Methods Used

The most prominent method I used was parallelizing the for loops in dwt3d_main.c, and the fwt97_* functions in cdf97.c.
(about 2x speedup)

I also used loop unrolling in various locations to maximize cache hits on parallel iterations.
(less than 1.5x speedup)

Within fwt_dl_foo and fwt_dl_bar, I eliminated the even/odd check by creating two separate for-loops, one that iterates along the even number and one along the odd (and also implement loop unrolling). I saved a lot of time taking out the logic check. I attempted to run the loops in parallel, but the overhead was too high.
(about 2x speedup)

Lastly, I simplified some dereferencing by using pointer offsets rather than multiplicative dereferencing of the predefined x(i, j) construct.(see x_temp variable in cdf97.c).
(less than 1.5x speedup)

Performance Results (in seconds)

# Threads	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
1	5.14	3.90	4.46	3.93	4.08
2	2.38	2.97	3.38	2.33	2.12
4	1.23	2.2	1.34	1.51	1.46
8	0.92	0.97	0.95	1.32	1.35
16	0.92	1.92	1.20	0.83	0.56

My run times were pretty inconsistent, mostly because many other people were testing their programs as well (I could tell by a call to top). When I ran my program on a different Linux server (e.g. 02, 03, 04, etc.) that weren't crowded, my run times were extremely consistent.

Intuition would state that running the program on one thread would yield the same runtime as the sequential code. A possible explanation (beyond a crowded server) is that the overhead of the omp pragma might still be affecting the program, but since there is only one thread, there is no experience speedup from parallelization.