連立一次方程式の直接解法

東京大学 森田直樹

2018 年 5 月 12 日

1 はじめに

本資料は、直接法による疎行列の分解（sparse factorization）とグラフ（graph）についてまとめたものである。直接法は、Gauss の消去法 (LU 分解と前進後退代入) に基づいており、有限回の演算で解を得る堅固な手法である。直接法の特徴は、LU 分解の過程において元々の行列で要素の値が零である部分に零以外の値が出現する、フィルインが発生することである。本資料では、有限要素法から得られる疎な係数行列をもつ連立一次方程式 (1) を考える。

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

ここで、$\boldsymbol{A}$ は $n \times n$ の疎な正定値対称行列、$\boldsymbol{x}$ は解ベクトル、$\boldsymbol{b}$ は右辺ベクトル、n は自由度である。

2 グラフ

2.1 グラフの定義

グラフ $\boldsymbol{G}$ とは、節点集合 $\boldsymbol{V} = \{1, 2, \cdots, n\}$ と、辺集合 $\boldsymbol{E} = \{(i, j) \mid i, j \in \boldsymbol{V}\}$ から決まる構造 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ である [1, 2]。ここで n は節点数である。有向グラフは辺 (i, j) と辺 (j, i) を異なる辺として取扱い、無向グラフは辺 (i, j) と辺 (j, i) を同一とみなす。節点 i の隣接点 j は、隣接集合 $\mathrm{Adj}(i) = \{j \mid (i, j) \in \boldsymbol{E}\}$ で表される。

2.2 係数行列とグラフ

係数行列 $\boldsymbol{A}$ から得られる隣接リスト $\mathcal{A}_{i*}$、$\mathcal{A}_{*j}$ は式 (2)、式 (3) で表される。

$$\mathcal{A}_{i*} = \{j \mid a_{ij} \neq 0,\ i \neq j\} = \mathrm{Adj}(i) = \{j \mid (i, j) \in \boldsymbol{E}\} \tag{2}$$
$$\mathcal{A}_{*j} = \{i \mid a_{ij} \neq 0,\ i \neq j\} = \mathrm{Adj}(j) = \{i \mid (i, j) \in \boldsymbol{E}\} \tag{3}$$

隣接リスト $\mathcal{A}_{i*}$、$\mathcal{A}_{*j}$ に対し、係数行列 $\boldsymbol{A}$ の対角成分 a_{ii} に対応する self-edge (i, i) を考えると、係数行列 $\boldsymbol{A}$ 成分の非零要素と一致する。このとき、係数行列 $\boldsymbol{A}$ から得られるグラフ構造を $\boldsymbol{G}_{\boldsymbol{A}}$ で表す。

3　直接法アルゴリズム

3.1　LU 分解 (外積形式)

LU 分解は、正則な正方行列 $\boldsymbol{A}$ を、下三角行列 $\boldsymbol{L}$ と上三角行列 $\boldsymbol{U}$ を用いて、$\boldsymbol{A} = \boldsymbol{LU}$ と行列分解する手法である。図 1 に、外積形式分解（outer product factorization）を示す。外積形式分解は、ピボット i を用いた i 番目の LU 分解において、残った行列成分を i 行と i 列の外積を用いて更新する手法である。ピボット i に対して、i よりも大きい行列成分を参照するため、right looking アルゴリズムと呼ばれる。

1: **for** $k = 1, 2, 3, \ldots, n$ **do**
2:　**for** $i = k+1, k+2, \ldots, n$ **do**
3:　　$a_{i,k} = a_{i,k}/a_{kk}$
4:　**end for**
5:　**for** $j = k+1, k+2, \ldots, n$ **do**
6:　　**for** $i = k+1, k+2, \ldots, n$ **do**
7:　　　$a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$
8:　　**end for**
9:　**end for**
10: **end for**

図 1　Outer product (right looking) factorization

3.2　LU 分解 (内積形式)

図 2 に、内積形式分解（inner product factorization）を示す。内積形式分解は、ピボット i を用いた i 番目の LU 分解において、i 列自身を分解済みの成分の内積を用いて更新する手法である。ピボット i に対して、i よりも小さい列成分を参照するため、left looking アルゴリズムと呼ばれる。

3.3　LU 分解 (クラウト法)

図 3 に、Crout 法を示す。Crout 法は、ピボット i を用いた i 番目の LU 分解において、i 行と i 列自身を分解済みの成分の内積を用いて更新する手法である。行列成分の更新順序の特徴から、共有メモリ型並列計算機に適していることが特徴である。

```
1: for k = 1, 2, 3, ..., n do
2:   for j = 1, 2, 3, ..., k − 1 do
3:     for i = j + 1, j + 2, ..., n do
4:       a_{i,k} = a_{i,k} − a_{i,j} a_{j,k}
5:     end for
6:   end for
7:   a_{k,k} = a_{k,k}^{−1}
8:   for i = k + 1, k + 2, ..., n do
9:     a_{i,k} = a_{i,k} a_{k,k}
10:  end for
11: end for
```

図 2　Inner product (left looking) factorization

```
1: a_{11} = 1/a_{11}
2: for i = 2, 3, ..., n do
3:   a_{1,i} = a_{1,i} a_{1,1}
4: end for
5: for k = 1, 2, 3, ..., n do
6:   for j = 1, 2, 3, ..., k do
7:     for i = k, k + 1, ..., n do
8:       a_{i,k} = a_{i,k} − a_{i,j} a_{j,k}
9:     end for
10:  end for
11:  a_{k,k} = a_{k,k}^{−1}
12:  for j = 1, 2, ..., k do
13:    for i = k + 1, k + 2, ..., n do
14:      a_{k,i} = a_{k,j} − a_{k,j} a_{j,i}
15:    end for
16:  end for
17:  for j = k + 1, k + 2, ..., n do
18:    a_{k,j} = a_{k,j} a_{k,k}
19:  end for
20: end for
```

図 3　Crout factorization

3.4 Frontal 法

図 4 に、frontal 法を示す。frontal 法は、フロンタル行列 $\boldsymbol{F}$ とアップデート行列 $\boldsymbol{U}$ を用いて LU 分解する、外積形式の直接法である。計算手順を理解するために、正定値対称行列 $\boldsymbol{A}$ を Frontal 法によって Cholesky 分解 $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\boldsymbol{t}}$ を計算することを考える。ここで、$\boldsymbol{A}$ は正定値対称行列、$a_{i,j}$ は $\boldsymbol{A}$ の第 (i,j) 番目の行列要素、$\boldsymbol{F}$ はフロンタル行列、$\boldsymbol{U}$ はアップデート行列、$\boldsymbol{L}$ は下三角行列、$l_{i,j}$ は $\boldsymbol{L}$ の第 (i,j) 番目の行列要素である。

特に図 4 は、行列 $\boldsymbol{A}$ が密行列の場合を示している。ここで、$\boldsymbol{F}^k$ は、ある行番号 $k(1 \leq k \leq n)$ に対するフロンタル行列で、$(n-k+1) \times (n-k+1)$ の大きさをもつ。

$\boldsymbol{U}^0 = \boldsymbol{0}$
for $k = 1, 2, 3, \ldots, n$ **do**
  $\boldsymbol{F}^k = \begin{pmatrix} a_{k,k} & a_{k,k+1} & a_{k,k+2} & \cdots & a_{k,n} \\ a_{k+1,k} & 0 & 0 & \cdots & 0 \\ a_{k+2,k} & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n,k} & 0 & 0 & \cdots & 0 \end{pmatrix}$
  $\boldsymbol{F}^k = \boldsymbol{F}^k + \boldsymbol{U}^{k-1}$
  $l_{k,k} = 1/\sqrt{f^k_{1,1}}$
  for $i = 1, 2, 3, \ldots, n-k$ **do**
    $l_{k+i,k} = f^k_{i,k} l_{k,k}$
  end for
  for $j = 1, 2, 3, \ldots, n-k$ **do**
    for $i = j, j+1, j+2, \ldots, n-k$ **do**
      $u^k_{i,j} = f^k_{i,j} - l_{k+i,k} l_{k+j,k}$
    end for
  end for
end for

図 4　Frontal method (行列 $\boldsymbol{A}$ が密行列の場合)

4　フィルインとシンボリック分解

4.1　フィルイン

正定値対称行列 $\boldsymbol{A}$ は、$\boldsymbol{A} = \boldsymbol{LU}$ の形に LU 分解することができる。ここで、$\boldsymbol{L}$ は下三角行列である。このとき、LU 分解をすることで現れる、$\boldsymbol{G_A}$ に含まれない辺集合をフィルイン（fill-in）と呼び、$\boldsymbol{G_F}$ と表す。フィルインを考慮したグラフ構造 $\boldsymbol{G_A^+}$ は、式 (4) で表される。

$$\boldsymbol{G_A^+} = \boldsymbol{G_{L+L^t}} = \boldsymbol{G_{A+F}} \tag{4}$$

LU 分解の効率的な計算のために、グラフ構造 $\boldsymbol{G_A^+}$ を予め決定しておく必要がある。グラフ構造 $\boldsymbol{G_A}$ からフィルイン $\boldsymbol{G_F}$ を求めることを、シンボリック分解（symbolic factorization）と呼ぶ。

4.2　フィルインの決定

フィルイン $\boldsymbol{G_F}$ の決定のために、グラフ構造 $\boldsymbol{G_A}$ を用いる。ここで、$\mathcal{A}_i$ を $\boldsymbol{A}$ の第 i 列の隣接リスト、$\mathcal{L}_i$ を $\boldsymbol{L}$ の第 i 列の隣接リストとする。このとき、$\mathcal{A}_1$ と $\mathcal{L}_1$ は、明らかに一致する。

最も簡単なフィルインの決定方法は、定理 1 、定理 2 を用いて、$\mathcal{L}_1$ から順に、全ての非零要素を確認していくものである。図 1 にフィルイン決定のアルゴリズムを示す [1]. 定理 1、定理 2 に基づく方法は、演算量 $\mathcal{O}(m^2)$ と見積もられる。ここで、m は隣接リスト $\mathcal{A}$ の要素数である。

定理 1　LU 分解 $\boldsymbol{A} = \boldsymbol{LL}^t$ において、数値の桁落ちを無視すれば、$a_{ij} \neq 0 \Rightarrow l_{ij} \neq 0$ が成り立つ。

定理 2　LU 分解 $\boldsymbol{A} = \boldsymbol{LL}^t$ において、数値の桁落ちを無視すれば、$i < j < k \wedge l_{ji} \neq 0 \wedge l_{ki} \neq 0 \Rightarrow l_{kj} \neq 0$ が成り立つ。

```
1: for i = 1, 2, 3, ··· , n do
2:     for j ∈ A_i do
3:         for k ∈ A_i ∧ j < k do
4:             A_j ⇐ A_j ∪ k
5:         end for
6:     end for
7: end for
```

図 5　定理 1、定理 2 に基づくフィルインの決定

図 1 に示したフィルイン決定のアルゴリズムは、行列サイズと隣接リストの要素数が大きくなるに従って、多くの計算時間を要する。この解決のため、図 2 に、定理 3 に基づいたフィルイン決定のアルゴリズムを示す [3]。定理 3 に基づいた方法は、演算量 $\mathcal{O}(m)$ と見積もられる。

定理 3　Cholesky 分解 $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^t$ において、節点 j が節点 i の親であれば、$\mathcal{L}_i \setminus j \in \mathcal{L}_j$ が成り立つ。ここで、節点 i の親 j は $\min\{j \mid j > i,\ l_{ij} \neq 0\}$ である。

1: **for** $i = 1, 2, 3, \cdots, n$ **do**
2: $\quad j = \min\{j \mid j > i,\ j \in \mathcal{L}_i\}$
3: $\quad \mathcal{A}_j \Leftarrow \mathcal{A}_j \cup \mathcal{A}_i \setminus j$
4: **end for**

図 6　定理 3 に基づくフィルインの決定

4.3　フィルインの決定例

図 3 に、フィルインを考慮する前の 2 値行列 $\boldsymbol{G_A}$ を示す。図 4 に、フィルインを考慮した後の 2 値行列 $\boldsymbol{G_A^+}$ を示す。この行列は、19×19 の対称行列で、便宜的に下三角行列部分のみを示した。非零要素の対角成分を青、非零要素の対角成分以外をグレー、フィルインを赤で示した。

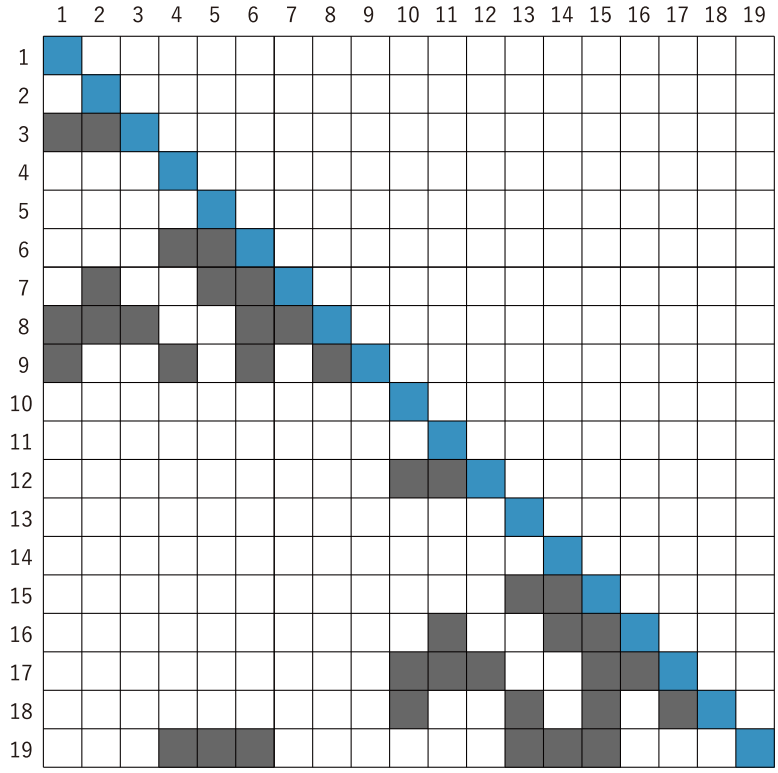

図 7　フィルインを考慮する前の 2 値行列 $\boldsymbol{B}(\boldsymbol{G_A})$

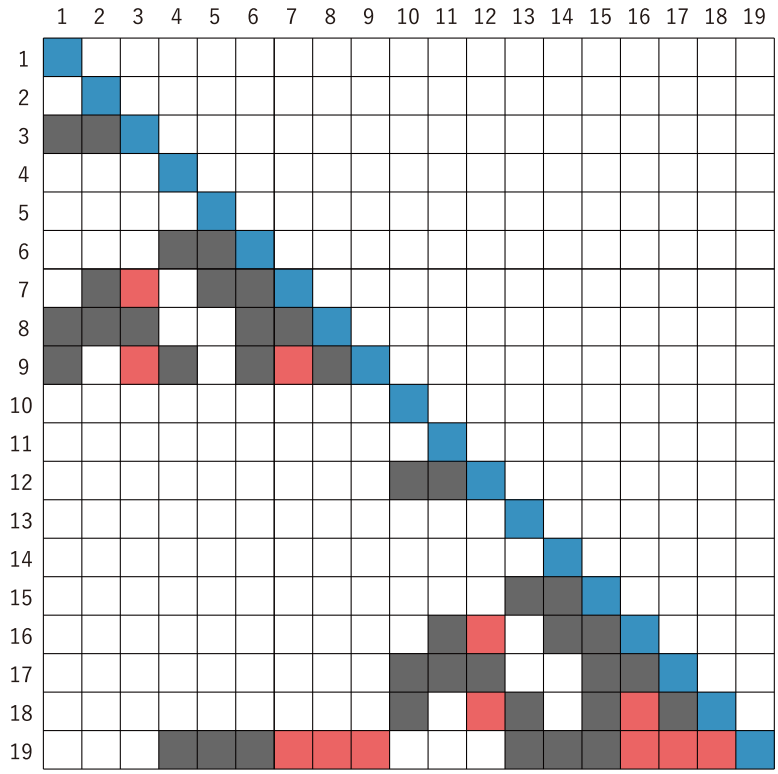

図 8　フィルインを考慮した後の 2 値行列 $\boldsymbol{B}(\boldsymbol{G}_{\boldsymbol{A}}^{+})$

図 5 に、定理 3 に基づくアルゴリズムの例として、図 3 におけるフィルイン決定例を示す。図 5 s(a)〜(d) の順に従って、以下の手続きを実施している。

(a)　節点 1 の親の探索

(b)　親節点 3 にフィルインの追加

(c)　節点 2 の親の探索

(d)　親節点 3 にフィルインの追加

定理 3 に基づくアルゴリズムは、定理 1、定理 2 に基づくアルゴリズムに比べ、効率的にフィルインを決定できることが、確認できる。

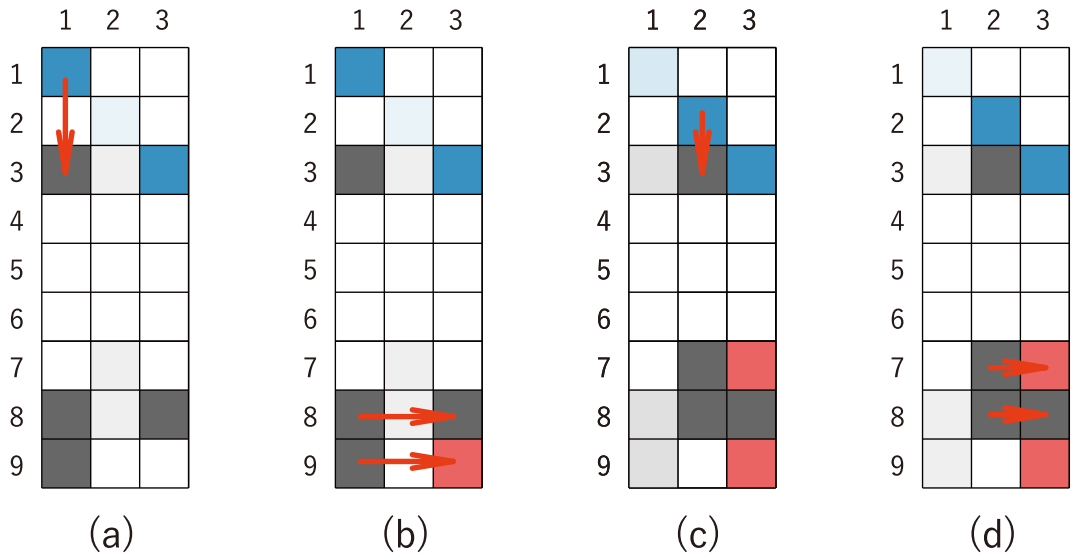

図 9　定理 3 に基づくアルゴリズムによるフィルイン決定例

4.4　リオーダリング

直接法によって疎行列を求解するときは、フィルインの数が元の行列の非零要素数に比べて非常に大きくなり、計算量とメモリ使用量が増大する。フィルインの増大を抑制させるために、適当な置換行列を用いて行と列を置換するリオーダリングを行う必要がある。

リオーダリングの例としては、minimum degree 法、Cuthill-Mackee 法と nested dissection 法などが挙げられる。minimum degree 法と Cuthill-Mackee 法は、LU 分解の各ステップで生じるフィルインを評価値として、フィルインが少なくなるように置換行列を決定する幅優先探索アルゴリズムの一種である。nested dissection 法は、ひとつの領域を分割するセパレータ領域最小化するように、二つの領域に分割することを再帰的に繰り返し、フィルインを削減する手法である。

4.5　直接法における並列計算

これまでに述べた LU 分解のアルゴリズムは、逐次的な計算に基づくため、並列計算を行うには別途、並列計算のための計算手順を考慮する必要がある。ここでは特に、分散メモリ型並列計算機における LU 分解の手法として、nested dissection method、multi-frontal method、SPIKE msethod について述べる。

4.5.1 Nested Dissection Method

nested dissection 法は、ある領域を 2 分割する最小なセパレータ領域を再帰的に決定することで提案された、並列直接法である。式 (5) のような連立一次方程式を考える。

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{f} \tag{5}$$

ここで、セパレータ領域 Γ で 2 分割した領域を考え、その係数行列を行列形式で書き換えると、式 (6) で表される。

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{II}^{(1)} & 0 & \boldsymbol{A}_{I\Gamma}^{(1)t} \\ 0 & \boldsymbol{A}_{II}^{(2)} & \boldsymbol{A}_{I\Gamma}^{(2)t} \\ \boldsymbol{A}_{I\Gamma}^{(1)} & \boldsymbol{A}_{I\Gamma}^{(2)} & \boldsymbol{A}_{\Gamma\Gamma} \end{pmatrix} \tag{6}$$

式 (6) より、内部領域 $\boldsymbol{A}_{II}^{(1)}$ と $\boldsymbol{A}_{II}^{(2)}$ は、それぞれ独立に LU 分解可能であり、その LU 分解による行列成分の更新は、セパレータ領域 $\boldsymbol{A}_{\Gamma\Gamma}$ に集約されることがわかる。このような領域分割を再帰的に内部領域 $\boldsymbol{A}_{II}^{(i)}$ に行うことで、LU 分解の並列計算が可能となる。

4.5.2 Multi-frontal Method

multi-frontal 法は、frontal 行列の演算順序の依存性を考慮することで提案された、並列直接法である。図 10 に、multi-frontal 法のアルゴリズムを示す。ここで、$\boldsymbol{A}$ は正定値対称行列、$a_{i,j}$ は $\boldsymbol{A}$ の第 (i,j) 番目の行列要素、$\boldsymbol{F}$ はフロンタル行列、$\boldsymbol{U}$ はアップデート行列、$\boldsymbol{L}$ は下三角行列、$l_{i,j}$ は $\boldsymbol{L}$ の第 (i,j) 番目の行列要素、$\mathcal{A}_i$ は係数行列 $\boldsymbol{A}$ の第 i 行の非零要素、q_n は $\mathcal{A}_i$ の非零要素数、$\mathcal{L}_i$ は下三角行列 $\boldsymbol{L}$ の第 i 行の非零要素、$\mathrm{index}(i, \mathcal{L}_k)$ は $\mathcal{L}_k$ の第 i 番目の非零要素の行番号であり、$\bigoplus$ は拡張和を示す。

1: $\boldsymbol{U}^0 = \boldsymbol{0}$
2: **for** $k = 1, 2, 3, \ldots, n$ **do**
3: $\boldsymbol{F}^k = \begin{pmatrix} a_{k,k} & a_{k,q_1} & a_{k,q_2} & \cdots & a_{k,q_n} \\ a_{q_1,k} & 0 & 0 & \cdots & 0 \\ a_{q_2,k} & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{q_n,k} & 0 & 0 & \cdots & 0 \end{pmatrix}$
$(q_1, q_2, \ldots, q_n \in \mathcal{A}_k)$
4: $\boldsymbol{F}^k = \boldsymbol{F}^k \bigoplus \boldsymbol{U}^{k-1}$
5: $l_{k,k} = 1/\sqrt{f^k_{1,1}}$
6: **for** $i \in \mathcal{L}_k$ **do**
7: $p = \mathrm{index}(i, \mathcal{L}_k)$
8: $l_{i,k} = f^k_{p,k} l_{k,k}$
9: **end for**
10: **for** $j \in \mathcal{L}_k$ **do**
11: **for** $i \in \mathcal{L}_k$ **do**
12: $p = \mathrm{index}(i, \mathcal{L}_k)$
13: $q = \mathrm{index}(j, \mathcal{L}_k)$
14: $u^k_{p,q} = f^k_{p,q} - l_{i,k} l_{j,k}$
15: **end for**
16: **end for**
17: **end for**

図 10　Multi-frontal method

参考文献

[1] Davis, T., Direct Methods for Sparse Linear Systems, SIAM (2006).

[2] Duff, I., Erisman, A., M. and Reid, J. K., Direct Methods for Sparse Matrices, Clarendon Press Oxford (1986).

[3] Gupta, A., Karypis, G. and Kumar, V., Highly Scalable Parallel Algorithms for Sparse Matrix Factorization, IEEE Transactions on Parallel and Distributed Systems, Vol. 8, No. 5, pp. 502–520 (1997).