

## Assignment 9: Tuning Aligned DDOT using x86-64 assembly and SSE2

Due: Monday 04/34/05, Multiplication Factor: 2.0

In this assignment, we will write two implementations of `ddot` using SSE2. For both implementations, you need only implement the loops that assume  $X$  and  $Y$  have the same alignment (i.e., I will test only those cases where  $X\%16 == Y\%16$ ). The main grade will be on performance as measured by the provided timer. 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.,  $N=80000$ ).

The two implementations you need to write (in the order in which I suggest tackling them) are:

1. A simple `ddot` that works for both x86-32 and x86-64. Only correctness will matter for this code (though the main loop must be vectorized). Call this file `ddot_x86.S`.
2. A `ddot` specialized and optimized for x86-64. This loop must produce correct answers, and will be graded by performance, as measured by my timer on `animal.cs.fsu.edu`. This file should be called `ddot_sse.S`. My initial implementation gets roughly 433 MFLOPS.

Both these files may be given in the same file if you like. If you do this, name the combined file `ddot_sse.S`.

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. It's probably not a good idea to assume that optimizations that are the best for the PIII are also the best for the Opteron. To get the files you need, perform the following in `animal.cs.fsu.edu`:

```
cp ~rwhaley/teach/cis5930/ASG9/Make* .
make archdirs
```

You should create the files `ddot_sse.S` and `ddot_x86.S` in the source directory, and you should e-mail them 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 mem-

ory 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.

If the file you are testing is not `../ddot_sse.S`, then be sure to specify it by adding `urout=<file>` to the appropriate make command.