CSE691 HW2

Yuchen Wang
905508464

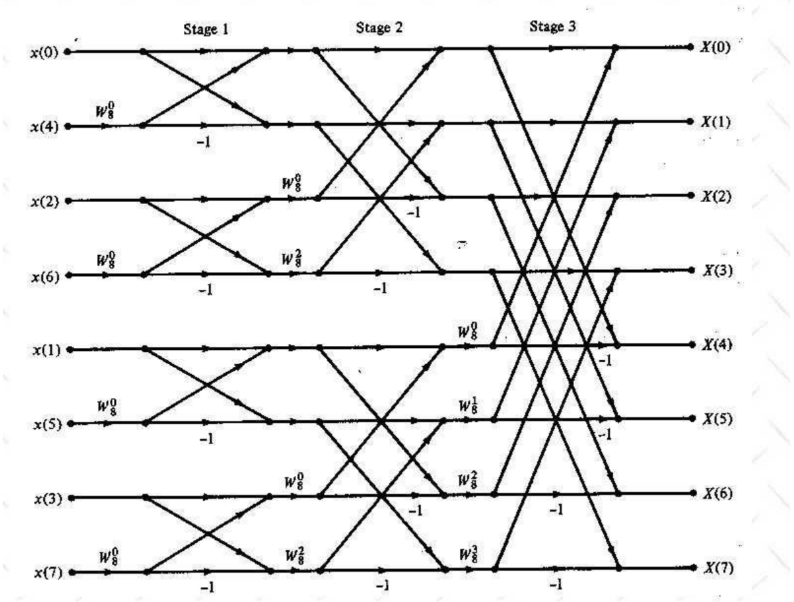
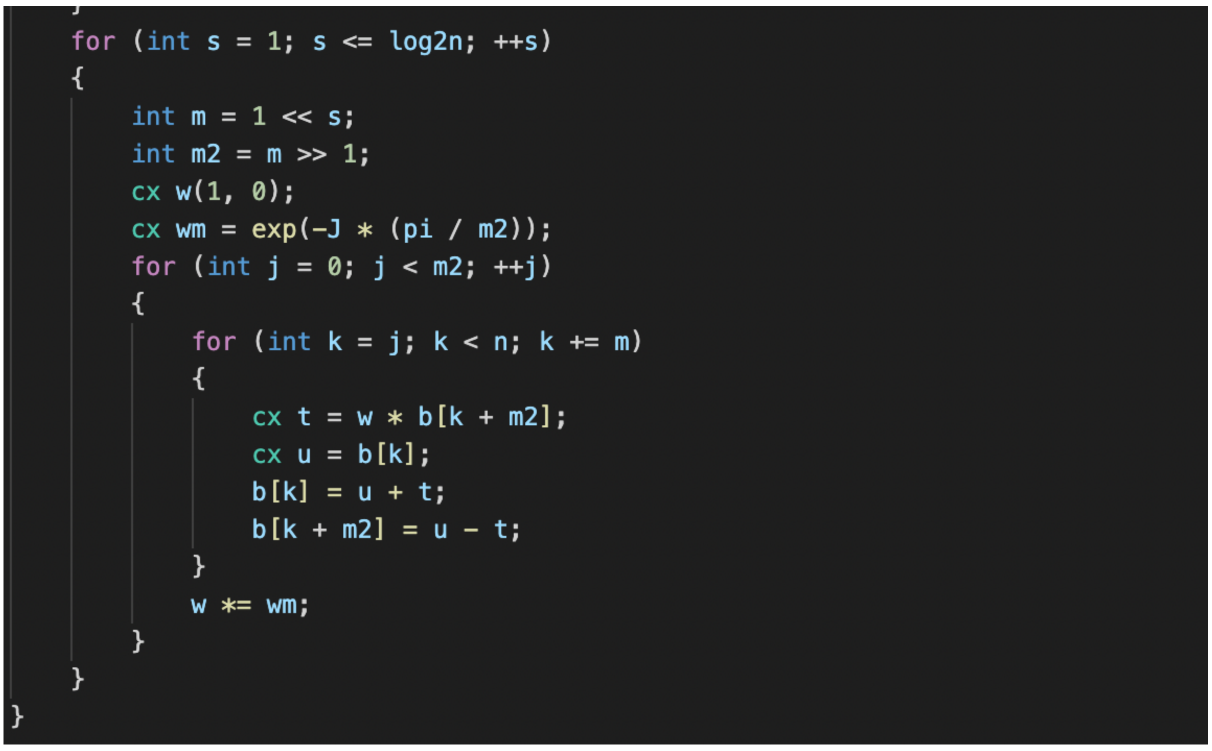


Figure TC.3.3 Eight-point decimation-in-time FFT algorithm.

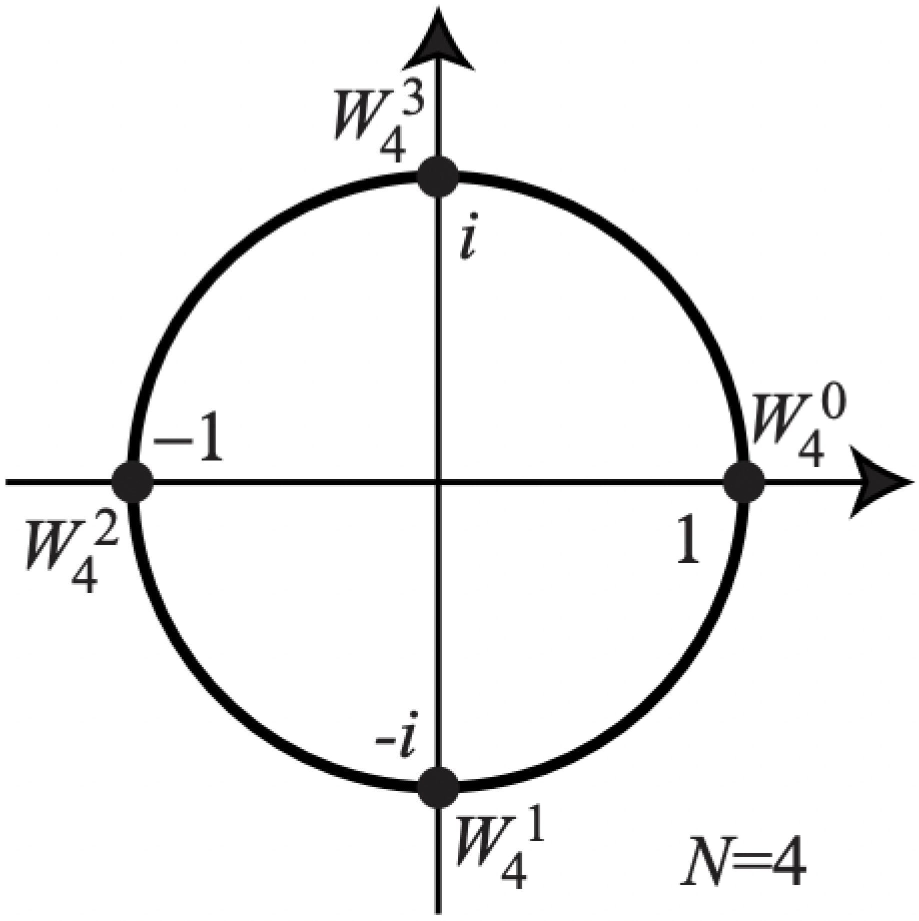
Based on the butterfly diagram we can notice one thing: The calculation of the next stage is based on the result of the previous stage, thus there can not be any multithreading operation between threads.



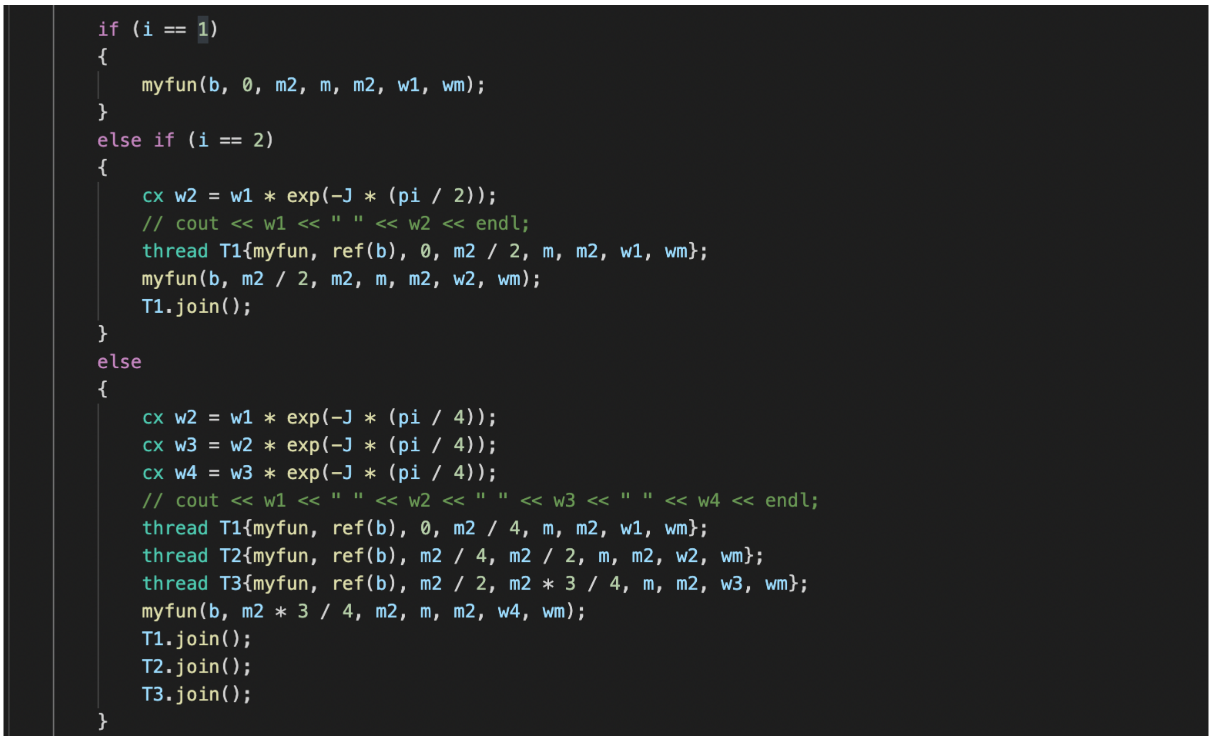
```
    for (int s = 1; s <= log2n; ++s)
    {
        int m = 1 << s;
        int m2 = m >> 1;
        cx w(1, 0);
        cx wm = exp(-J * (pi / m2));
        for (int j = 0; j < m2; ++j)
        {
            for (int k = j; k < n; k += m)
            {
                cx t = w * b[k + m2];
                cx u = b[k];
                b[k] = u + t;
                b[k + m2] = u - t;
            }
            w *= wm;
        }
    }
}
```

Therefore, we can only have multithreading on the two inner nested loops, but not the most outside one.

Regarding the outside loop of the two inner nested loops, we can see that the ‘w’ in the next iteration depends on the ‘w’ in the previous iteration. However, we can still do multithreading on it by doing some tricks.



Based on the definition of ‘w’s, we know ‘w’s are the on unity cycle with equivalent degree differences of ‘wm’, and a ‘w’ multiplying by ‘wm’ is equivalent to rotation clockwise by this angle. Also, all the ‘w’s are evenly located on the unity cycle. Hence, if we want to run four threads in parallel, we can start the first thread from ‘w’ = (1, 0), then start the second with ‘w’ rotating 90 degrees of ‘wm’, then rotate to get the start point of threads three and so on.



```
if (i == 1)
{
    myfun(b, 0, m2, m, m2, w1, wm);
}
else if (i == 2)
{
    cx w2 = w1 * exp(-J * (pi / 2));
    // cout << w1 << " " << w2 << endl;
    thread T1{myfun, ref(b), 0, m2 / 2, m, m2, w1, wm};
    myfun(b, m2 / 2, m2, m, m2, w2, wm);
    T1.join();
}
else
{
    cx w2 = w1 * exp(-J * (pi / 4));
    cx w3 = w2 * exp(-J * (pi / 4));
    cx w4 = w3 * exp(-J * (pi / 4));
    // cout << w1 << " " << w2 << " " << w3 << " " << w4 << endl;
    thread T1{myfun, ref(b), 0, m2 / 4, m, m2, w1, wm};
    thread T2{myfun, ref(b), m2 / 4, m2 / 2, m, m2, w2, wm};
    thread T3{myfun, ref(b), m2 / 2, m2 * 3 / 4, m, m2, w3, wm};
    myfun(b, m2 * 3 / 4, m2, m, m2, w4, wm);
    T1.join();
    T2.join();
    T3.join();
}
```

However, as we only starts from one point when i == 1 and starts from two points when i == 2, we can only have two threads when i == 2 and 1 thread when i == 1.