```
// ---------------------------------------------------------------------------
// matrix multiplication
//
// 0. choose appropriate compiler version e.g.
//
//      module load intelcompiler
//
// 1. compile with MKL and OpenMP and Pthread library included
//
//      icc -O3 -lmkl -liomp5 -lpthread matmult.c -o matmult
//
// 2. set number of threads
//
//      export CILK_NWORKERS=4
//
//    (only if not set within program by __cilkrts_set_param()).
//
// 3. prepare runtime monitoring in other window showng all threads
//
//      top -u <username> -H
//
// 4. execute with timing switched on
//
//      time ./matmult <threads> [recursive]
//
// ---------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mkl_gmp.h>

#include <cilk/cilk_api.h>

#define N 500

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

static void multiply(int from, int to);

int main(int argc, char *argv[])
{
  if (argc < 2)
  {
    printf("synopsis: matmult <threadnumber> [recursive]");
    exit(-1);
  }
  __cilkrts_set_param("nworkers", argv[1]);

  int i;

  // initialize A and B
  for (i=0; i<N; i++)
  {
    int j;
    for (j=0; j<N; j++)
    {
      mpz_init(A[i][j]);
      mpz_init(B[i][j]);
      mpz_init(C[i][j]);

      mpz_set_si(A[i][j], rand()); // A[i][j] = rand()
      mpz_set_si(B[i][j], rand()); // B[i][j] = rand()
    }
  }
```

```c
  // print part of A and B
  mpz_out_str(stdout, 10, A[0][0]); printf("\n");
  mpz_out_str(stdout, 10, B[0][0]); printf("\n");

  if (argc >= 3 && strcmp(argv[2], "recursive") == 0)
  {
    // recursive multiplication
    multiply(0, N);
    mpz_out_str(stdout, 10, C[0][0]); printf("\n");
    return 0;
  }

  // iterative multiplication
  _Cilk_for (i=0; i<N; i++)
  {
    // each thread must have a local copy of j, k, s, p
    int j, k;
    mpz_t s, p;
    mpz_init(s);
    mpz_init(p);
    for (j=0; j<N; j++)
    {
      mpz_set_si(s, 0);                // s = 0
      for (k=0; k<N; k++)
      {
        mpz_mul(p, A[i][k], B[k][j]); // p = A[i][k]*B[k][j]
        mpz_add(s, s, p);             // s = s+p
      }
      mpz_set(C[i][j], s);            // C[i][j] = s
    }
  }

  mpz_out_str(stdout, 10, C[0][0]); printf("\n");
  return 0;
}

static void multiply(int from, int to)
{
  if (from < to-1)
  {
    // recursive case: fork two threads and join
    int mid = (from + to)/2;
    _Cilk_spawn multiply(from, mid);
    _Cilk_spawn multiply(mid, to); // spawn before sync is redundant
    _Cilk_sync;
    return;
  }

  // empty base case
  if (from > to-1) return;

  // non-empty base case (from == to-1)
  int i = from;
  int j, k;
  mpz_t s, p;
  mpz_init(s);
  mpz_init(p);
  for (j=0; j<N; j++)
  {
    mpz_set_si(s, 0);                // s = 0
    for (k=0; k<N; k++)
    {
      mpz_mul(p, A[i][k], B[k][j]); // p = A[i][k]*B[k][j]
      mpz_add(s, s, p);             // s = s+p
    }
    mpz_set(C[i][j], s);            // C[i][j] = s
  }
}
```