여러 가지 컴퓨터 메모리 참조 방법과 자료구조에 대한 단체법 프로그램 수행 속도의 비교*

박찬규** · 임성묵** · 김우제*** · 박순달**

Experimental Comparisons of Simplex Method Program's Speed with Various Memory Referencing Techniques and Data Structures

Chan Kyoo Park · Sung Mook Lim · Woo Jae Kim · Soon Dal Park

〈Abstract〉

In this paper, various techniques considering the characteristics of computer memory management are suggested, which can be used in the implementation of simplex method. First, reduction technique of indirect addressing, redundant references of memory, and scatter/gather technique are implemented, and the effectiveness of the techniques is shown. Loop-unrolling technique, which exploits the arithmetic operation mechanism of computer, is also implemented. Second, a subroutine frequently called is written in low-level language, and the effectiveness is proved by experimental results. Third, row-column linked list and Gustavson's data structure are compared as the data structure for the large sparse matrix in LU form. Last, buffering technique and memory-mapped file which can be used in reading large data file are implemented and the effectiveness is shown.

1. 서론

선형계획법 문제의 해법 중의 하나인 단체법의 구현은 진입·탈락변수 선정 방법, 기저역행렬 유지 방법 등의 알고리즘 선정 뿐만 아니라 자료구조와 실제 구현 방법에 의해 그 효율이 달라진다[1][2]. 효율적인 단체법 프로그램은 자료구조의 설계와 코딩 단계에서 컴퓨터 내부 산술 연산이 수행되는 과정을 충분히 이해하고 이를 잘 이용할 수 있을 때에 가능해진다.

대부분의 수치연산에 관련된 코드들은 컴퓨터의 메모리 구조나 수치연산 과정의 특성 등을 활용하는 면에 많은 관심을 두고 있다. 특히 컴퓨터의 메모리 관리 방법을 잘 활용하면 계산시간을 상당히 줄일 수 있다. 또한 벡터 컴퓨터, Superscalar Computer, 병렬 컴퓨터[8]에 프로그램을 효율적으로 구현하고자 하는 노력도 시도되고 있다.

현재까지 단체법 프로그램에 대해 컴퓨터 메모리 관리 특성을 이용하는 기법에 대한 연구는 거의 없었다. 단체법 프로그램 이외에는 네트워크 단체법에서 C언어 특성을 이용한 효율적인 연결 리스트의 구현에 대한 연구가 Lustig에 의해 행해졌고[10], 내부점 선형계획법 프로그램의 구현에 있어 계층적 메모리 구조의 특성을 활용하기 위하여 캐쉬메모리(cache memory)의 크기를 고려하는 방안에 대한 연구가 Rothberg와 Gupta에 의해 수행되었다[13].

본 연구에서는 고속 단체법 프로그램의 구현을 위하여 컴퓨

* 본 연구는 한국과학재단의 목적기초연구과제(과제번호 95-0200-39-01-2)에 의해 지원되었음

** 서울대학교 산업공학과

*** 대진대학교 산업공학과

터 특성을 이용하는 여러 가지 기법들을 단체법에 도입하고 실험 결과를 비교 분석한다. 본 연구에서 주로 다루는 연구내용은 다음과 같다.

첫째, 컴퓨터 메모리 관리 특성에 의한 효율화 기법으로 간접 주소 사용을 줄이기 위한 기법, 메모리의 중복 참조의 제거, scatter/gather 방법[14]들을 단체법 프로그램에 적용하고 실험결과를 비교 · 분석한다. 또한 컴퓨터의 산술 연산 메커니즘을 이용하여 프로그램 수행 시간을 줄이는 기법으로 loop-unrolling 기법을 구현하여 그 효과를 분석한다.

둘째, 저수준 언어를 단체법 프로그램의 서브루틴에 구현하고 실험 결과를 비교 분석한다.

셋째, 대형 희소행렬을 보관하는 효율적인 자료구조 중 연결 리스트구조와 Gustavson자료 구조[6]를 비교 분석한다.

넷째, 대형 자료화일의 입력과정에서 사용될 수 있는 버퍼링기법에 대해 그 효율을 분석한다.

본 연구에서 수행하는 모든 실험은 SUN Ultra Sparc 170상에서 행해졌으며, 프로그램의 컴파일은 GNU gcc ver 2.1을 사용하였다. 그리고, 실험에서 사용되는 단체법 프로그램은 LPAKO ver 3.3f[3]이다.

2. 컴퓨터 구조와 메모리 관리 특성에 의한 효율화

단체법의 흐름

단체법은 다음과 같은 선형계획법을 푸는 해법으로,

$$Min \quad c^T x$$
$$s.t \quad Ax=b \quad A: m\times n,\ b: m\times 1,\ c: n\times 1$$
$$x\geq 0$$

기저행렬을 상하분해[5]하여 유지하고 평가전략으로 최소할인가법을 사용하는 단체법의 주요 흐름은 다음과 같다[1].

단계 1 : 초기기저의 상하분해
$B = LU$, L : 하삼각행렬, U : 상삼각행렬

단계 2 : 단체승수 π의 계산
$\pi = c_B^T B^{-1}$, c_B는 기저에 대응되는 목적함수 계수

단계 3 : 진입변수 x_p의 선택 및 최적판정
$p = argmin\{ \bar{c}_i : \bar{c}_i = c_i - \pi A_{\cdot i} \}$
만일 $\bar{c}_p \geq 0$이면 최적

단계 4 : 진입열 $A_{\cdot p}$의 수정 및 탈락변수의 선택
$\bar{A}_{\cdot p} = B^{-1} A_{\cdot p}$
$r = argmin\{ (B^{-1}b)_i / \bar{A}_{ip} \}$, r는 탈락변수의 기저에서의 지수

단계 5 : 상하분해 요소 및 해의 수정 후 단계 2로 간다.

메모리 참조 시간의 감소 방안

대량의 수치 계산을 주로 처리하는 컴퓨터 프로그램을 작성할 때 주의할 점 중의 하나가 가능한 기억공간의 참조회수를 줄이는 것이다. 프로그램의 수행시간은 수치연산을 수행하는 시간과 데이터를 메모리로부터 읽고 메모리에 쓰는 시간 즉, 데이터 이동에 소요되는 시간으로 구성된다. 대부분의 연구에서는 수치연산에 수행되는 시간을 줄이기 위한 알고리즘이나 방법 등을 주로 다루지만 주어진 방법의 실제 구현에서 데이터 이동에 소요되는 시간의 중요성에 대해서는 많은 연구가 없었다. 메모리 참조 시간을 줄이는 방안으로 다음과 같은 3가지 기법을 고려하였다.

- 직접주소에 의한 메모리 참조
- 메모리 참조의 중복 제거
- scatter/gather 방법

첫 번째 방법은 간접주소에 의한 메모리 참조를 직접주소에 의한 메모리 참조 방식으로 전환하는 것을 말하고 두 번째 메모리 참조의 중복 제거는 동일한 데이터를 사용하는 부분을 묶어 계산함으로써 메모리 참조의 중복을 피하는 방법을 말한다. 세 번째 scatter/gather 방법도 일종의 직접주소에 의한 메모리 참조 방법인데 첫 번째 방법과는 구현상의 차이가 있어 따로 고려하였다.

① 직접주소에 의한 메모리 참조

상하분해를 이용한 단체법 프로그램에서 간접주소에 의한 연산이 가장 많이 발생하는 부분은 단체승수와 수정된 진입열을 구하는 부분이다. 기저행렬이 $B = LU$로 상하분해되었을 때 전 · 후방 치환을 이용하여 다음과 같이 단체승수와 수정된 진입열를 구한다. 각각의 연산을 BTRAN, FTRAN이라 한다.

$$B^T \pi = U^T L^T \pi = c_B \quad \text{(BTRAN)}$$
$$B\bar{A}_{\cdot j} = LU\bar{A}_{\cdot j} = A_{\cdot j} \quad \text{(FTRAN)}$$

여기서 L과 U는 기저행렬 B를 상하분해할 때 나오는 하삼

각행렬과 상삼각행렬을 말한다. 상하분해의 실제 구현에서 가우스 소거법을 사용할 경우 L^{-1}와 U를 보관하는데 이 경우 BTRAN과 FTRAN는 다음과 같이 수행된다. 아래 BTRAN에서 v는 $(U^{-1})^T c_B$의 계산 결과를 임시로 저장하는 배열이다.

$U^T v=c_B,\ \pi=(L^{-1})^T v$ (BTRAN)
$U\bar{A}_{\cdot j}=L^{-1}A_{\cdot j}$ (FTRAN)

위와 같이 BTRAN과 FTRAN에서 열벡터 $A_{\cdot j}$에 L^{-1}를 곱하는 부분이 있는데 FTRAN 부분만을 간접주소를 사용하는 코드와 직접주소를 사용하는 C 코드로 나타내면 다음과 같다. 계산 결과는 ALBAR라는 배열에 저장된다.

```
ALBAR[] ←A.j;
for(i=LENL; i >= 1; i--)
  ALBAR[BRN[i]]+=BIN[i]*ALBAR[BCN[i]];
```

〈그림 1〉 $L^{-1}A_{\cdot j}$ 계산(간접주소)

```
ALBAR[] ←A.j;
for(i=LENL; i >= 1; i--){
  irn=BRN[i]; icn=BCN[i];
  ALBAR[irn]+=BIN[i]*ALBAR[icn];
}
```

〈그림 2〉 $L^{-1}A_{\cdot j}$ 계산(직접주소)

L^{-1}의 각 요소는 (irn, icn, val) 형태로 저장되는데 irn는 행위치, icn는 열위치, val는 실제값을 의미한다. 위의 그림에서 BRN[]는 행위치, BCN[]는 열위치, BIN[]는 실제값을 저장하는 배열이고 LENL는 L^{-1}의 비영인 요소의 개수이다. 〈그림 2〉은 미리 irn과 icn에 BRN[i]과 BCN[i]의 값을 저장한 다음 곱셈연산을 수행한다는 점에서 〈그림 1〉과 다르다. 즉, 〈그림 2〉는 irn과 icn를 사용하여 곱셈 연산시에 직접주소를 사용하는 반면 〈그림 1〉는 간접주소를 사용하는 방법을 취하고 있다.

일반적으로 간접주소 참조는 직접주소 참조에 비해 많은 시간이 걸리는 것으로 알려져 있다. 동일한 기능을 수행하는 위의 두 코드를 실제 구현하면 코드 2가 코드 1에 비해 약 5~20%정도 수행시간이 단축된다.

② 중복적인 메모리 참조의 제거

단체법은 일반적으로 문제 크기가 커질수록 반복회수가 많아져 전체 수행시간이 증가하게 된다. 단체법의 반복회수를 줄이는 방법 중의 하나가 steepest-edge 방법[7]인데 이 방법은 진입변수를 선택하는 방법을 새롭게 제시한다. Steepest-edge 방법은 대부분의 문제에서 단체법의 반복회수를 획기적으로 감소시키는 것으로 알려져 있으나 매 회 다음과 같은 추가적인 연산을 필요로 한다. 여기서 p는 진입열을 의미하고 e_p는 p번째 요소만 1이고 나머지 요소는 0인 m차원 열벡터이다.

$v=B^{-T}A_{\cdot p}$
$\sigma=B^{-T}e_p$

Steepest-edge방법은 반복회수의 감소로 인한 계산시간의 감소 효과가 위의 추가적인 계산으로 인한 연산 시간의 증가보다 클 경우에만 그 개선효과가 나타나게 된다. Steepest-edge 방법은 보통 문제 크기가 어느 정도이상인 경우에 개선효과가 나타나는 것으로 알려져 있다.

v, σ의 계산은 B^{-T}, 즉 L^{-1}와 U를 공통적으로 참조하는데 만약 v, σ를 순차적으로 계산하면 L^{-1}와 U를 각각 두 번 참조하게 된다. 그러나 v, σ를 병렬적으로 계산하면 L^{-1}와 U를 각각 한 번만 참조해도 된다. 즉, L^{-1}와 U를 한 번 참조하면서 v와 σ를 동시에 계산함으로써 기저역행렬 참조시간을 줄이는 것이다. L^{-1}를 곱하는 부분을 병렬적으로 계산하면 다음 그림과 같다. 물론 U를 사용하여 후방치환하는 부분도 병렬적으로 처리할 수 있다.

```
v[] ←Ā.p;
σ[] ←e.p;
.....
for(i=1; i <= LENL; i--) {
  irn=BRN[i]; icn=BCN[i];
  v[icn] += BIN[i]*v[irn];
  σ[icn] += BIN[i]* σ[irn];
}
```

〈그림 3〉 기저역행렬연산의 병렬화

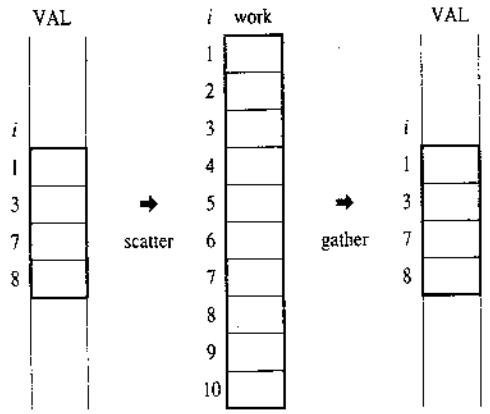
Steepest-edge 방법은 단체법의 국면 2에서 비기저변수의 할인가와 단체승수를 매 회 새로 계산하지 않고 이전 회의 값을 수정함으로써 할인가와 단체승수를 구하게 된다. 그러나 Wolfe가 제시한 국면 1 방법에서는 기저변수의 목적함수 계수가 매

회 달라져서 수정을 통해 단체승수를 구할 수 없다. 이 경우 매 회 BTRAN를 수행하여 단체승수를 구하게 되는데 이 때도 위의 v, σ를 구하는 루틴과 동일한 형태로 L^{-1}와 U를 참조하게 된다. 따라서 BTRAN과 v, σ의 계산 부분을 모두 병렬적으로 수행하면 국면 1에서의 L^{-1}와 U 참조 회수를 보다 줄일 수 있다.

③ scatter/gather 기법

단체법 프로그램은 대형 희소행렬로 이루어진 문제를 다룰 수 있도록 행렬의 비영요소만을 보관하고 계산 과정에서도 비영요소만을 고려하는 방법을 사용하고 있다[4]. 이 경우에 각 비영요소의 행렬 내에서의 위치를 같이 보관해야 한다.

scatter/gather 기법은 비교연산을 줄이기 위해 임시배열을 사용하여 계산에 관련된 배열 가운데 하나를 비용요소만 압축 보관된 상태에서 0를 포함하여 모든 요소를 보관하는 상태로 풀어 놓고 계산한 후 다시 압축하여 보관하는 방법이다. 압축된 배열을 임시배열에 풀어 넣는 과정을 scatter라 하고 계산이 완료된 다음 다시 원래 배열에 압축하여 저장하는 과정을 gather라고 한다. 다음의 그림은 scatter와 gather 과정을 설명하고 있다.



〈그림 4〉 scatter/gather

VAL은 계산하려는 데이터를 비영요소만 압축하여 보관하는 배열이고 work는 VAL에 있는 데이터를 풀어놓기 위해서 임시로 사용하는 배열이다. 위의 그림에서 VAL의 10의 요소 중에서 1, 3, 7, 8번째 요소가 비영요소이다.

단체법 프로그램에서 가우스 소거 과정은 선회열을 선회요소 아래의 열에 대해 차례로 더하는 연산으로 볼 수 있는데 이 부분에 gatter/scatter 기법을 사용하면 수행시간을 줄일 수 있다. 가우스 소거 과정은 선회연산의 연속인데 다음은 기저역행렬의 i_2번째 행을 선회행으로, i_1행을 소거대상행으로 하여 선회연산을 행하기 위한 부분 중 비영요소만으로 압축된 형태로 보관된 i_1번째 행을 임시배열 PIVWORK로 scatter하는 부분을 보여주고 있다.

```
/* Decompress row i1 into PIVWORK */
for (k=BROW[i2]; k!=-1; k=BRNEX[k]) {
    temp=BCN[k];
    PIVWORK[temp]=BIN[k];
}
```

〈그림 5〉 선회연산과정 중 적용된 scatter

Loop-unrolling 기법에 의한 효율화

Loop-unrolling은 컴퓨터의 특성을 활용하는 전형적인 기법이다. 다음과 같은 코드를 살펴보자.

```
mult=2.234;
for(i=0; i < 100; i++) {
  ind=address[i]; val[ind]*=3;
}
```

〈그림 6〉 일반적인 for-loop문

```
mult=2.234
for(i=0; i<100; )
  ind=address[i]; val[ind]*=3; i+=1;
  ind=address[i]; val[ind]*=3; i+=1;
  ind=address[i]; val[ind]*=3; i+=1;
  ind=address[i]; val[ind]*=3; i+=1;
}
```

〈그림 7〉 loop-unrolling의 예

위의 〈그림 7〉은 〈그림 6〉 코드의 for-loop내 부분을 4번 반복한 것이다. 수행시간면에서 〈그림 6〉과 〈그림 7〉이 동일할 것으로 예상할 수 있으나 실제로는 〈그림 7〉이 〈그림 6〉 코

드보다 빠르다. 〈그림 7〉은 for-loop를 수행하는 동안 'i〈100' 인가를 판단하는 비교연산이 〈그림 6〉의 경우에 비해 (1/4)배이다. 또한 대부분의 수치연산를 다루는 컴퓨터가 파이프라이닝(pipelining)효과를 이용하는데 파이프라이닝효과를 최대한 이용하기 위해서는 가능한 한 비교연산를 포함하는 분기연산(branching operation)이 수치연산속에 포함되지 않도록 해야 한다[12]. 이런 면에서 〈그림 7〉은 비교연산 없이 동일한 형태의 수치연산을 4번 반복함으로써 파이프라이닝효과를 극대화할 수 있다.

상하분해를 사용하는 단체법 프로그램에서는 단체승수와 수정된 진입열을 구하기 위해 전·후방 치환을 하게 되는데 이때 L, U의 비영요소를 차례대로 곱하는 연산을 수행한다. 여기에 loop-unrolling 기법을 도입할 수 있는데 실험에 의하면 약 5-10%정도의 속도 개선을 가져오는 것으로 나타났다.

실험결과

메모리 참조시간의 감소방안으로 직접주소에 의한 메모리 참조, 메모리 참조의 중복 제거, scatter/gather 방법, loop-unrolling 기법 등을 구현하고 Netlib. 문제[9]로 실험한 결과 평균적으로 11.5%정도의 계산시간 단축효과가 있었다. 〈표 1〉은 그 실험결과이다.

〈표 1〉 메모리 참조시간의 감소, loop-unrolling기법의 구현 실험결과

문제이름	문제크기			계산시간		개선율 (%)
	행개수	열개수	비영요소수	개선 前	개선 後	
truss	1001	8806	36642	77.86	70.90	8.9
degen3	1504	1818	26230	51.59	44.95	12.9
d2q06c	2172	5167	35674	181.25	166.12	8.3
bnl2	2325	3489	16124	34.08	29.73	12.8
stocfor3	16676	15695	74004	780.03	697.65	10.6
fit2d	26	10500	138018	195.37	165.92	15.1
perold	626	1376	6026	15.18	13.67	10.0
pilot	1442	3652	43220	575.55	490.75	15.0
wood1p	245	2594	70216	23.17	19.21	17.1
greenbea	2393	5405	31499	86.72	79.17	8.7
pilotnov	976	2172	13129	19.91	17.10	14.1
dfl001	6071	12230	35632	19913.62	17303.80	13.1
80bau3b	2263	9799	29063	73.13	65.52	10.4
pilot	1442	3652	43220	510.53	493.33	3.4
평균						11.5

3. 저수준 언어에 의한 효율화

어셈블리 언어의 도입 필요성

저수준 언어와의 혼합 프로그래밍으로 구현하는 프로그램의 예는 문서 편집기(text editor)에서 흔히 발견할 수 있다. 입력된 문자에 대한 비교 연산, 메모리 복사 등이 빈번한 문서 편집기는 사용자가 수행 속도에 매우 민감한 특성을 가지고 있어서 비교 연산 부분이나 메모리를 복사하는 부분 등을 저수준 언어로 구현하고 나머지 부분에 대해서는 고급 언어를 사용하는 혼합 프로그램을 사용하여 수행 속도를 개선한다.

고속 선형계획법 프로그램을 개발하는 데에 있어서 저수준 언어의 응용 사례는 공개된 예가 없으나, 컴퓨터의 특성을 최대한 활용하기 위해서는 저수준 언어를 사용하는 것이 유리하다. 저수준 언어인 어셈블리 언어는 컴퓨터의 작동을 기계어 명령어 수준으로 제어할 수 있으므로 불필요한 연산이나 메모리 참조를 줄이고 파이프라이닝의 효과 등을 잘 이용하면 고급언어로 작성한 프로그램에 비해 보다 효율적인 프로그램을 만들 수 있다[12]. 그러나 어셈블리 언어는 코딩 자체가 쉽지 않고 컴퓨터의 구조를 명확하게 이해하고 있을 때에만 그 효과를 살릴 수 있으므로 단체법 프로그램 전체를 어셈블리 언어로 구현하는 것이 현실적으로 어렵다. 따라서 단체법 프로그램에서 특히 시간이 많이 걸리거나 또는 빈번히 사용되는 핵심부분만을 어셈블리 언어로 구현하는 것이 바람직하다.

실험 결과

대부분의 단체법 프로그램에서는 MPS형태의 자료화일을 입력으로 받는다. MPS형태의 자료화일을 입력할 때 비교적 시간이 많이 소요되는 부분은 문자열을 부동소수점 숫자로 변환하는 부분이다. 본 연구에서는 그 부분을 저수준언어인 어셈블리 언어를 사용하여 구현하여 실험해 보았다. 그 결과 C 컴파일러가 제공하는 최적화 옵션을 사용하지 않았을 때 어셈블리로 작성된 코드가 C로 작성된 코드보다 평균 15%정도 빠르다는 것을 확인할 수 있었다. 〈표 2〉는 Netlib. 문제에 대한 실험결과이다.

〈표 2〉 어셈블리어 구현 실험결과

문제이름	문제크기			입력시간		개선율
	행개수	열개수	비영요소수	C 언어	어셈블리어	(%)
perold	626	1376	6026	0.19	0.15	21.1
wood1p	245	2594	70216	1.50	1.43	4.7
stocfor3	16676	15695	74004	2.47	2.11	14.6
greenbea	2393	5405	31499	0.83	0.69	16.9
pilotnov	976	2172	13129	0.35	0.24	31.4
dfl001	6071	12230	35632	1.04	0.89	14.4
d2q06c	2172	5167	35674	0.67	0.54	19.4
truss	1001	8806	36642	0.71	0.66	7.0
80bau3b	2263	9799	29063	0.69	0.61	11.6
pilot	1442	3652	43220	0.82	0.74	9.8
평균						15.1

단체법 프로그램의 전체 수행시간 중에서 자료입력부분이 차지하는 비중은 상당히 작지만 저수준언어의 효용성을 보인다는 측면에서 위 실험은 의미를 가진다고 할 수 있다.

4. 자료구조

대형 희소행렬의 보관

행렬을 컴퓨터 메모리에 저장할 때 비영요소의 수가 많으면 모든 요소를 저장하는 것이 가능하지만 행렬이 크고 또 비영요소의 수가 매우 적은 희소행렬의 경우는 비영요소만 보관하는 방법이 기억장소 및 계산 시간 측면에서 효율적이다. 행렬의 비영요소만을 보관하는 방법은 여러 가지가 있는데 이 중 연결 리스트(Linked list)로 보관하는 방법과 Gustavson 구조로 보관하는 방법은 다음과 같다.

연결 리스트 구조

연결 리스트 구조는 비영요소들을 행과 열의 링크(Link)로 연결시켜 놓은 구조이다.

행렬이 수정되면 요소들의 값이 변하게 되므로 비영요소의 삽입, 삭제가 필요하고 또 같은 행 · 열에 있는 비영요소를 검색할 필요가 있기 때문에 비영요소의 값, 행 · 열 번호, 다음 비영요소의 주소, 각 행 · 열의 시작 주소를 보관하는 배열 등이 필요하다[2].

예제 행렬 A를 연결 리스트로 표현하면 〈그림 8〉과 같다.

$$A = \begin{bmatrix} 0 & 1 & 2 & 4 \\ 13 & 0 & 0 & 15 \\ 0 & 0 & 10 & 0 \\ 9 & 0 & 0 & 3 \end{bmatrix}$$

배열 주소	0	1	2	3	4	5	6	7	8	9	10
비영요소	9	1	*	10	3	4	2	15	*	13	*
행번호	4	1	0	3	4	1	1	2	0	2	0
열번호	1	2	*	3	4	4	3	4	*	1	*
행방향 링크	4	5	*	#	#	#	1	9	*	#	*
열방향 링크	9	#	*	6	7	#	#	5	*	#	*

행 시작주소	6	7	3	0
열 시작주소	0	1	3	4

* : 임의의 값, # : link의 끝임을 알려 주는 특정한 값

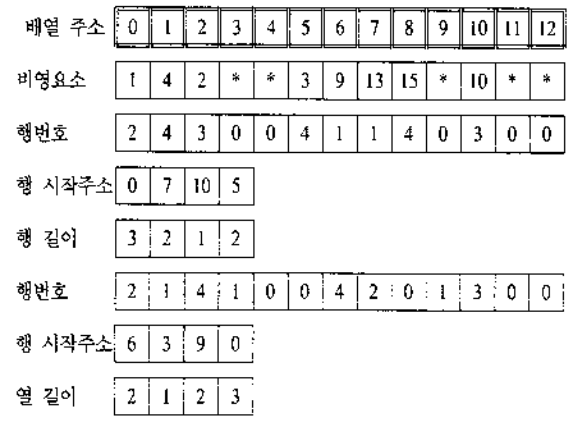
〈그림 8〉 연결리스트 자료구조

연결리스트 자료구조는 삽입, 삭제가 쉬우나 동일한 열이나 행에 속하는 요소들이 인접하게 저장되는 것이 보장되지 못하므로 열이나 행을 따라 행해지는 메모리 참조에서 페이지 부재(page fault)현상[14]이 자주 일어나는 단점이 있다.

Gustavson 구조

Gustavson 자료구조는 행렬의 비영요소만을 행별로 보관한다. 즉, 비영요소를 보관하는 배열과 각 비영요소의 열 번호를 보관하는 배열이 있고 각 행의 시작번지와 비영요소 갯수를 보관하는 배열이 있다. 또 각 열의 비영요소의 행번호를 알 필요가 있기 때문에 각 열의 비영요소의 행번호를 보관하는 배열이 있고, 이 배열에서 각 열의 시작번지와 비영요소 갯수를 알려주는 배열이 있다.

위의 예제 행렬 A를 Gustavson 구조로 나타내면 〈그림 9〉와 같다.



〈그림 9〉 Gustavson 자료구조

Gustavson 구조에서는 같은 행(또는 같은 열)에 있는 비영요소는 연속해서 나타나야 한다. 위의 예에서 4행에 새로운 비영요소가 생기는 경우에는 4행의 마지막 요소 뒤에 여유공간이 없으므로 그 행 전체를 맨 뒤로 옮긴 다음 새로운 비영요소를 넣는다[6].

비영요소가 많이 생겨서 더 이상의 여유공간이 없게 되면 공간을 제거해주는 압축(Compression)과정을 통하여 기억용량을 효율적으로 사용한다. 그러나 이 자료구조는 삽입, 삭제가 많이 일어나면 기억공간을 재할당 받아야 하고 삭제가 많이 일어나면 기억공간이 낭비되는 단점이 있다.

Reid에 의하면, 선형계획법 프로그램인 LA05의 경우, 초기(LA03)에는 기저역행렬(정확히는 상삼각행렬)의 보관에 연결리스트를 사용하였으나, 당시로서는 정수로 링크를 나타낼 경우 약 32000개 이상의 비영요소를 사용할 수 없고, 페이징(Paging)시의 문제점들 때문에 Gustavson 자료구조로 전환하였다고 한다[11]. 그러나 최근의 컴퓨터들에서는 4바이트 정수를 사용하고 페이징 기법에서도 많은 개선이 이루어졌기 때문에, 연결 리스트를 좋은 대안으로 생각할 수 있어 이들에 대한 새로운 비교 검토가 필요하다.

실험 결과

Bartels-Golub[5]과 Reid 수정 방법[11] 각각에 대해서 행·열 연결 리스트 방법과 Gustavson 자료구조를 비교하는 실험을 수행한 결과 다음 표에 나타난 바와 같이 서로 비슷한 성능을 보인다. 〈표 3〉은 행·열 연결 리스트의 수행속도를 기준(100)으로 Gustavson자료구조의 수행속도를 나타낸 것이다.

〈표 3〉 연결 리스트와 Gustavson 자료구조의 수행속도 비교

	행·열 연결 리스트	Gustavson 자료구조
Bartels-Golub 방법	100	91.2
Reid 방법	100	101.7

5. 대형자료화일의 입력

MPS 자료화일 입력방법의 효율화

MPS형 자료는 선형계획법 문제를 나타내는 자료형식의 일종으로, 대부분의 상용 패키지들에서 표준적으로 사용되고 있다. 이러한 MPS 자료화일은 그 크기가 상당히 크기 때문에 자료입력시간이 비교적 많이 소요되고, 그에 따른 효율적인 자료입력 방법이 요구된다. MPS 자료입력 시간의 단축을 위한 기법들은 자료검색방법의 효율화 측면과 컴퓨터 특성을 이용한 효율화 측면으로 나누어 볼 수 있다. 전자의 경우에는 해싱기법의 사용, 그에 따른 다양한 collision 해결방안의 적용, 정렬방법의 효율화 등이 있으며, 후자의 경우는 자료 입력시 버퍼링 기법의 사용 등이 있다. 본 연구에서는 후자의 경우를 살펴 보겠다.

버퍼링(buffering) 기법

가장 기본적으로 사용할 수 있는 자료화일의 입력은 줄단위 입력이다. 즉, 디스크상에 있는 화일에 접근하여 한 줄을 읽어들여 처리하는 과정을 화일의 끝까지 반복하는 방법이다. 그러나 그 방법은 계속해서 디스크 접근이 필요하게 되어 속도저하의 원인이 될 수 있다. 지나친 디스크 접근을 방지하기 위해 사용되는 기법이 버퍼링 기법이다. 즉, 한번의 디스크 접근을 통해 많은 양의 자료화일의 내용을 버퍼에 읽어들여서 사용하면 디스크 접근의 수를 줄일 수 있다.

실험 결과

본 연구에서는 자료 입력시 사용되는 버퍼의 크기에 따라 입력시간이 어떻게 변하는지 실험을 통해 확인하였으며 그 결

과 버퍼의 크기가 클수록 입력시간이 단축된다는 것을 알 수 있다.

〈표 4〉 버퍼의 크기에 따른 자료입력시간의 비교

문제 이름	문제 크기			입력시간(초)		개선율 (%)
	행개수	열개수	비영요소수	버퍼 5Mbytes	버퍼 1Mbytes	
woodlp	245	2594	70216	1.50	2.31	35.1
d2q06c	2172	5167	35674	0.67	0.92	27.2
truss	1001	8806	36642	0.71	1.11	36.0
80bau3b	2263	9799	29063	0.69	0.97	28.9
pilot	1442	3652	43220	0.82	1.27	35.4
perold	626	1376	6026	0.19	0.23	17.4
stocfor3	16676	15695	74004	2.47	2.86	13.6
greenbea	2393	5405	31499	0.83	1.05	21.0
pilotnov	976	2172	13129	0.35	0.49	28.6
dfl001	6071	12230	35632	1.04	1.48	29.7
평균						27.3

4. 결론

본 연구에서는 단체법을 구현할 때 적용될 수 있는 컴퓨터 특성을 이용한 기법들을 검토하고 그 효율성을 실험을 통해 검증하였다.

컴퓨터 메모리 관리 특성을 이용한 효율화 기법으로 간접 주소 사용을 줄이기 위한 기법, 메모리의 중복 참조의 제거, scatter/gather 방법들을 단체법 프로그램에 적용하여 그 효과를 검증하였다. 또한 저수준 언어를 단체법 프로그램의 중요 서브루틴에 구현하고 그 효과를 살펴보았고, 또한 컴퓨터의 산술 연산 메커니즘을 이용하여 프로그램 수행 시간을 줄이는 기법으로 loop-unrolling 기법을 구현하여 그 효과를 분석하였다. 그리고 대형 희소행렬을 보관하는 효율적인 자료구조 중 연결 리스트구조와 Gustavson자료 구조를 비교 분석하여 다양한 상황에서 두가지 방법이 모두 경쟁력이 있다는 것을 확인하였다. 마지막으로 대형 자료화일의 입력과정에서 사용될 수 있는 버퍼링 기법에 대해 그 효율을 보였다.

[참 고 문 헌]

[1] 박순달, 선형계획법(3정판), 민영사, 1992

[2] 박순달, OR(경영과학), 민영사, 1991

[3] 박순달, LPAKO ver 3.3f 사용자 안내서, 서울대학교 체계분석실, 1997

[4] 김우제, 강완모, 김민정, 박순달, "일반한계 선형계획법에서 비영요소만 보관하는 자료구조와 평가전략의 효율성에 관한 연구", 전산활용연구, 제6권 제1호, pp. 55-66, 1993

[5] Bartels. R. H. and G. H. Golub., "The Simplex method of linear programming using LU decomposition", Communication of ACM, 12, pp. 266-268, 1969

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

[7] Forrest, J. J. and Goldfarb, D., "Steepest-edge simplex algorithms for linear programming", Mathematical Programming, 57, pp. 341-374, 1992

[8] Hwang, K., Advanced Computer Architecture, McGraw-Hill, 1993

[9] Gay, D. M., "Electronic mail distribution of linear programming test problems", Mathematical Programming Society Committee on Algorithms Newsletter, 13, pp.10-12. 1985

[10] Lustig, I.J., "The influence of computer language on computational comparisons: an example from network optimization", ORSA Journal on Computing, 2, pp. 152-161, 1990

[11] Reid J. K., "A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases", Computer Science and Systems Devision, A.E.R.E., Harwell, CSS20, pp. 1-23, 1975

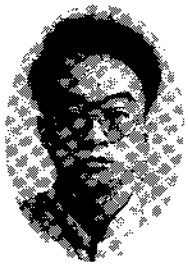
[12] Richard D. Paul, SPARC Architecture, Assembly Language Programming, And C, Prentice Hall, 1994

[13] Rothberg, Edward and Anoop Gupta, "Efficient sparse matrix factorization on high-performance workstations - exploiting the memory heirarchy", Technical Report, Department of Computer Science, Standford Univ.. Aug. 21, 1990

[14] Uresh Vahalia, UNIX Internals the New Frontiers, Prentice Hall, 1996

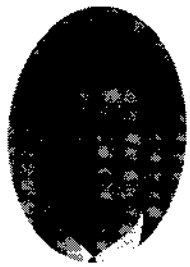
박찬규

현재 서울대학교 산업공학과 박사과정에 재학중이다. 서울대학교 산업공학과에서 공학사 및 공학석사 학위를 취득하였다. 주요 관심분야는 선형계획법, 네트워크 프로그래밍 및 GIS 활용분야이다.



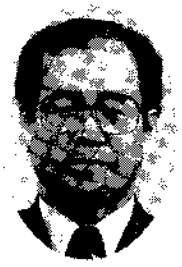
임성묵

현재 서울대학교 산업공학과 박사과정에 재학중이다. 서울대학교 산업공학과에서 공학사 및 공학석사 학위를 취득하였다. 주요 관심분야는 컴퓨터 활용분야, 선형계획법 및 조합 최적화 분야이다.



김우제

현재 대전대학교 산업공학과 조교수로 재직중이다. 서울대 산업공학과에서 공학사(1986), 공학석사(1988) 그리고 공학박사 학위를 취득하였다. 동양경제연구소에서 연구원으로 근무하기도 하였다. 주요 관심분야는 컴퓨터응용 OR 분야, 물류관리분야이다.



박순달

현재 서울대학교 산업공학과 교수로 재직중이다. 미국 University of Cincinati에서 이학박사 학위를 취득하였고, 독일 Ruhr University에서 연구원으로 근무하였다. 주요 관심분야는 Deterministic OR과 컴퓨터 활용분야이다.

97년 7월 최초접수, 98년 4월 최종수정