CS540: Final Exam Slides

Experiments with the WHT Package

Dustin Ingram
Teamates: Aaron Rosenfeld, Tom Wambold

Drexel University Department of Computer Science, Philadelphia PA

December 8, 2011

What we did...

- Implemented the following versions of the WHT:
  - Iterative;
  - Recursive;
  - Vectorized Iterative using SSE;
  - Vectorized Recursive using SSE;
  - Multithreaded Recursive using OpenMP;
  - Multithreaded Recursive using Pthreads;
  - Multithreaded Recursive using Hybrid OpenMP/Pthreads.
- Made the following measurements using the WHT package:
  - Iterative speed;
  - Recursive speed;
  - Best found w/ Dynamic Programming;
  - WHT Performance.

Outline

System Background

- Platform Information

```
$ uname -a
Linux float.cs.drexel.edu 2.6.35-28-generic #50-Ubuntu SMP x86_64 GNU/Linux
```

- Compiler Information

```
$ gcc --version
gcc (Ubuntu/Linaro 4.4.4-14ubuntu5) 4.4.5
```

- Options Used

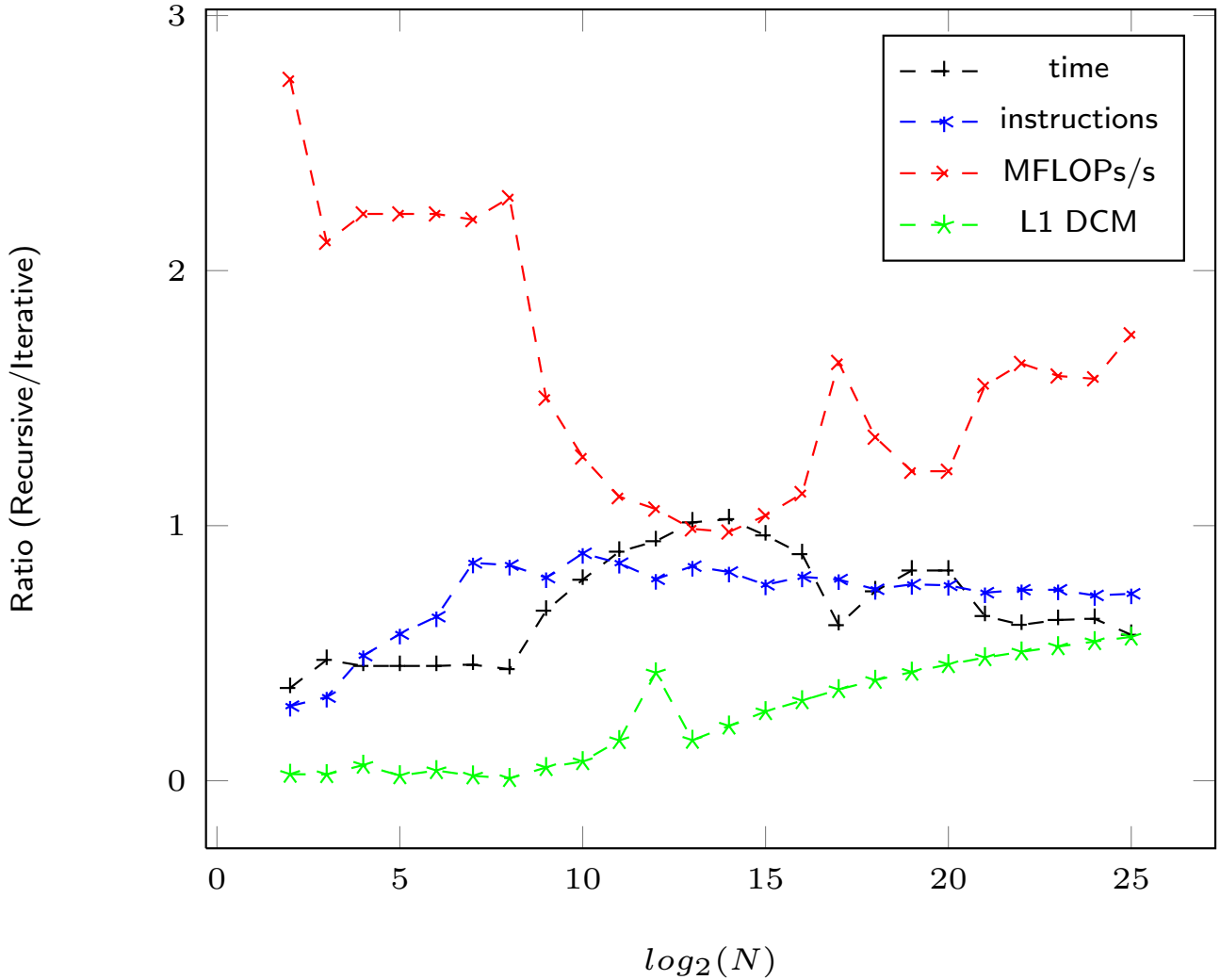
```
-std=gnu99 -Wall -fopenmp -msse -msse2 -lm -lpthread -lpapi -O3
```

Recursive and Iterative Implementations

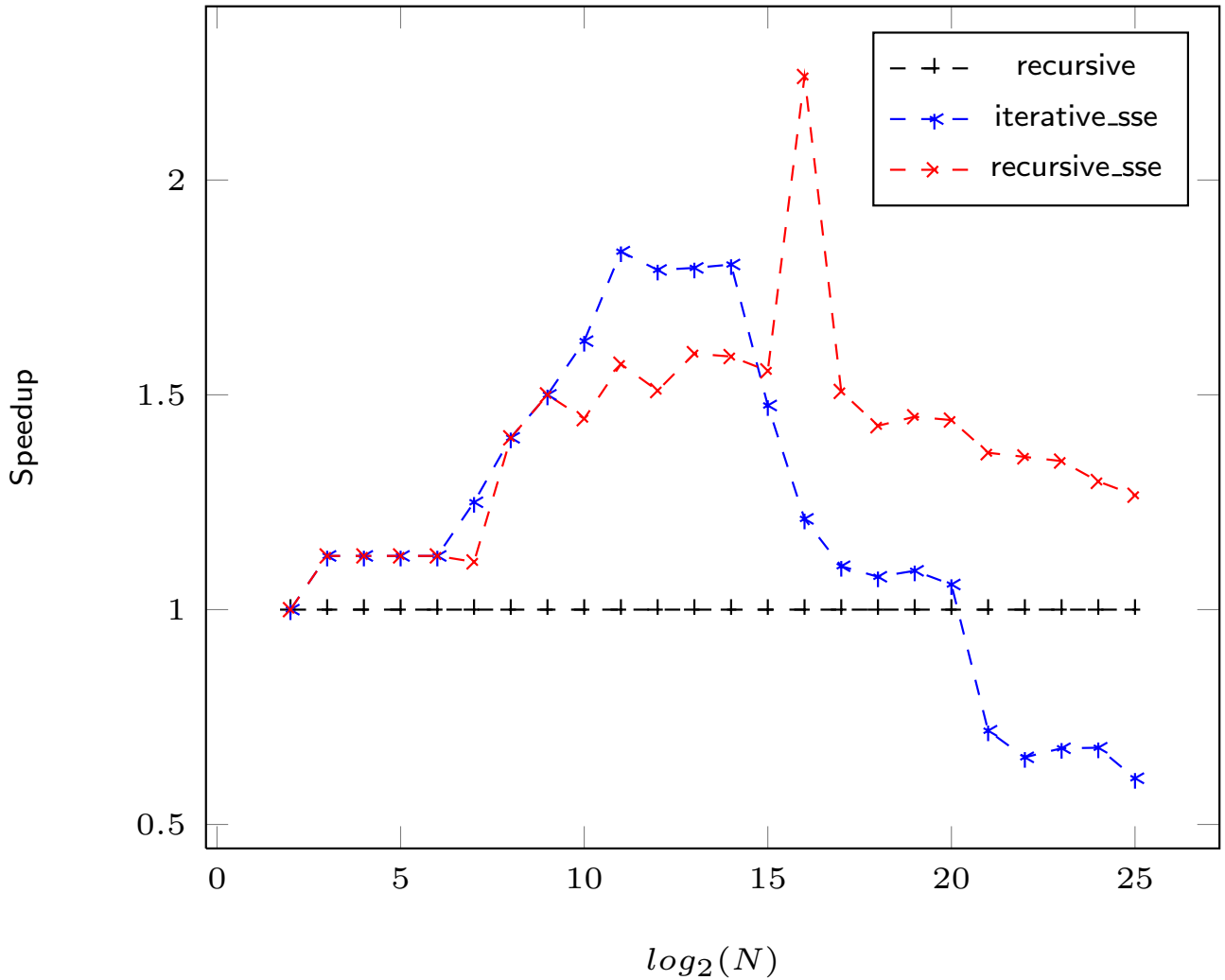
```
void wht_iterative(double *a, int n) {
    int m = n;
    int N = 1;
    for (int k = 0; k < log2(n); k++) {
        m /= 2;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < N; j++) {
                int i1 = i * 2 * N + j;
                int i2 = i * 2 * N + j + N;
                double x = a[i1];
                double y = a[i2];
                a[i1] = x + y;
                a[i2] = x - y;
            }
        }
        N *= 2;
    }
}
```

```
void wht_recursive(double *a, int n) {
    int half = n / 2;

    if (n == 2) {
        double x = a[0];
        double y = a[half];
        a[0] = x + y;
        a[half] = x - y;
        return;
    }

    wht_recursive(a, half);
    wht_recursive(a + half, half);

    for (int i = 0; i < half; i++) {
        double x = a[i];
        double y = a[i + half];
        a[i] = x + y;
        a[i + half] = x - y;
    }
}
```

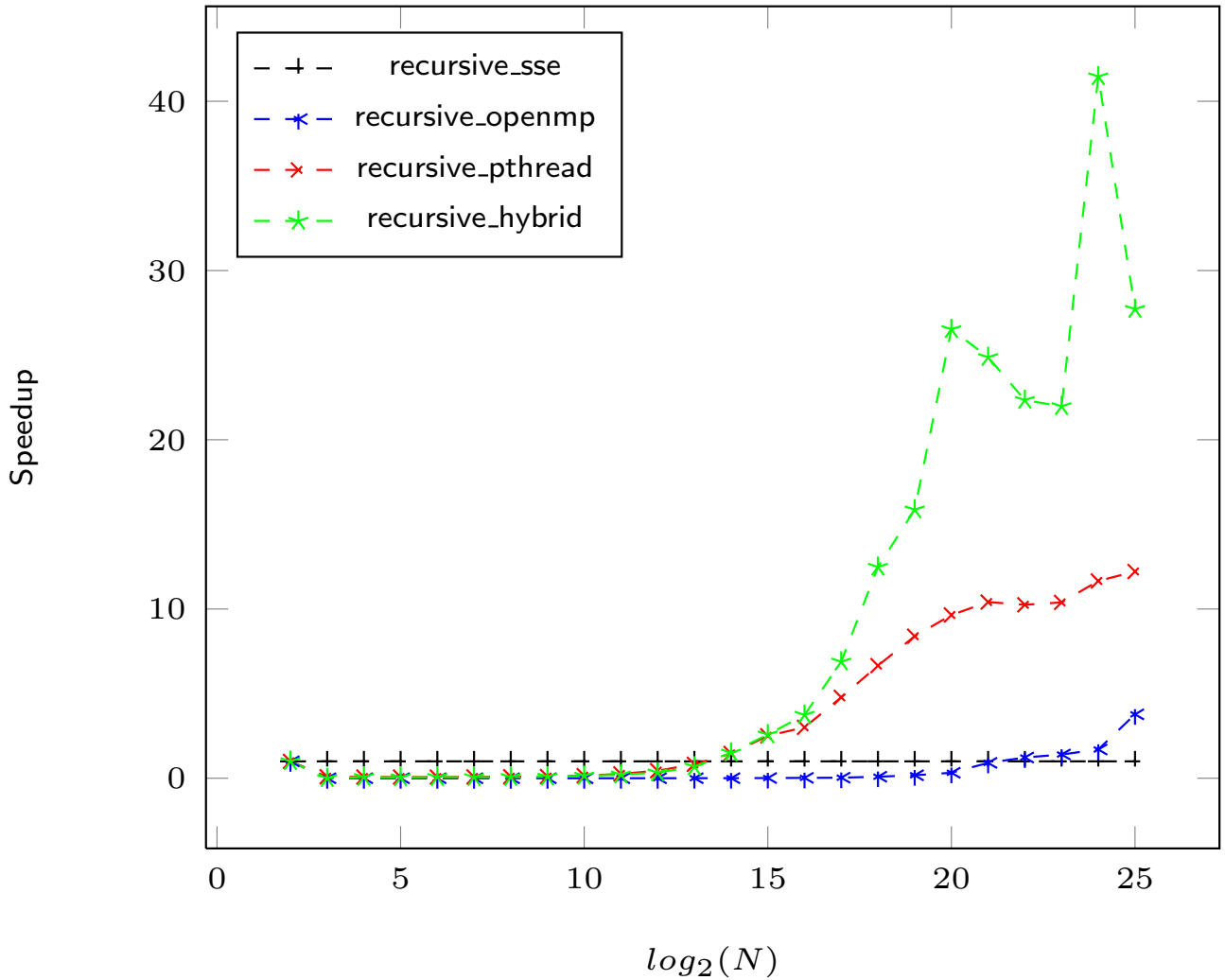
Performance Summarization



Vectorized vs. Non-Vectorized Recursive Speedup



Multithreaded vs. Vectorized Recursive Speedup



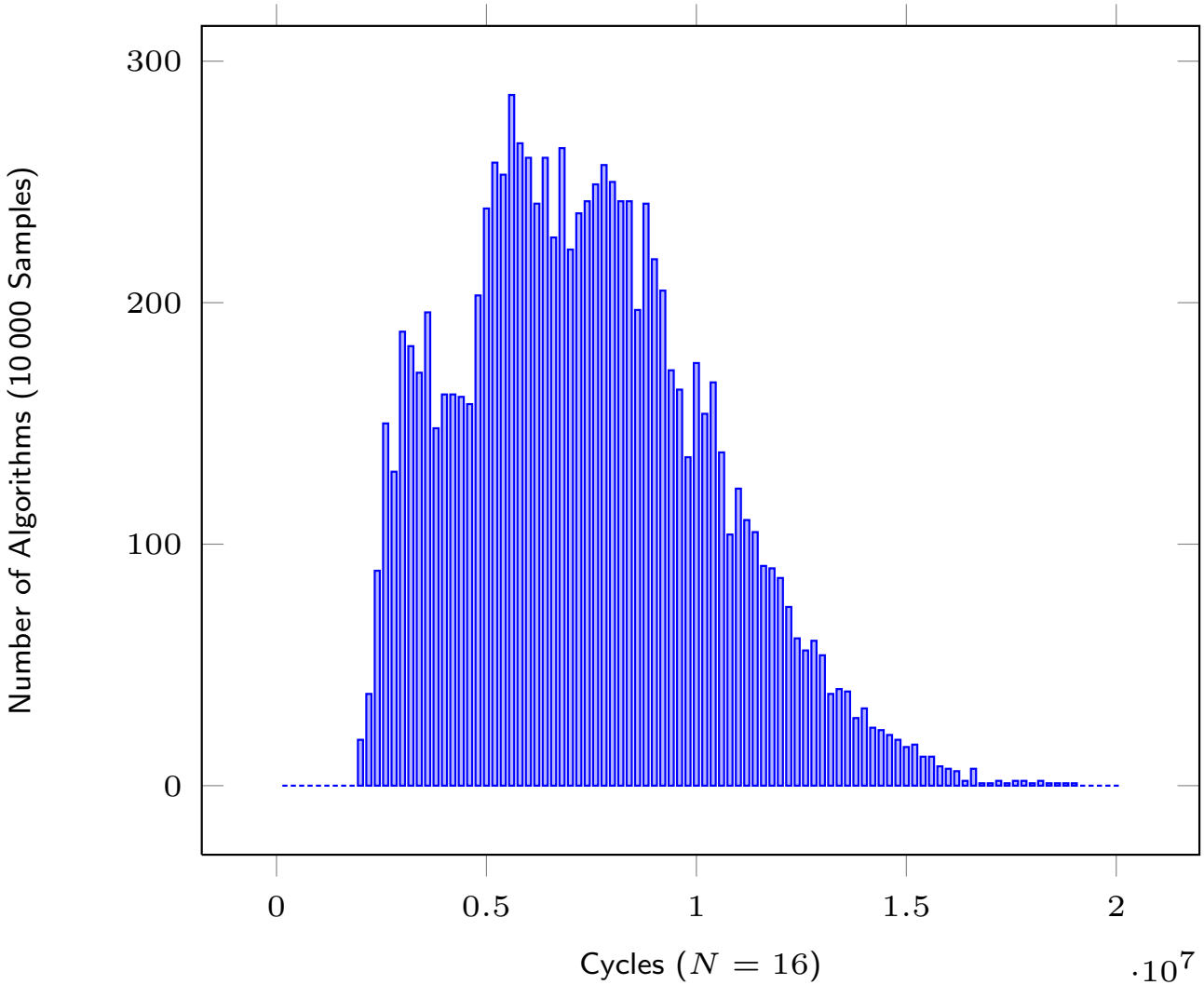
Vectorized vs. Non-Vectorized Recursive Speedup

- No different radix versions were used.

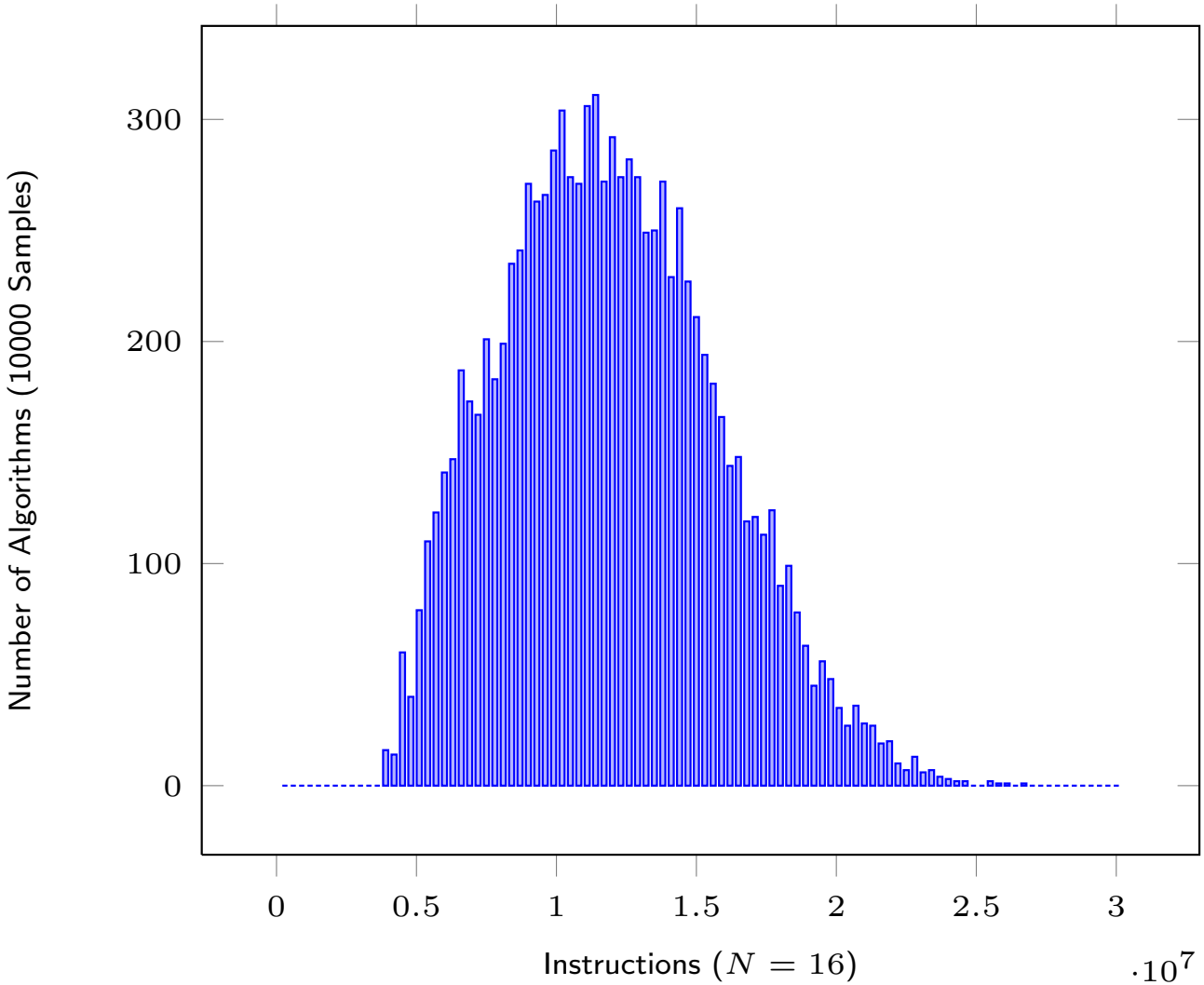
WHT Package Data - Best by Dynamic Programming

N	Time (μs)	Plan
1	354.0	`small[1]`
2	357.0	`small[2]`
3	518.0	`small[3]`
4	604.0	`small[4]`
5	1061.0	`small[5]`
6	1379.0	`split[small[3],small[3]]`
7	2210.0	`split[small[4],small[3]]`
8	3647.0	`split[small[4],small[4]]`
9	8005.0	`split[small[3],small[3],small[3]]`
10	16564.0	`split[small[4],small[3],small[3]]`
11	34021.0	`split[small[4],small[4],small[3]]`
12	70951.0	`split[small[4],small[4],small[4]]`
13	192616.0	`split[small[3],small[3],small[3],small[2],small[2]]`
14	385364.0	`split[small[3],small[3],small[3],small[3],small[2]]`
15	769626.0	`split[small[3],small[3],small[3],small[3],small[3]]`
16	1855763.0	`split[small[3],small[3],small[3],small[3],small[2],small[2]]`
17	3715648.0	`split[small[3],small[3],small[3],small[3],small[3],small[2]]`

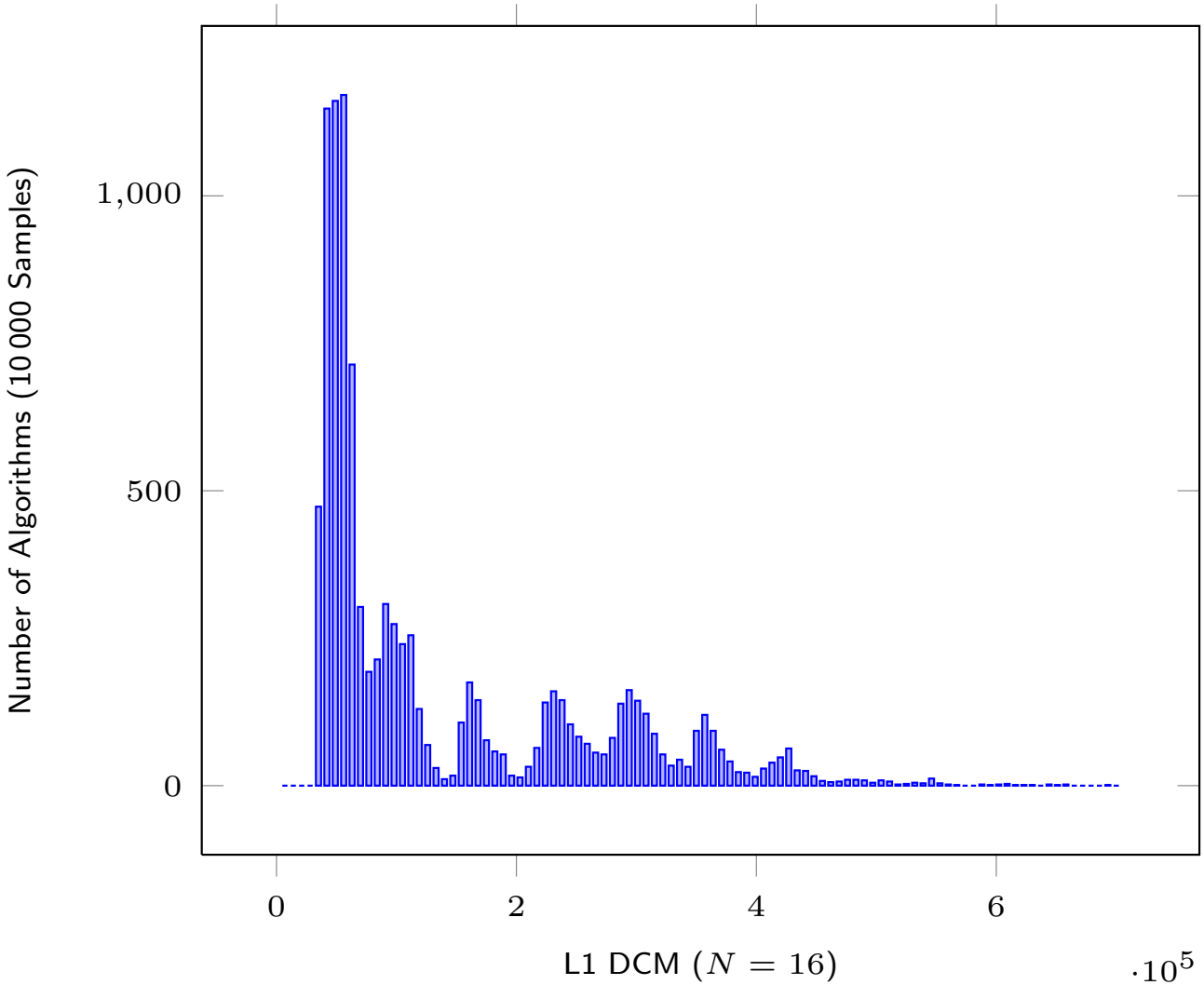
WHT Performance - Cycles



WHT Performance - Instructions



WHT Performance - L1 DCM



Conclusion

- We see significant improvements with the recursive WHT algorithm over the iterative version;
- Vectorization of the algorithm will improve speed, but for a large length, this only works for the vectorized recursive version;
- Adding multithreading to the algorithm significantly improves the speed, by up to a factor of 40;
- A hybrid openmp/Pthreads algorithm provided the fastest speedup;
- Examining random WHT package performance via a histogram reveals trends in the cycles, instruction count, and L1 DCM for the processor.