方によって計算モデルが変わってしまい, 平均のケースでのモデルの構築が今後の課題である.

データ分散については, 一般的には多段同時消去を行なった場合ピボット選択の時間が増えるため, ピボット選択と通信遅延が消去の時間を上回らない様に列方向にプロセッサを用意し, 残りは行方向に用意するのが良いと考えられる. 今後, プロセッサの処理能力 (MFLOPS), メッセージハンドリングオーバーヘッドと最適なデータ分散を関連付けたい.

参考文献

[1] Dongarra, J. J., "Performance of Various Computers Using Standard Linear Equations Software," Tech. Rep. CS-89-85, Computer Science Department, University of Tennessee, June 1995.

[2] Golub, G. H. and C. F. V. Loan, *Matrix Computations*. The Johns Hopkins University Press, 1983.

[3] 石畑 宏明, 堀江 健志, 清水 俊幸, 林 憲一, 小柳 洋一, 今村 信貴, 白木 長武, "AP1000+: デザインコンセプト," in *IPSJ SIM Notes, 94-ARC-107*, pp. 153–160, 1994. SWoPP 琉球 '94.