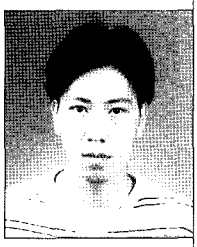
세상에는 정말 빠른 Solver도 있구나

Introduction to Recent Open Source Sparse Solver



조 정 래*

*한국건설기술연구원 구조연구부 선임연구원

1. 들어가면서

안녕하세요. 저는 한국건설기술연구원 구조연구부에 근무하는 조정래입니다. 앞으로 유한요소 프로그램 개발과정의 경험을 수기 형식을 빌어 풀어 나가고자 합니다. 앞으로 연재될 목차는 다음과 같습니다.

1. 세상에는 정말 빠른 Solver도 있구나.
2. 혼합 컴파일과 스크립트 언어
3. Visualization
4. 객체지향프로그래밍

이번 연재에서는 유한요소법 등에 사용할 수 있는 최신 sparse solver들의 필요성을 설명하고, 그 중 대표적인 opensource 라이브러리인 CHOLMOD와 UMFPACK을 Windows 환경에서 인스톨하는 방법에 대해 설명하겠습니다.

저는 2004년부터 고속전철용 프로그램을 개발하고 있습니다. 언어는 C/C++를 이용하고 있고, 객체지향기법(object-oriented technique)을 도입하여 설계된 것이죠. 객체지향기법은 연재 마지막에 잠시 소개하도록 하겠습니다. 일단은 복잡한 시스템을 표현하는 데 좋은 프로그래밍 기법 정도로 정리하기로 하겠습니다. 최근 프로그램 개발이 거의 완료되었고, 실제 예제를 한번 돌려봤습니다. 해석할 교량은 PSC Box 거더 형식의 2@40m 교량이고, 요소는 4절점 쉘, 8절점 솔리드, 2절점 보요소 등이 사용되었습니다(그림 1, 2 참조). 자유도는 약 7만개 정도이고, 약 3000 step 정도를 필요로 하는 동적 이동하중 해석이었습니다. 해석은 순조롭게 되었지만 시간이 문제더군요. 10시간 정도 걸렸습니다. 참고로 제 컴퓨터는 Pentium 4에 L2 Cache가 1M입니다. 그리고 Kx = b와 같은 선형시스템을 풀기위해 Bathe나 Hughes 책에 소개되어 있는 전통적인 skyline solver를 사용했습니다. 그리고 자유도 번호는 Cuthill-McKee 알고리즘 또는 Reverse Cuthill-McKee 알고리즘으로 최적화하였습니다. 앞으로 이런 해석을 열차 속도별로, 경계조건별로 수십 가지를 수행해야 합니다. 또한, 열차-교량 상호작용을 고려한 해석을 수행해야 하는데, 이 해석은 이동하중 해석에 비해 2~3배는 시간이 더 걸릴 것으로 예상됩니다. 예를 들어 100가지 해석 조건을 이동하중 해석으로 해석한다고 보면, 100×10시간=1000시간=약 41일 정도 소요됩니다. 한숨만 나오더군요.

보다 빠른 해석 시간을 얻기 위해 인터넷에서 희소행렬(sparse matrix)을 푸는 루틴을 찾기 시작했습니다. 검색해 보니 몇몇 opensource 라이브러리가 있더군요. 제가 직접 사용해 본 것은 TAUCS(www.tau.ac.il/~stoledo/taucs/)와 University of Florida에서 개발한 CHOLMOD, UMFPACK (www.cise.ufl.edu/research/sparse/) 등 입니다. 결론부터 말해서 얼마나 빨라지는지 한번 확인해 볼까요? 표 1은 Kx

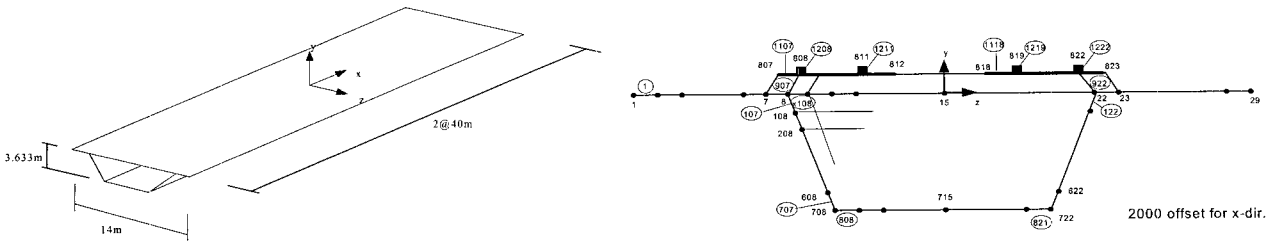


그림 1 2@40 PSC Box 거더교의 모델링

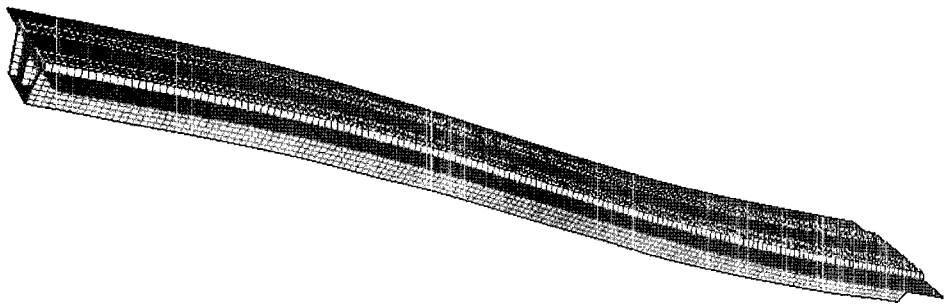


그림 2 2@40 PSC Box 거더교의 모델링 - 1차 모드 형상

표 1 Solver의 성능 비교(CPU wall time, P4 3.2GHz, L2 Cache 1M)

Solver 종류	대상행렬	BLAS	CPU wall time	
			Factorization	Backsubsitution
skyline	대칭행렬	-	82 sec	6 sec
TAUCS	대칭행렬	ATLAS	5 sec	5 sec
CHOLMOD	대칭행렬	ATLAS	5 sec	0.16 sec
UMFPACK	비대칭행렬	ATLAS	12 sec	0.18 sec

= b 행렬을 푸는 데 걸리는 시간을 각 solver별로 비교한 것입니다. skyline solver는 앞에서 언급한 대로 제가 직접 Bathe 책에 있는 것을 코딩한 것입니다. 그리고 factorization은 행렬 K를 LU 나 LDL^T 등으로 분해는 것을 의미하고, backsubstitution은 분해된 행렬에서 x 벡터를 구하는 과정입니다. 직접 구현한 skyline solver와 제일 빠른 속도를 나타낸 CHOLMOD와 비교할 경우 factorization에 걸리는 시간은 82초에서 5초로, backsubstitution은 6초에서 0.16초로 줄었습니다. 정말 빠르더군요. CHOLMOD와 TAUCS는 대칭(symmetric)행렬, UMFPACK은 비대칭(unsymmetric)행렬에 대해 사용할 수 있습니다. UMFPACK은 비대칭행렬에 대한 solver이기 때문에 시간이 좀 더 걸렸을 뿐입니다. TAUCS는 backsubstitution할 때 시간이 오래 걸렸는데, 아마도 이 부분 루틴이 그렇게 우수하지는 않는 모양입니다. 전체적으로 10시간이 걸리는 해석 시간은 어떻게 됐을까요? 아쉽게도 이 시간은 4시간 정도로 줄었습니다. 그 이유는 풀고자 하는 문제가 선형이기 때문에 factorization을 한번만 하고 요소단위에서 상태를 업데이트하는데 아직 코드가 최적화되지 못했기 때문입니다.

선형 시스템(linear system of equation : Ax = b)을 푸는 방법은 직접법(direct method)와 반복법(iteration method)이 있습니다. 푸는 방법에 따라 direct solver 또는 iterative solver라고 합니다. 위에서 skyline이나 TAUCS, CHOLMOD, UMFPACK 등은 모두 direct solver입니다. Direct solver는 대부분 A 행렬이 symmetric positive definite 행렬인 경우 LDL^T decomposition법을, 그 외의 경우에는 LU decomposition법을 이용해 행렬을 풀게 됩니다. Ax = b를 구하는 과정은 먼저 수식번호를 최적화하고(이를 symbolic analysis라 합니다), A를 LU 또는 LDL^T로 분해하고(decomposition), 마지막으로 해 x를 구하는 과정(backsubstitution)으로 구성됩니다. 예를 들어 제가 직접 코딩한 skyline solver는 symbolic analysis로 그래프 이론에 근거한 Cuthill-McKee 알고리즘과 Reverse Cuthill-McKee 알고리즘을 사용하고, LDL^T decomposition법을 사용합니다. 물론 TAUCS, CHOLMOD, UMFPACK 등도 고유의 symbolic analysis를 수행하고, TAUCS, CHOLMOD는 LDL^T decomposition 법을, UMFPACK는 LU decomposition을 사용합니다.

표 1에서 BLAS라고 하는 것은 Basic Linear Algebra

Subprogram의 줄임말로 행렬과 벡터 연산을 하는 수학함수 집합을 의미하는데, 보통 개별 컴퓨터에 맞도록 최적화된 라이브러리입니다. 여기에서는 matlab에서 사용되고 있는 ATLAS(http://math-atlas.sourceforge.net)를 사용하였습니다. 참고로 TAUCS는 Mathematica에, CHOLMOD와 UMFPACK은 Matlab에 적용되었습니다.

2. 인스톨하기

이제 앞에서 소개한 CHOLMOD와 UMFPACK을 Windows 환경의 컴파일러(여기서는 Microsoft Visual C/C++)에서 사용할 수 있도록 인스톨하는 방법을 소개하겠습니다. TAUCS는 여기서 다루지 않겠습니다. Opensource이기 때문에 인스톨하기가 만만치 않습니다. 왜냐하면 컴파일 과정을 Linux환경에서 해야 하기 때문입니다. 첫 번째 해야 할 일은 Windows에서 Linux환경을 사용할 수 있게 하는 CYGWIN환경을 구축해야 합니다. 그 다음으로 ATLAS와 LAPACK을 인스톨하고, METIS라는 것도 인스톨해 주어야 합니다. 마지막으로 CHOLMOD와 UMFPACK을 인스톨하는 순서로 진행하게 됩니다. 모든 인스톨 과정이 끝나면 생성된 라이브러리를 Windows의 대표적인 컴파일러인 Visual C++같은 툴로 링크해서 사용하면 됩니다. 과정이 꽤 복잡하기 때문에 차근차근 설명하기로 하겠습니다.

2.1 CYGWIN 환경 구축

CYGWIN은 Windows에서 Linux를 사용할 수 있도록 에뮬레이션하는 프로그램입니다. 먼저 www.cygwin.com에서 화면 중앙에 있는 Install or update now라는 것을 선택하면 그림 3과 같이 setup 화면이 뜨게 됩니다. 그 다음 부터는 잘 읽어 보고 인스톨하면 됩니다.

인스톨이 끝나면 Cygwin Bash Shell을 실행 시킨 후 명령행에서 gcc -v를 입력하십시오. 그림 4와 같은 결과가 나오면 Linux 환경에서 컴파일 할 환경이 완성된 것입니다.

GCC라고 하는 것은 GNU Compiler Collection의 줄임

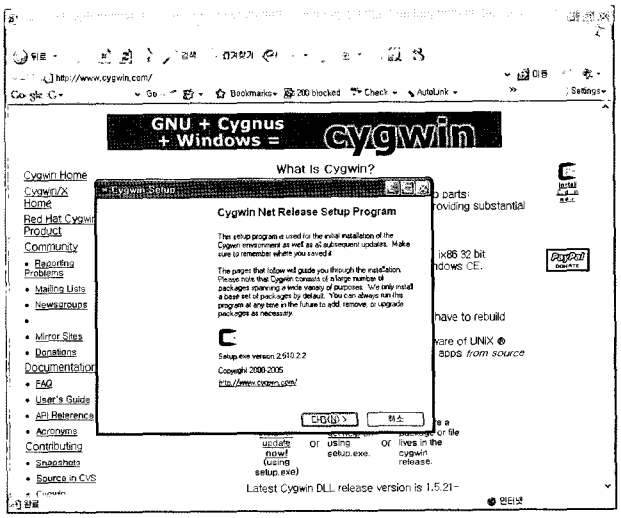


그림 3 CYGWIN

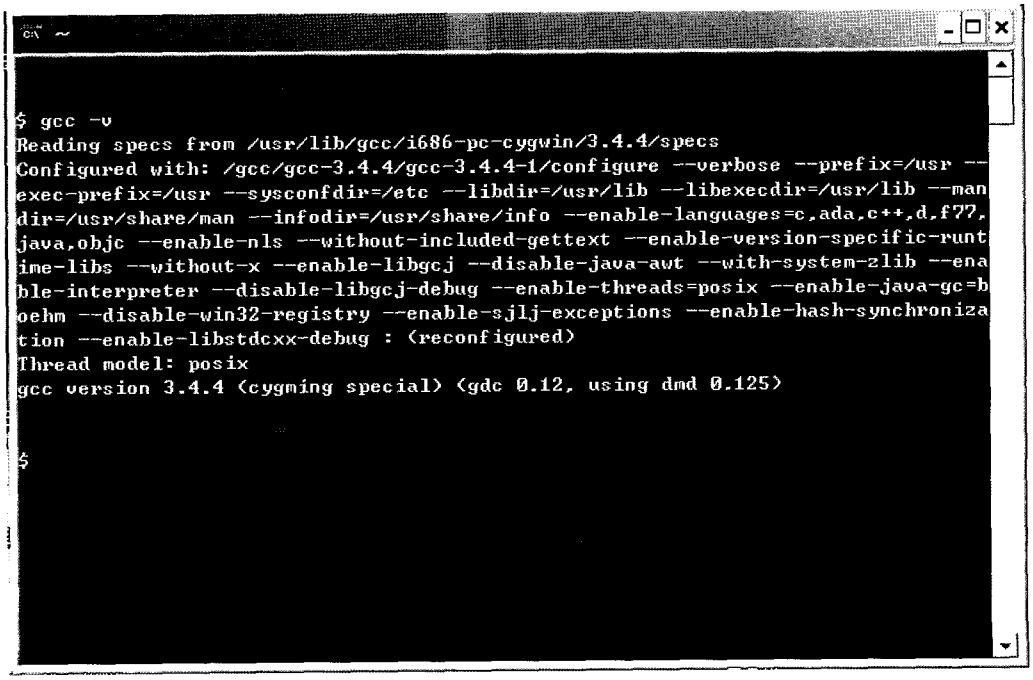


그림 4 Cygwin의 Bash Shell 실행 전경

말로 C, C++, Fortran 등의 여러 언어를 컴파일 할 수 있도록 GNU에서 개발한 컴파일러 툴 집합을 의미합니다. 예를 들어 화면에 hello, world라는 문자열을 출력하는 프로그램의 경우 C 언어 소스 파일은 다음과 같습니다.

```
/* hello.c */
#include <stdio.h>

int main()
{
  printf("hello, world\n");
}
```

컴파일하고 실행하는 것은 다음과 같습니다.

```
$ gcc -o hello.exe hello.c
$ ./hello.exe
```

위에서 ./는 현재 디렉토리를 의미합니다. Windows의 명령행과 달리 리눅스에서는 이와 같이 지정해 주어야 합니다.

2.2 ATLAS+LAPACK 인스톨

ATLAS는 대표적인 opensource BLAS입니다. 성능 역시 상용 BLAS에 못지않다고 알려져 있습니다. 여기에 소개된 방법은 www.scipy.org에 소개된 내용을 요약한 것으로 CYGWIN을 이용하는 것을 기준으로 합니다. 리스트 1에 나타낸 순서로 컴파일하면 됩니다. 컴파일 시간은 1~2시간 정도 걸리는데 그 이유는 컴파일하는 컴퓨터 하드웨어에 맞도록 코드를 최적화하기 때문입니다.

리스트 1의 과정을 통해 생성되는 라이브러리 파일 중에 libatlas.a가 가장 기본이 되는 ATLAS 라이브러리입니다. libcblas.a와 libf77blas.a는 C와 Fortran에서 사용할 수 있도록 libatlas.a의 루틴에 대한 인터페이스 정보를 갖는 라이브러리(각각 cblas, f77blas라고 보통 부름)입니다. liblapack.a는 ATLAS가 최적화한 LAPACK 루틴(LAPACK은 linpack의 확장판)을 가지고 있습니다. LAPACK 전체가 아닌 일부 루틴만을 가지고 있습니다. 전체 LAPACK에 대한 라이브러리는 구성하는 것은 리스트 2를 따라서 하면 됩니다. 참고로 리눅스에서는 라이브러리의 이름은 lib*.a 형태로 사용하게

리스트 1 Installing ATLAS with CYGWIN-GCC

1. ATLAS(http://math-atlas.sourceforge.net/)에서 최신 ATLAS 소스를 download하여 적당한 곳에 푼다(/ATLAS 디렉토리). 현재의 최신버전은 atlas3.7.11이고, 여기에서도 이것을 사용하였다.
2. /ATLAS/CONFIG/probe_SSE3.c의 77 line을 다음과 같이 고쳐 SSE3를 disable한다.
   /* if (testv3[0] != 3.0 || testv3[1] != 7.0) */
3. CYGWIN command prompt에서 Atlas 루트 디렉토리로 이동하여(예, $ cd /cygdrive/d/ATLAS) make를 실행한다.
4. make를 실행한 뒤, default 값들을 선택하면 된다. 다만, 여기에서 L2Cache size는 정확한 값을 주는 것이 좋다. 자기 컴퓨터의 CPU정보는 CPU-Z같은 프로그램을 이용하면 된다.
5. 실행이 끝날 때 표시되는 것처럼 make install arch= YOUR_ARCHITECTURE를 실행하고 기다린다. (예를 들어 $make install arch=WinNT_P4ESSE2). 컴파일이 실패할 수 있으므로 다른 작업을 하지 않는 것이 좋다.
6. make sanity_test arch=YOUR_ARCHTECTURE를 실행하여 테스트를 수행한다. 이때 [sanity_test] Error 1 (ignored) 라고 나오면 성공적으로 컴파일 된 것이다.
7. 생성된 library파일은 /ATLAS/bin/ARCH/에 있으며, liblapack.a, libcblas.a, libf77blas.a, libatlas.a 등이다.

리스트 2 Installing LAPACK with CYGWIN-GCC

1. Lapack 소스(http://www.netlib.org/lapack/lapack.tgz)를 다운받아 풀고, http://www.netlib.org/lapack-dev/에서 최신 patch를 다운받아 동일한 폴더에 풀어 놓는다(LAPACK 디렉토리).
2. /LAPACK/INSTALL/make.inc.LINUX 파일을 LAPACK/make.inc로 복사한다.
3. /LAPACK/Makefile의 마지막 라인에 .PHONY: install testing timing을 추가한다.
4. make install lib을 실행한 후 컴파일이 끝날 때까지 기다린다(The timing error in the beginning is without meaning.).
5. make testing timing을 실행한다. 이 부분은 꼭 할 필요는 없다.
6. 생성된 라이브러리 파일은 Lapack_LINUX.a이다.
7. 이제 ATLAS에서 생성된 최적화된 lapack을 포함하는 full LAPACK 라이브러리를 구성하면 된다. ATLAS 라이브러리 파일들(liblapack.a, libcblas.a, libf77blas.a, libatlas.a)과 Lapack_LINUX.a를 적당한디렉토리(/LIB)로 복사해서 옮긴다.

```
$ ar x liblapack.a
$ ar r lapack_LINUX.a *.o
$ rm *.o
$ mv lapack_LINUX.a liblapack.a
```

8. 이제 남은 libcblas.a, libf77blas.a, liblapack.a, libatlas.a가 최적화된 static CBLAS, BLAS, (complete) LAPACK, low-level ATLAS library이다.

됩니다.

위의 ATLAS+LAPACK 라이브러리는 확장자만 a에서 lib으로 바꾸면 MSVC(Microsoft Visual C/C++)에서 바로 사용할 수 있습니다. MSVC에서 사용할 때의 주의할 점은 Debug버전에서 컴파일 할 때 project settings의 C/C++ 탭에서 Debug Info를 디폴트인 Program Database For Edit and Continue에서 Program Database로 바꾸어야 한다는 점입니다. 만약 바꾸지 않으면, 디버그 실행시 에러가 발생하게 됩니다.

2.3 METIS 인스톨

METIS는 그래프 분할(graph partition)에 사용되는 라이브러리로 CHOLMOD 등에서 symbolic analysis할 때 사용합니다. 리스트 3의 순서에 따라 인스톨하면 됩니다.

2.4 CHOLMOD와 UMFPACK 인스톨

UMFPACK를 인스톨하기 위해서는 BLAS, AMD가 필요하고, CHOLMOD는 BLAS, AMD외에 LAPACK, COLAMD, CCOLAMD, METIS가 필요합니다. 또한 설정파일을 포함하고 있는 UFCONFIG가 공통적으로 필요합니다. 여기에서 BLAS, LAPACK, METIS 등은 앞서에서 인스톨한 것들을 사용하기로 합니다. 그 외의 AMD, COLAMD, CCOLAMD, UFCONFIG 등은 http://www.cise.ufl.edu/research/sparse/에서 다운로드할 수 있습니다. 인스톨은 리스트 4와 같이 수행하면 됩니다.

3. 간단한 사용예

리스트 5에 4×4 대칭 행렬을 CHOLMOD로 사용한 예를 나타냈습니다.

리스트 5에서 유의깊게 살펴보아야 할 부분은 행렬을 정의하는 방식입니다. 유한요소 교과서에 나오는 banded 행렬, skyline 행렬 등과 같이 모든 행렬 값을 저장하는 것이 아니라 0이 아닌 요소에 대해서만 행렬 값을 정의합니다. 하지만, 여기에서 사용하는 것은 Compressed Sparse Column format (CSC format)으로, 희소행렬을 정의하기 위해 pcol,

리스트 3 Installing METIS with CYGWIN-GCC

1. metis 4.0.1(http://glaros.dtc.umn.edu/gkhome/views/metis)을 다운 받아 압축을 푼다(/METIS 디렉토리).
2. metis-4.0.1-patch 폴더의 파일 3개를 rename.h, string.h, util.c를 METIS/Lib에 복사한다. 원래의 metis의 util.c에서는 stdout를 fflush(stdout), frpintf(stdout,...)과 같이 stdout을 사용하는 함수 errexit()가 정의되어 있다. 이 함수들을 호출하는 것은 cygwin1.dll에 의존하기 때문에 util.c에서 printf()함수로 교체하였다. 또한 MSVC에서 지원하지 않는 srand48, drand48 함수를 재정의한 부분이 있다. strings.h는 metis에서 사용하지 않는 헤더파일이고, MSVC에서 지원하지 않는다. 여기에서는 아무 내용이 없는 파일이다. 마지막으로 rename.h에서는 log2 함수의 재정의와 관련된 버그를 고친 것이다.
3. cygwin 커맨드라인에서 make를 실행한다.
4. 실행하고 나면, libmetis.a 파일이 생성된다. 이 파일을 /LIB 디렉토리로 이동시킨다.

리스트 4 Installing COLMOD, UMFPACK with CYGWIN

1. CHOLMOD(버전 1.0), UMFPACK(버전 4.6), AMD, COLAMD, CCOLAMD, UFCONFIG를 다운로드(http://www.cise.ufl.edu/research/sparse)하여 동일한 디렉토리에 나란히 압축을 푼다. (/CHOLMOD, /UMFPACK, /AMD, /COLAMD, /CCOLAMD, /UFCONFIG)
2. /UFCONFIG/UFconfg.mk 파일에서 BLAS, LAPACK, XERBLA, METIS_PATH, METIS, UMFPACK_CONFIG를 다음과 같이 설정한다.

```
BLAS = -lf77blas -lcblas -latlas -llapack -L../../atlas371BIN/WinNT_P4ESSE2
LAPACK = -llapack  -L../../atlas371BIN/WinNT_P4ESSE2
XERBLA = ../../UFconfig/xerbla/libcerbla.a
METIS_PATH = ../../metis-4.0
METIS = ../../metis-4.0/libmetis.a
UMFPACK_CONFIG = -DCBLAS -I../../ATLAS371BIN/WinNT_P4ESSE2 -DNPOSIX
```

3. /CHOLMOD/CHECK/cholmod_read.c에서 #include <ctype.h>를 #include <stdlib.h>로 바꾼다. 이것은 isspace() 등의 함수 때문이다.
4. /CHOLMOD/CORE/cholmod_error.c에서 Line 60~68 사이에 있는 fflush(stdout), flush(stderr) 함수를 comment 처리한다. 이것은 stdout,stderr 등의 표준 입출력 객체를 직접 사용할 때 호환성 문제가 발생하기 때문이다.
5. /CHOLMOD와 /UMFPACK에서 각각 make library을 수행한다.
6. 라이브러리 /CHOLMOD/LIB/libcholmod.a, /UMFPACK/LIB/libumfpack.a가 생성된다.

리스트 5 CHOLMOD 사용 예

```
/* ========================================================================== */
/* === cholmod_simple.cpp=================================================== */
/* ========================================================================== */

/*
[ 1.0 0.5 0.0 0.0 ]       [ 1.0 ]
[ 0.5 1.0 0.5 0.0 ]       [ 2.0 ]
[ 0.0 0.5 1.0 0.5 ] X   = [ 3.0 ]
[ 0.0 0.0 0.5 1.0 ]       [ 4.0 ]
x = [ 0 2 4 0]'
*/

#include "cholmod.h"
#include "stdio.h"

#define TRUE 1
#define FALSE 0

int main (void)
{
    double one [2] = {1,0}, m1 [2] = {-1,0} ;           /* basic scalars */

    cholmod_common c ;

    cholmod_start (&c) ;                          /* start CHOLMOD */

    int nrow=4;
    int ncol=4;
    int nzmax=7;

    cholmod_sparse                                  *A
cholmod_allocate_sparse(nrow,ncol,nzmax,1,1,-1,CHOLMOD_REAL,&c);
    c.print = 4;  // full print

    if(A == NULL)
    {
      cholmod_free_sparse(&A,&c);
      cholmod_finish(&c);
    }
    int* pcol = (int*) A->p;
    int* irow = (int*) A->i;  // this is void pointer.
    double* val = (double*) A->x;

    pcol[0]=0; pcol[1]=2; pcol[2]=4; pcol[3]=6; pcol[4]=7;
    irow[0]=0; irow[1]=1; irow[2]=1; irow[3]=2; irow[4]=2; irow[5]=3; irow[6]=3;
    val[0]=1.; val[1]=0.5; val[2]=1.; val[3]=0.5; val[4]=1.; val[5]=0.5; val[6]=1.0;
```

irow, val 이라는 변수들을 사용했는데, pcol은 열 포인터(column pointer)를 의미하고 irow는 행의 위치를 의미합니다. 그리고, val은 실제 값을 담고 있습니다. 위의 예제는 대칭행렬이기 때문에 행렬 중에서 아래의 삼각행렬(lower

리스트 6 비대칭 행렬에 대한 CSC format

$$\begin{pmatrix} 1. & -3. & 0 & -1. & 0 \\ 0 & 0 & -2. & 0 & 3. \\ 2. & 0 & 0 & 0 & 0 \\ 0 & -4. & 0 & -4. & 0 \\ 5. & 0 & 5. & 0 & 6. \end{pmatrix}$$

```
C/C++인 경우( 0-based offset)
colptr[7] =  {0 3 5 7  9 11}
irow[11] =   {0 2 4  0 3 1 4 0 3 1 4}
values[11] = {1. 2. 5. -3. 4. -2. -5. -1 -4. 3. 6.}

Fortran인 경우 ( 1-based offset)
colptr[7] =  {1 4 6 8 10 12}
irow[11] =   {1 3 5  1 4 2 5 1 4 2 5}
values[11] = {1. 2. 5. -3. 4. -2. -5. -1 -4. 3. 6.}
```

triangle)만 정의하고 있습니다. 만약 UMFPACK 같이 비대칭행렬을 다룰 때는 리스트 6과 같이 정의하게 됩니다.

4. 맺음말

이번 회는 최신 희소 행렬 solver의 필요성과 CHOLMOD, UMFPACK 등 opensource 솔버를 인스톨하는 방법에 대해 설명하였습니다. 다음 연재에서는 Fortran에서 호출하는 것에 대해 설명합니다. C 언어로 작성된 라이브러리를 Fortran에서 호출하여 사용하기 때문에 이러한 것들을 혼합 컴파일(mixed compilation)이라 합니다. 그리고 최근 유한요소 프로그램에서 많이 도입되고 있는 Python과 같은 스크립트 언어에 대해 소개하도록 하겠습니다.