내부점 선형계획법에서의 멀티프런탈방법에 관한 연구
(A study on the Multifrontal Method in Interior Point Method)

김병규*, 박순달*

* 서울대학교 산업공학과

ABSTRACT

선형계획법의 해법으로 최근에는 내부점기법(Interior Point Method)가 관심을 끌고 있다. 이 내부점 기법은 계산복잡도 뿐만 아니라 수행속도면에서도 우수한 결과를 보이고 있다. 이 방법은 매 회 대칭양정치(Symmetric Positive Definite)인 선형시스템을 풀어야 하는데 이 과정이 전체 내부점 수행시간의 80-90%를 차지한다. 따라서 내부점 기법의 수행속도는 대칭양정치인 선형시스템을 효율적으로 푸는 방법에 달려 있다. 대칭양정치인 선형시스템을 풀기위해서는 상하분해를 이용하게 되는 데 가우스소거를 이용해서 상하 분해를 하는 경우 매 단계에서 행렬의 모든 요소를 가지고 있을 필요가 없다. 행렬의 모든 요소에 대한 정보를 동시에 필요로 하지 않는다. 즉, 현 단계에서 가우스소거와 관련된 열들에 대한 정보만 있으면 상하 분해가 가능하고 이러한 개념을 이용한 방법이 프런탈방법이다. 프런탈 방법은 대형 선형계획 문제를 풀기에 유리하다는 장점이 있다. 이러한 프런탈 방법을 확장해서 동시에 여러 개의 프런탈을 계산하는 방법이 멀티프런탈방법이다. 이 방법은 알고리듬 자체가 병렬처리에 적합하기 때문에 병렬처리과 관련해서도 많은 연구가 수행되고 있다.

본 연구에서는 삭제나무(Elimination Tree)를 이용한 프런탈방법과 프런탈방법에 슈퍼노드의 개념을 도입한 슈퍼노들 프런탈방법등에 대해서 이제까지의 연구 현황을 알아보고 프런탈방법에 적합하고 효율적인 자료 구조와 멀티프런탈 방법에 적용 가능한 병렬알고리듬에 대하여 연구하고자 한다.

본 연구결과 기대효과로는 프런탈 방법에 적합하고 효율적인 자료 구조와 멀티프런탈 방법에 적용 가능한 병렬알고리듬을 개발함으로써 내부점 선형계획법의 수행속도의 개선에 도움이 될 것이다.

1. 서론

최근의 선형계획법의 해법으로 내부점기법 (Interior Point Method)이 크게 관심을 끌고 있다[9]. 이 내부점 기법은 계산 복잡도(Computational Complexity) 면에서 다항시간(Polynomial Time)의 복잡도를 가지고 있으며[12] 수행속도면에서 우수한 결과를 보이고 있다. 대형 문제에 대한 실험 결과에 의하면 문제의 규모가 커질 수록 단체법(Simplex Method)에 비해 내부점 선형계획법의 수행속도가 우수한 것으로 나타났다[4]. 내부점기법의 중요한 장점 중에 하나는 문제의 크기에 둔감하다는 것이다. 대부분의 문제들은 20-80회(Iteration)정도에 풀리는 것으로 나타났다[13]. 따라서, 단체법이 진

입 변수와 탈락 변수를 잘 선택함으로써 수행횟수을 줄이고자 하는데 반해 내부점기법에서는 대체적으로 수행횟수가 일정한 횟수 이내에 있으므로 각 단계의 수행시간을 줄이는 것이 주요 관심사가 된다.

모든 내부점 기법은 $(A\Theta A^T) = b$ 의 대칭양정치(Symmetric Positive Definite)인 선형시스템을 풀어야 하는데 이 과정이 전체 내부점 수행시간의 80-90%를 차지한다. 따라서 내부점 기법의 수행속도는 위의 대칭양정치인 선형시스템을 효율적으로 푸는 방법에 달려 있다. 대칭양정치인 $(A\Theta A^T)$ 행렬은 기억 장소의 제한으로 LL^T형태로 상하 분해를 하고 하삼각 행렬 L만을 보관하게 된다. 내부점 기법의 각 회에서는 대각 행렬 Θ 만 변하므로 $(A\Theta A^T)$ 을 상하 분해할 때 하삼각 행렬 L의 비영요소(Nonzero Element)의 값만이 매회 변할 뿐 그 비영요소의 위치는 변하지 않는다. 따라서 비영요소의 구조를 한번 결정하면 이후에는 대각 행렬 Θ 의 변화에 따른 비영요소들의 값만 구하면 된다. 이 때 L 의 비영요소의 구조를 정하는 과정을 심볼릭 팩토리제이션(Symbolic Factorization) 이라 하고 매회 그 값을 구하는 과정을 뉴메릭 팩토리제이션(Numeric Factorization) 이라 한다. 이러한 뉴메릭 팩토리제이션은 매 회 수행되기 때문에 이 과정에서의 속도 향상은 큰 의미를 지닌다.

대칭양정치인 희소 행렬을 가우스소거(Gaussian Elimination)를 이용해서 상하 분해를 하는 경우 매 단계에서 행렬의 모든 요소를 가지고 있을 필요가 없다. 즉, 모든 요소에 대한 정보를 필요로 하지는 않는다. 현 단계에서 가우스소거와 관련된 열(Column)들에 대한 정보만 있으면 상하 분해가 가능하고 이러한 개념을 이용한 방법이 프런탈(Frontal) 방법이다[8]. 프런탈 방법은 대형(Large Scale) 문제를 풀기에 유리하다는 장점이 있다. 프런탈 방법을 확장해서 동시에 여러 개의 프런탈을 계산하는 방법이 멀티프런탈(Multifrontal) 방법이다. 이 방법은 알고리듬 자체가 병렬처리(Parallel Processing)에 적합하기 때문에 병렬처리를 수행하면 수행속도를 상당히 개선할 수 있다. 따라서 최근에 활발히 연구되고 있는 병렬처리와 관련해서 연구가 활발히 진행되고 있는 중이다. 본 논문에서는 제거나무(Elimination Tree)[11]를 이용한 프런탈방법과 프런탈방법에 Indistinguishable 노드[15]의 개념을 이용한 슈퍼노들 프런탈방법[3][8]등에 대해서 이제까지의 연구 현황을 알아보고 프런탈방법에 적합하고 효율적인 자료 구조와 멀티프런탈 방법에 적용 가능한 병렬알고리듬에 대하여 연구하고자 한다.

2. 프런탈방법과 병렬처리

2.1. 프런탈방법

프런탈 방법이란 희소 행렬(Sparse Matrix)전체의 분해를 조그만 밀집된(dense)행렬들의 분해를 통해서 이루고자 하는 것이다. 일반적인 선형시스템을 풀기 위해 가우스 소거를 하는 경우 행렬 전체 요소를 가지고 있을 필요가 없이 현 단계에서 가우스 소거에 참여하는 열들만 가지고 있으면 가우스 소거가 가능하다. 이러한 상황은 특히 행렬이 밴드(Band)형태를 띠는 경우에 더욱 두들어 진다[16]. 이 때 가우스 소거에 참여하는 열들의 집합을 프런탈(Frontal)이라고 하며 그 열들로 이루어

진 행렬을 프런탈 행렬(Frontal Matrix)이라고 한다[16]. 이 프런탈 행렬은 크기가 작고 밀집된 행렬의 형태를 띠기 때문에 Full Matrix형태로 보관하는 편이 비영요소만을 보관하는 것보다는 전체적인 경상비(Overhead)가 적고 계산상의 이득이 된다. 또한 행렬 전체를 보관하므로 직접 주소법(Direct Addressing)이 가능해지므로 수행속도면에서도 장점을 가지게 된다. 이러한 프런탈 방법은 기억 장소가 제한적인 경우에 더욱 유용하게 되고 기존의 방법들에 비해 대형 문제를 푸는 데 있어서 장점을 지닌다.

프런탈개념을 좀 더 확장하면 동시에 독립적인 여러 개의 프런탈을 활용할 수 있다. 즉, 동시에 여러 개의 프런탈을 분해함으로써 전체적인 수행시간을 줄일 수 있는 것이다. 이러한 방법을 멀티프런탈방법(Multifrontal Method)이라고 하며 이 방법은 기본적으로 각각의 프런탈이 독립적으로 계산 가능하기 때문에 개개의 프런탈을 서로 다른 프로세서(Processor)에 할당함으로써 병렬처리가 가능하다. 이러한 속성으로 인해서 최근의 병렬처리와 관련된 컴퓨터 구조(Architecture)나 알고리듬의 연구와 함께 관심이 되는 분야이다. 주로 어떤 특정한 컴퓨터 구조하에서 가우스 소거의 성능이나 복잡도, 멀티프런탈 방법 등에 대한 연구가 진행되고 있다.

프런탈에 슈퍼노드(Supernode)의 개념[15]을 도입한 방법으로 슈퍼노들 프런탈방법(Supernodal Multifrontal Method)[3]이 있는데 이 방법은 프런탈방법이 한 번에 한 열의 분해를 소거하는 데 반해 슈퍼노드로 대표되어지는 몇몇의 열들을 한꺼번에 분해하고자 하는 방법을 말한다.

프런탈이라는 용어는 Irons에 의해 처음으로 소개된다. 하지만 본격적인 연구는 프런탈 방법을 일반화한 멀티프런탈방법에 대해 Duff & Reid[7]가 논문을 발표하면서 부터이다. 이 논문이 발표된 이후로 주로 유한요소(Finite Element Applications)분야에서 많은 활용을 보여 왔지만 최근에는 최적화문제 특히 내부점기법을 이용한 선형계획법에서 그 활용도를 높이고 있다.

프런탈 방법은 지금까지 비교적 개념적으로는 정리가 잘되어 있지만[8] 이를 구현하는 데 있어서는 구현 방법에 따라서 성능에 많은 차이를 내고 있다. 따라서 이를 효율적으로 구현하는데 있어서는 여러 가지 기법들을 필요로 한다.

프런탈 방법을 구현하는 데 있어서 반드시 언급해야 할 사항이 제거나무이다. 이 제거나무가 역할은 Liu[11]에 의해 잘 설명되고 있고, 프런탈 방법과 연관지어서는 프런탈행렬과 수정행렬(Update Matrix)을 보관하는 순서와 그리고 스택구조(Stack Structure)을 이용하기 위한 후순서(Postorder)을 구하는 데 사용된다[8]. 한편 기억 장소를 최소화하기 위한 알고리듬 또한 제시되어 있다[8]. 프런탈방법의 실질적인 성능 향상은 슈퍼노드의 개념을 이용한 슈퍼노들 멀티프런탈방법이다. 이 방법은 Duff & Reid[7][8]에 의해 제안되었다. 프런탈 방법을 구현하는 데 있어서 여러 가지 기법중 Full Matrix기법은 Duff & Reid[8]에 의해 제안되었으며 Local Indices기법은 Schreiber[8]에 의해, Relaxed Supernode는 Ashcraft & Grimes[6]에 의해 제안되었다. 최근에 Ho-Won Jung, Marsten & Saltzman[3]은 슈퍼노들 멀티프런탈방법에서 loop unrolling기법을 효율적으로 구현함으로써 수행속도에서 10-50%의 성능 향상을 이루었다.

2.2. 병 렬 처 리

병렬 처리는 지금까지의 기술력으로도 엄청난 시간이 걸리는 계산을 좀 더 빠른 시간 내에 하고

자 하는 의도에서 시작되었고 이는 자연현상에서도 많이 발견되는 자연스러운 방법이다. 이 방법은 컴퓨터 구조와 매우 밀접하게 관련이 되어 있다. 여러 개의 프로세서(Processor)들은 서로 정보를 교환하기 위해서는 공유 메모리와 메시지 패싱의 2가지 방법을 사용한다[18][19][20]. 공유 메모리(Shared Memory)를 이용한 컴퓨터는 Multiprocessor라고도 불리며 개개의 프로세서들이 공유 메모리를 엑세스해서 메모리 상에 있는 여러 데이타값들과 자료 구조들을 공유한다. 메시지 패싱(Message Passing)은 각 프로세서에게 지역 메모리(local memory)를 두고 각 프로세서들 간의 통신은 메시지 패싱을 통해서 하는 방법으로 이러한 형태의 병렬 컴퓨터를 Multicomputer라고 한다. 병렬처리에서는 수행속도의 개선이 주목적이므로 주로 곱셈이나 나눗셈등 기본이 되는 산술 연산(Arithmetic Operation)들을 병렬적으로 처리함으로써 계산을 효율적으로 하고자 하는 노력이 있어 왔다[14][17].

위에서 언급한 프런탈이론이 등장한 이후로 이를 적용하려는 시도가 있어 왔다. 이는 대부분의 병렬 알고리듬이 그렇듯이 주로 하드웨어적인 특수한 환경, 즉, 특정 컴퓨터 구조를 설정한 후에 그 환경에 적합한 알고리듬이나 계산 방법을 찾아보고자 하는 시도들이었다.

3. 프런탈 방법

3.1 Factorization by rows, columns and submatrices

1) row-Cholesky (또는 bordering scheme)
   L의 요소들을 행별(Rowwise)로 구해나간다. 각행을 구하기 위해서는 매번 삼각 시스템(triangular system)을 풀어야한다.
2) column-Cholesky (Factorization by columns)
   L의 요소들을 열별(Columnwise)로 구해나간다. 가장 널리 사용되는 방법이다.
3) submatrix-Cholesky (Factorization by submatrix)
   이 방법은 cholesky factor의 각 열을 만들때마다 앞으로 분해될 부분행렬에 대한 이 열의 모든 기여도가 같이 계산된다. Multifrontal방법도 일종의 submatrix-Cholesky방법으로 볼 수 있다.

3.2. Outer-product updates

SPD인 행렬 A을 LL^T 로 분해할 때 outer-product formulation를 이용하면 공식를 이용해서 바로 Cholesky Factor의 한 열을 구할 수 있다. 공식은 다음과 같다.

$$A=\begin{pmatrix} d & v^T \\ v & C \end{pmatrix}=\begin{pmatrix} \sqrt{d} & 0 \\ \frac{v}{\sqrt{d}} & I \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & C-\frac{vv^T}{d} \end{pmatrix}\begin{pmatrix} \sqrt{d} & \frac{v^T}{\sqrt{d}} \\ 0 & I \end{pmatrix}$$

여기서 d : 첫번째 대각 원소 , v : (n-1) 벡터

$C-\frac{vv^T}{d}$: 앞으로 분해될 부분행렬

지금 이 단계에서 한 단계 더 나아가면 block form로 분해가 가능하다.

$$A=\begin{pmatrix} B & V^T \\ V & C \end{pmatrix}=\begin{pmatrix} L_B & 0 \\ VL_B^{-T} & I \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & C-VB^{-1}V^T \end{pmatrix}\begin{pmatrix} L_B^T & L_B^{-1}V^T \\ 0 & I \end{pmatrix}$$

$B=L_BL_B^T$: (j-1) x (j-1)

Schur complement $C-VB^{-1}V^T$ 은 앞으로 분해될 부분행렬이다.

여기서 $-VB^{-1}V^T$ 은 (n-j+1) x (n-j+1)행렬이고 이 행렬은 처음 j-1개의 열들로 부터 $C-VB^{-1}V^T$ 을 만들기 위해 C에 가해진 update contrbution을 나타낸다. 따라서, 이 update contribution submatrix은 Cholesky factor L의 처음 j-1개의 열들로 표현이 가능하다. 즉,

$$-VB^{-1}V^T=-(VL_B^{-T})(L_B^{-1}V^T)=-\sum_{k=1}^{j-1}\begin{pmatrix} l_{j,k} \\ \vdots \\ l_{n,k} \end{pmatrix}(l_{j,k}\ \cdots\ l_{n,k})$$

이 행렬은 Cholesky factor L의 처음 j-1개열에서 나온 outer-product updates의 합으로 생각할 수 있다. 실제로 multifrontal방법은 행렬이 희소할 때 부분 행렬 $-VB^{-1}V^T$ 에서 outer-product 을 어떻게 효과적으로 관리하는냐하는 방법이라고 할 수 있다.

3.3. Elimination tree

3.3.1 elimination tree의 정의

여기서 A는 대칭양정치은 행렬이고 irreducible(즉, block들로 분해가 되지 않는다)하다고 가정한다. 이 때

① elimination graph

G(A) = (X(A), E(A))
단, X(A) = { $x_1, x_2, \cdots, x_n$ } , E(A) = { (i,j) | $a_{ij} \neq 0$ }

② elimination tree : T(A)

L을 cholesky factor라고 할 때 elimination tree는 다음과 같이 정의된다.

PARENT(j) = min { i | $l_{ij} \neq 0$, i > j }

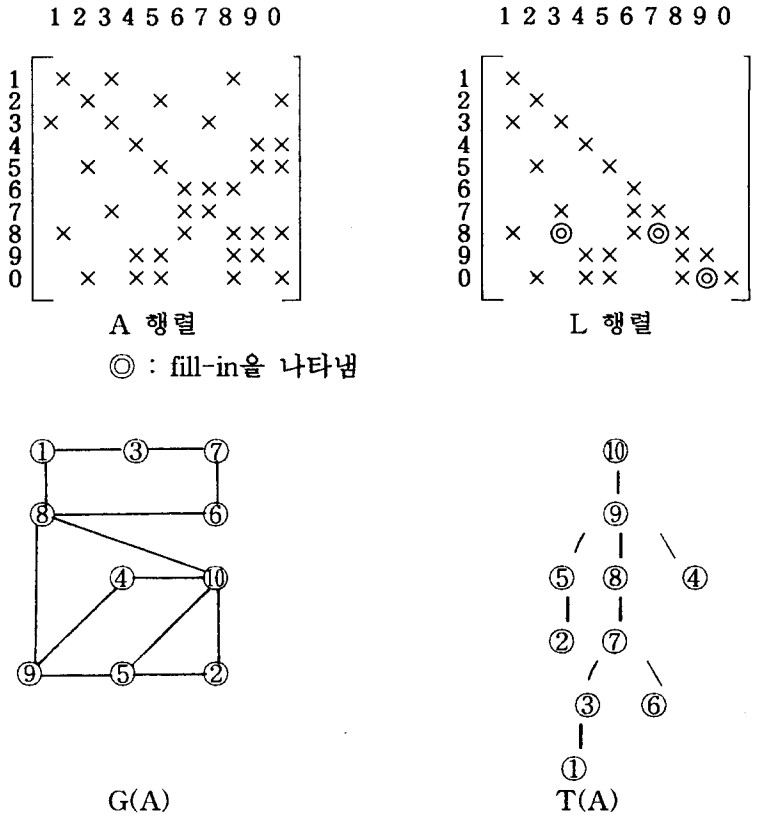
완벽성을 위해서 PARENT(n) = 0 으로 정의한다.

③ A가 irreducible하면 G(A)는 연결된 그래프(connected graph)이다.

④ 정리3.1

i > j, $l_{ij} \neq 0$ 이면 L_{*i} 는 L_{*j}에 의존한다. ■

다음은 삭제 그래프와 삭제 나무의 예를 보여준다.



[그림 3.1] 삭제 그래프와 삭제 나무

3.3.2 Row/Column Structure

이제부터 $Adj_G(v)$ 은 그래프 G에서 v 노드에 인접한 노드들의 집합,

$Adj_G(S)$은 $\{ x \notin S \mid x \in Adj_{G(v)} \text{ for } some\ v \in S \}$,

T[x_j]은 x_j을 root로 하는 subtree라고 정의한다.

① 열구조(각 열에서 비영요소의 행 번호)

정리3.2

cholesky factor 에서 j열의 구조(비영요소의 행번호)는 다음과 같다.,

$$Adj_{G(A)}(T[x_j]) \cup \{x_j\} = \{ x_i \mid l_{ij} \neq 0,\ i \geq j \}. \blacksquare$$

예를 들어 노드 3번을 보면 T[3] = { 1, 3 } 이 되고,

$Adj_{G(A)}$(T[3]) = { 7, 8 } 은 3번째 열의 비영요소의 위치가 된다.

② 행구조(각 행에서 비영요소의 열 번호)

row subtree $T_r[x_i]$ 을 다음과 같이 정의하면

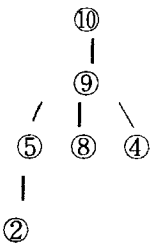
$$T_r[x_i] = \{\ x_j \mid l_{ij} \neq 0,\ i \geq j\ \}$$

이 row subtree는 $T_r[x_i]$은 i행의 구조를 나타내고 이 tree는 elimination tree에서 x_i을 root로 하는 잘려진(pruned) 나무가 된다.

정리3.3

x_j 가 $T_r[x_i]$ 의 leaf node이기 위해서는 $a_{ij} \neq 0$이고 x_j의 모든 자손(descendant)노드 x_k에 대해서 $a_{ik} = 0$ 이어야 한다. ■

[그림3.1]에서 위의 정리를 이용하면, $T_r[10]$ 의 경우 다음과 같다.



[그림 3.2] 삭제 나무에서 행의 구조

이 tree의 leaf node는 { 2, 4, 8 } 이고 이 노드들은 정리 3.2를 만족한다.

3.3.3 Elimination Tree을 만드는 방법

Tarjan이 제시한 방법으로 다음의 3가지 operations을 사용한다.

makeset(x) : 원소가 x하나뿐인 새로운 집합을 만든다.

find(x) : x가 속한 집합에서 그 집합의 대표노드를 찾아서 return한다.

link(x,y) : x가 속한 집합과 y가 속한 집합의 합집합을 만들고 합집합의 대표노드를 return 한다.

여기서 대표노드는 그 집합의 root로 한다.

이 3가지 operation을 이용해서 여러개의 작은 tree들을 결국 하나의 tree로 합치는 과정을 통해서 elimination tree을 만드는 방법이다.

3.4. 일반적인 형태의 SPD행렬에 적용할 수 있는 multifrontal 방법

일반적인 형태의 행렬에 작용하기 위해서 먼저 subtree update matrix, update matrix 그리고 frontal matrix을 정의한다.

1) subtree update matrix at column j

$i_0, i_1, i_2, \cdots, i_r$ 을 Cholesky factor L의 j번째 열 L_{*j}에서 비영요소의 행번호라 하자. 여기서 $i_0 = j$이고 따라서 L_{*j}은 r개의 off-diagonal을 가진다. 그러면

$$\overline{U_j} = - \sum_{k \in T[\ j\ ] - \{\ j\ \}} \begin{pmatrix} l_{j,k} \\ l_{i_1,k} \\ \vdots \\ l_{i_r,k} \end{pmatrix} (\, l_{j,k}\ \ l_{i_1,k} \cdots\ l_{i_r,k})$$

여기서 T[j]는 elimination tree에서 j 번째 노드의 자손노드들의 집합으로 j을 root로 하는 subtree로 볼 수있다.

이렇게 정의된 $\overline{U_j}$ 을 subtree update matrix라고 하며 이 행렬은 (r+1) x (r+1) 행렬이 되고 elimination tree에서 j번째 노드의 자손노드들의 기여정도를 나타내게 된다.

2) frontal matrix F_j

frontal 행렬 F_j 을 다음과 같이 정의하면

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & 0 & \\ a_{i_r,j} & & & \end{pmatrix} + \overline{U_j}$$

F_j 은 (r+1) x (r+1) 행렬이고 r+1은 L_{*j}의 비영요소의 수가되며

이를 분해하면

$$F_j = \begin{pmatrix} l_{j,j} & \mathbf{0} \\ l_{i_1,j} & \\ \vdots & I \\ l_{i_r,j} & \end{pmatrix} \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{0} & U_j \end{pmatrix} \begin{pmatrix} l_{j,j} & l_{i_1,j} \cdots & l_{i_r,j} \\ \mathbf{0} & I \end{pmatrix}$$

따라서 L_{*j} 을 구할 수 있다. 여기서 U_j 을 update matrix라고 이 행렬은 full이 되고 다음과 같이 구할 수 있다.

정리 3.4

update matrix U_j은 Cholesky factor에서 T[j]에 속한 노드들의 열들에 의해서 다음과 같이 구해진다.

$$U_j = - \sum_{k \in T[j]} \begin{pmatrix} l_{i_1,k} \\ \vdots \\ l_{i_r,k} \end{pmatrix} (l_{i_1,k} \cdots l_{i_r,k}). \blacksquare$$

여기서 $\overline{U_j}$ 은

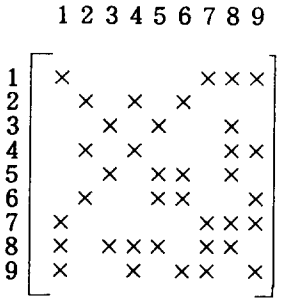
- frontal 행렬 F_j을 만들기 위해서 사용되고
- 차원은 (r+1) x (r+1) (r+1은 L_{*j}의 비영요소 수)
- T[j]-{j}에 있는 모든 열들의 기여도를 나타낸다.

반면에 U_j 은

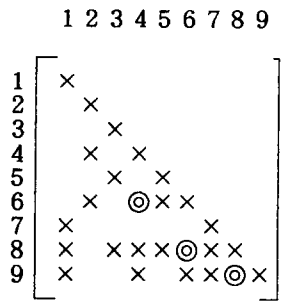
- frontal 행렬 F_j에서 소거를 한 단계 거쳐서 만들어지는 행렬이고
- 차원은 (r) x (r)이며 full matrix 가 된다.
- subtree T[j] 에 있는 모든 열들의 기여도를 나타낸다.

의 성질을 만족한다.

예를 들어, A 와 cholesky factor L이 다음과 같은 때



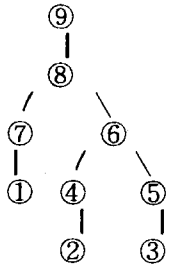
A 행렬



L 행렬

◎ : fill-in을 나타냄

elimination tree는 다음과 같이 구할 수 있고,



5. Tree Postordering의 이용

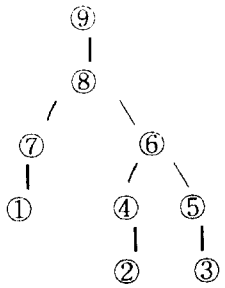
update matrix을 보관하고 찾는 방법이 multifrontal 방법을 구현하는 데 있어서 중요한 역할을 한다. 먼저 위의 예제의 경우를 들면 F_4가 processing될 때 U_2만 사용하고 U_1과 U_3는 나중에 사용되기 때문에 U_1과 U_3는 어딘가에 보관해야한다. F_4에서 U_4가 만들어지면 F_5을 위해서 저장되어야하고 F_5을 만들때 U_3이 필요하기 때문에 retrieve해야 한다. 따라서 U_j와 F_j가 만들어지고 저장되는 순서를 잘 조정하면 상당히 효율적인 구현을 할 수 있다. 그러한 저장 순서를 위한 방법의 하나로 postordering을 고려한다.

1) topological ordering

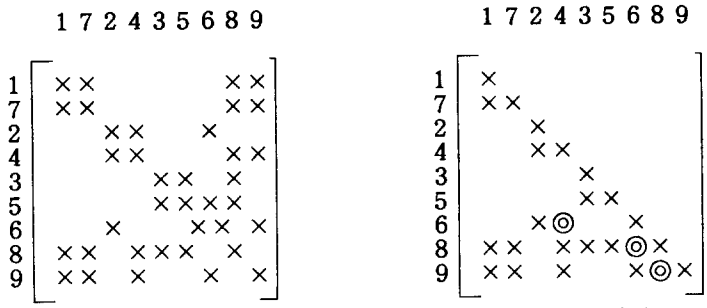
elimination tree에서 각 노드가 부모노드보다 먼저나오는 순서를 topological order라고 한다. 이에는 많은 순서가 가능하다.

2) postordering

subtree에 있는 노드들이 연속적인 순서를 가지는 topological ordering을 말한다. 이 postordering을 사용하면 update matrix을 LIFO(Last In First Out)형태로 즉, stack 자료구조를 이용하여 운영을 할 수 있다. 즉, 관련된 frontal 행렬에서 update 행렬를 만들면 이를 stack에 저장(push)하고 현재 frontal 행렬을 만드는데 필요하면 pop을 하게 되면 위의 예제에 대응하는 postordering과 stack에 저장되는 내용은 다음과 같다.



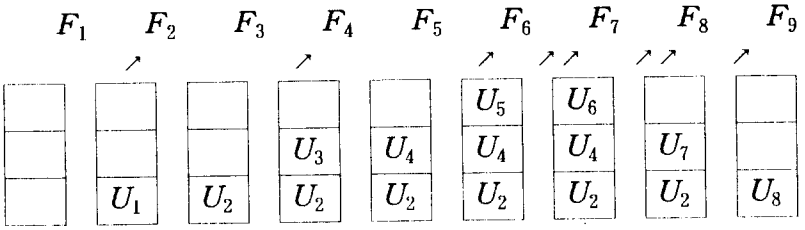
[그림 3.3] postordering 된 삭제 나무



ordered A 행렬 ordered L 행렬

◎ : fill-in을 나타냄

[그림 3.4] ordering된 A행렬과 L행렬



↗ : pop 하는 과정을 나타냄

↗↗ : 2개를 pop 하는 과정을 나타냄

상자에는 stack에 저장된 U_i을 보여준다.

[그림 3.5] 스택을 이용한 update matrix의 운용

6. 슈퍼노들 프런탈 방법

일반적으로 열또는 행들은 supernode[15]라는 개념을 이용해서 하나의 계산 단위(one computational unit)으로 그룹화할 수 있다. 따라서 매번 촐레스키분해의 한 열만 만들게 아니라, 가능하면 여러 개의 열을 한꺼번에 하는 것이 수행속도면에서 유리하다.

5. 참 고 논 문

[1] 박순달, "OR(경영과학)-이론과 전산연습" 1991. 민영사

[2] 박순달, "선형계획법",1992. 민영사

[3] HO-WON JUNG, ROY E. MARSTEN, MATTHEW J. SALTZMAN, Numerical Factorization Methods for Interior Point Algorithms, ORSA Journal on Computing Vol 6, N0. 1, Winter 1994

[4] Ilan Adler and Narendra Karmarkar and Mauricio G.C. Resende and Geraldo Veiga, Data Structures and Programming Techniques for the Implementation of Karmarkar's Algorithm, ESRC 87-12 (December 1987)

[5] Ilan Adler and Narendra Karmarkar and Mauricio G.C. Resende and Geraldo Veiga, An Implementation of Karmarkar's Algorithm For Linear Programming, ORC 86-8 (May 1986 : Revised June 1987)

[6] CLEVE ASHCRAFT and ROGER GRIMES, The Influence of Relaxed Supernode Partitions on the Multifrontal Method, ACM Transactions on Mathematical Software, Vol. 15, No. 4, December 1989, pp. 291-309

[7] I. S. DUFF and J. K. REID, The Multifrontal Solution of Indefinite Sparse Symmetric Linear Equations, ACM Transactions on Mathematical Software, Vol. 9, No. 3, September 1983, pp. 302-325

[8] JOSEPH W.H. LIU, The Multifrontal Method for Sparse Matrix Solution : Theory and Practice, SIAM REVIEW Vol. 34, No. 1, March 1992, pp. 82-109

[9] Ho-Won Jung, Direct Sparse Matrix Methods for Interior Point Algorithms, Ph. D. Dissertation

[10] JOSEPH W. H. LIU, On the Storage Requirement in the Out-of-Core Multifrontal Method for Sparse Factorization, ACM Transactions on Mathematical Software, Vol. 12, No. 3, September 1986, pp. 249-264

[11] JOSEPH W. H. LIU, The Role of Elimination Trees in Sparse Factorization, SIAM J. MATRIX ANAL. APPL. Vol. 11, No. 1, pp. 134-172, January 1990

[12] N. KARMARKAR, A New Polynomial-Time Algorithm for Linear Programming, COMBINATORICA 4(4) (1984) pp. 373-395

[13] Marsten R, R. Subramanian, and M. Saltzman, Interior Point Methods for Linear Programming : Just Call Newton Lagrange, adn Fiacco & McCormick, Interfaces 20,4, pp. 105-116, 1990

[14] S. Lakshmivarahan, Sudarshan K. Dhall, Analysis and Design of Parallel Algorithms ; Arithemetic and Matrix Problems, McGRAW-HILL INTERNATIONAL EDITIONS

[15] The Evolution of the Minimum Degree Ordering Algorithm, SIAM REVIEW Vol.31, No.1, pp.1-19, March 1989, ALAN GEORGE and JOSEPH W.H. LIU

[16] I.S. Duff, A.M. Erisman, J.K. Reid, Direct Methods for Sparse Matrices, CLARENDON PRESS OXFORD, 1986

[17] D.J. Kuck, K. Maruyama, Time Bounds on the Parallel Evaluation of Arithmetic Expressions, SIAM J. COMPUT Vol. 4, No.2, June 1975

[18] BRUCE P. LESTER, The Art of Parallel Programming, Prentice Hall International Editions

[19] Joseph JaJa, Introduction to Parallel Algorithm, Addison-Wesley Publishing Company

[20] SELIM G. AKL, The Design and Analysis of Parallel Algorithms, Prentice Hall International Editions

[21] F. THOMSON LEIGHTON, Introduction to Parallel Algorithms and Architectures, Morgan Kaufmann Publishers, San Mateo, California