

Assignment 8: Tuning Aligned SDOT using assembly and SSE

Due: Wednesday 03/30/05, Multiplication Factor: 2.5

In this assignment, we will optimize the X and Y aligned loop from assignment 7. The main grade will be on performance as measured by the provided tester. I will also be grading on readability, as assembly needs to be made as clear as possible if it is to be maintained. Use the cpp tricks we discussed in class, and provide good (though not effusive) comments, particularly when you are doing something non-straightforward. Do not leave debugging information in your code. You may assume the vector length is at least one, and should optimize for extremely long, out-of-cache vectors (eg., $N=80000$).

I suggest starting with your `sdot_sse.S` from assignment 7, and then optimizing the aligned loop. As you optimize, just keep N a multiple of the unroll factor you are timing/testing. You may want to wait until the routine is fully optimized (and thus any unroll/blocking factors are known) before putting in the calls to handle the cleanup appropriately, depending on how you implemented in asg 7.

For these long out-of-cache vectors, you'll probably need to do some memory optimizations such as block fetch and/or prefetch, and fetch pipelining. You'll also want to examine all the computational optimizations we've discussed. Certainly loop unrolling, scalar (accumulator) expansion, and scheduling would be of interest. To get the files you need, perform:

```
cp ~whaley/teach/cis5930/ASG8/Makefile .
make archdirs
```

You should create the file `sdot_sse.S` in the source directory, and this is the file you should e-mail to whaley@cs.fsu.edu by 10AM on the due date.

I have provided a tester and timer for this routine which allows you to control the alignment of your input arrays. You build and run the tester and timer with :

```
make [test,time] N=<#>
```

Where `#` is the size of the vector to be timed. If no value of N is given in the make command, a value of 80000 is assumed. Both programs align both vectors to 16 bytes by default.

Sometimes it helps to optimized the function computationally first, and then add the memory optimizations once the computation runs at peak rate assuming things are in cache. The timer supports this. To make the timer give you in-cache results, choose an N small enough to fit, and pass the `cacheSize` parameter (`CS`) of zero, so no cache flushing occurs. For instance:

```
make time N=1024 CS=0
```

Note if you are using block fetch, you'll want to choose N as your fetch blocking, and comment out the fetch loop. Note that I'm not necessarily advocating this approach, just mentioning that it is possible.