```
// ------------------------------------------------------------------------
// matrix multiplication
//
// 0. choose appropriate compiler version e.g.
//
//      module load intelcompiler/composer_xe_2013.4.183
//      (see "module avail" for available versions)
//
// 1. compile with automatic parallelization and appropriate reporting level
//
//      icc -O3 -parallel -par-report2 matmult.c -lrt -o matmult
//
// 2. set number of threads and prohibit dynamic adjustment of number
//
//      export set OMP_DYNAMIC=FALSE
//      export set OMP_NUM_THREADS=4
//
//    optionally bind threads to cores by *one* of the following:
//
//      export set GOMP_CPU_AFFINITY="0 1 2 3"
//
//      export set KMP_AFFINITY=
//        "verbose,granularity=core,explicit,proclist=[0,1,2,3]"
//
// 3. prepare runtime monitoring in other window showng all threads
//
//     top -u <username> -H
//     (press "f j <ENTER>" to see in column "P" mapping to cores)
//
// 4. execute with timing switched on
//
//     time ./matmult
//
// ------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 2000

double A[N][N], B[N][N], C[N][N];

main(int argc, char *argv[])
{
  int i, j, k, t;
  double s;
  struct timespec t1, t2;

  // initialize A and B
  for (i=0; i<N; i++)
  {
    for (j=0; j<N; j++)
    {
      A[i][j] = rand();
      B[i][j] = rand();
    }
  }

  // print part of A and B
  printf("%f %f\n", A[0][0], B[0][0]);

  // start wall-clock timing
  clock_gettime(CLOCK_REALTIME, &t1);

  for (i=0; i<N; i++)
```

```
  {
    for (j=0; j<N; j++)
    {
      s = 0;
      for (k=0; k<N; k++)
      {
        s += A[i][k]*B[k][j];
      }
      C[i][j] = s;
    }
  }

  // end wall-clock timing
  clock_gettime(CLOCK_REALTIME, &t2);
  t = (t2.tv_sec-t1.tv_sec)*1000+(t2.tv_nsec-t1.tv_nsec)/1000000;

  // print part of C and elapsed wall clock time
  printf("%f (%d ms)\n", C[0][0], t);
  return 0;
}
```