

## Assignment 6: Multilevel Static DGEMM Blocking

Due: Monday 03/02/05, Multiplication Factor: 2.0

In this assignment, you should implement a statically blocked DGEMM, as discussed in class. To get the files you need, perform:

```
cp ~whaley/teach/cis5930/ASG6/Make* .
cp ~whaley/teach/cis5930/ASG6/*.c .
make archdirs
```

The only file you are allowed to modify is `dgemm_sblk.c`, which is where you should implement your statically blocked DGEMM. You will send a completed `dgemm_sblk.c` to `whaley@cs.fsu.edu` by 10AM by the due date.

Your first order of business is to get correct answers when only blocking for the first level of the cache. Since we are supporting non-square blocks, you can vary the blocking by changing the macros `MB`, `NB` and `KB`. Note that with static blocking, there should be no need to prescale  $C$  by  $\beta$  in your interface routine.

Once things work, you may want to try increasing the performance by changing the block factor, trying non-square cases, etc. The Makefile is set up to compile the tester with varying blocking parameters automatically, if you pass in the values you want to use:

```
make nkb=XX mb=XX nb=XX kb=XX
```

`nkb` defaults to 0, and is the partitioning for L2 blocking (see below). The rest of the parameters are the blocking factors for the respective dimensions. They all default to whatever `nb` is set to, and `nb` defaults to 40.

The code that you compare (both answers and performance) against is my recursive implementation from assignment 4. Without L2 blocking, do you see a gap between your performance and it for large problems?

Once the first level of blocking is working, you'll want to implement the crude L2-blocking that we covered in class. In particular, you will pass into the Makefile `nkb`, which sets the number of KB-sized blocks to allow in partitioning  $K$  in order to guarantee reuse of the panel of  $A$  in the L2. Can you use this plus the non-square L1 blocking in order to have a statically blocked algorithm that runs faster than the generically blocked recursive scheme?

You will be graded on both correctness and relative (to best code) performance.

Your routine should get the correct answer, regardless of the setting of  $\alpha$  and  $\beta$ . To test a bunch of differing scalars with a square problem of size 300, issue:

```
./xdgemmtst -a 3 0 1 2.1 -b 3 0 1 0.8 -n 300
```

You must also produce correct answers (and decent performance) for non-square matrices. To test a gemm with  $M=50$ ,  $N=20$ ,  $K=77$ , issue:

```
./xdgemmtst -m 50 -n 20 -k 77
```

With no arguments, square problems are run between [100,1000:100]. You can get some usage info by issueing: `xgemmtst --help`. Note that I might not be so nice as to grade on these exact problem sizes, so try to make your block settings as size-independent as possible. You may want to time larger problems in order to see L2 blocking affects (eg., `./xgemmtst -N 600 1400 200`).

Once you have tuned your parameters to your satisfaction on program and linprog, define NB, MB, KB, and NKB based on whether or not the macro SPARC is defined, overriding anything passed in during compile time. For instance, if 40 is the best NB, for the SPARC, and 36 the best for linprog, you'd do:

```
#ifdef NB
    #undef NB
#endif
#ifdef SPARC
    #define NB 40
#else
    #define NB 36
#endif
```