Lab 5 Open Ended Portion

Naive MI output:

```
matmul_naive: 2069300 cycles, 63.1 cycles/iter, 17.2 CPI
```

Naive MSI output:

```
matmul_naive: 544681 cycles, 16.6 cycles/iter, 4.5 CPI
```

Final MI output:

```
matmul_opt: 237810 cycles, 7.2 cycles/iter, 2.7 CPI
```

Final MSI output:

```
matmul_opt: 149630 cycles, 4.5 cycles/iter, 1.7 CPI
```

Collect results for your optimized implementation with both the MI and MSI protocols. What did you do differently to achieve better performance over the naive matmul?

We reorder the loops and split up per core to minimize the amount of overlapping loads. This helped increase cache hits and minimize false sharing and invalidations. We preloaded each A value in registers so that we don't have to load it ever again while doing the while loop. Lastly, we unrolled the loop so that we minimize branching. We figured that trying to add barriers and synchronization would slow down the program because each thread would stall waiting for the memory to write back.

For MSI, we have a 56.33% improvement in cycle count over the naive version and we have a 73.8% improvement in CPI. For MI, we have 88.5% improvement in cycle count over the naive version and we have a 37% improvement in CPI.

Code:

```
        void matmul_opt(unsigned int coreid, unsigned int ncores, const size_t lda,  const data_t* A,
const data_t* B, data_t* C)

{

  /* TODO */

  size_t i, j, k;



  for (j = coreid * (lda / ncores); j < (coreid + 1) * (lda / ncores); j++) {

     size_t j_lda = j * lda;

     for (k = 0; k < lda; k += 2) {

        data_t A_val = A[j_lda + k];

        size_t k_lda = k * lda;

        //data_t A_val1 = A[j_lda + (k + 1)];

        for (i = 0; i < lda; i += 32) {

           data_t B_val = B[k_lda + i];

           data_t B_val1 = B[k_lda + (i + 1)];

           data_t B_val2 = B[k_lda + (i + 2)];

           data_t B_val3 = B[k_lda + (i + 3)];



           C[i + j_lda] += A_val * B_val;

           data_t B_val4 = B[k_lda + (i + 4)];



           C[(i + 1) + j_lda] += A_val * B_val1;
```

```
data_t B_val5 = B[k_lda + (i + 5)];


C[(i + 2) + j_lda] += A_val * B_val2;

data_t B_val6 = B[k_lda + (i + 6)];


C[(i + 3) + j_lda] += A_val * B_val3;

data_t B_val7 = B[k_lda + (i + 7)];


C[(i + 4) + j_lda] += A_val * B_val4;

data_t B_val8 = B[k_lda + (i + 8)];


C[(i + 5) + j_lda] += A_val * B_val5;

data_t B_val9 = B[k_lda + (i + 9)];


C[(i + 6) + j_lda] += A_val * B_val6;

data_t B_val10 = B[k_lda + (i + 10)];


C[(i + 7) + j_lda] += A_val * B_val7;

data_t B_val11 = B[k_lda + (i + 11)];


C[(i + 8) + j_lda] += A_val * B_val8;
```

```
        data_t B_val12 = B[k_lda + (i + 12)];


        C[(i + 9) + j_lda] += A_val * B_val9;

        data_t B_val13 = B[k_lda + (i + 13)];


        C[(i + 10) + j_lda] += A_val * B_val10;

        data_t B_val14 = B[k_lda + (i + 14)];


        C[(i + 11) + j_lda] += A_val * B_val11;

        data_t B_val15 = B[k_lda + (i + 15)];


        C[(i + 12) + j_lda] += A_val * B_val12;

        data_t B_val16 = B[k_lda + (i + 16)];


        C[(i + 13) + j_lda] += A_val * B_val13;

        data_t B_val17 = B[k_lda + (i + 17)];


        C[(i + 14) + j_lda] += A_val * B_val14;

        data_t B_val18 = B[k_lda + (i + 18)];


        C[(i + 15) + j_lda] += A_val * B_val15;
```

```
data_t B_val19 = B[k_lda + (i + 19)];


C[(i + 16) + j_lda] += A_val * B_val16;

data_t B_val20 = B[k_lda + (i + 20)];


C[(i + 17) + j_lda] += A_val * B_val17;

data_t B_val21 = B[k_lda + (i + 21)];


C[(i + 18) + j_lda] += A_val * B_val18;

data_t B_val22 = B[k_lda + (i + 22)];


C[(i + 19) + j_lda] += A_val * B_val19;

data_t B_val23 = B[k_lda + (i + 23)];


C[(i + 20) + j_lda] += A_val * B_val20;

data_t B_val24 = B[k_lda + (i + 24)];


C[(i + 21) + j_lda] += A_val * B_val21;

data_t B_val25 = B[k_lda + (i + 25)];


C[(i + 22) + j_lda] += A_val * B_val22;
```

```
data_t B_val26 = B[k_lda + (i + 26)];


C[(i + 23) + j_lda] += A_val * B_val23;

data_t B_val27 = B[k_lda + (i + 27)];


C[(i + 24) + j_lda] += A_val * B_val24;

data_t B_val28 = B[k_lda + (i + 28)];


C[(i + 25) + j_lda] += A_val * B_val25;

data_t B_val29 = B[k_lda + (i + 29)];


C[(i + 26) + j_lda] += A_val * B_val26;

data_t B_val30 = B[k_lda + (i + 30)];


C[(i + 27) + j_lda] += A_val * B_val27;

data_t B_val31 = B[k_lda + (i + 31)];


C[(i + 28) + j_lda] += A_val * B_val28;

C[(i + 29) + j_lda] += A_val * B_val29;

C[(i + 30) + j_lda] += A_val * B_val30;

C[(i + 31) + j_lda] += A_val * B_val31;
```

```
}



data_t A_val1 = A[j_lda + (k + 1)];

k_lda += lda;

for (i = 0; i < lda; i += 32) {

    data_t B_val = B[k_lda + i];

    data_t B_val1 = B[k_lda + (i + 1)];

    data_t B_val2 = B[k_lda + (i + 2)];

    data_t B_val3 = B[k_lda + (i + 3)];



    C[i + j_lda] += A_val1 * B_val;

    data_t B_val4 = B[k_lda + (i + 4)];



    C[(i + 1) + j_lda] += A_val1 * B_val1;

    data_t B_val5 = B[k_lda + (i + 5)];

    C[(i + 2) + j_lda] += A_val1 * B_val2;
    data_t B_val6 = B[k_lda + (i + 6)];
    C[(i + 3) + j_lda] += A_val1 * B_val3;
    data_t B_val7 = B[k_lda + (i + 7)];
    C[(i + 4) + j_lda] += A_val1 * B_val4;
    data_t B_val8 = B[k_lda + (i + 8)];
    C[(i + 5) + j_lda] += A_val1 * B_val5;
    data_t B_val9 = B[k_lda + (i + 9)];
    C[(i + 6) + j_lda] += A_val1 * B_val6;
    data_t B_val10 = B[k_lda + (i + 10)];
    C[(i + 7) + j_lda] += A_val1 * B_val7;
    data_t B_val11 = B[k_lda + (i + 11)];
    C[(i + 8) + j_lda] += A_val1 * B_val8;
```

```
        data_t B_val12 = B[k_lda + (i + 12)];
        C[(i + 9) + j_lda] += A_val1 * B_val9;
        data_t B_val13 = B[k_lda + (i + 13)];
        C[(i + 10) + j_lda] += A_val1 * B_val10;
        data_t B_val14 = B[k_lda + (i + 14)];
        C[(i + 11) + j_lda] += A_val1 * B_val11;
        data_t B_val15 = B[k_lda + (i + 15)];
        C[(i + 12) + j_lda] += A_val1 * B_val12;
       data_t B_val16 = B[k_lda + (i + 16)];
        C[(i + 13) + j_lda] += A_val1 * B_val13;
        data_t B_val17 = B[k_lda + (i + 17)];
        C[(i + 14) + j_lda] += A_val1 * B_val14;
        data_t B_val18 = B[k_lda + (i + 18)];
        C[(i + 15) + j_lda] += A_val1 * B_val15;
        data_t B_val19 = B[k_lda + (i + 19)];
        C[(i + 16) + j_lda] += A_val1 * B_val16;
        data_t B_val20 = B[k_lda + (i + 20)];
        C[(i + 17) + j_lda] += A_val1 * B_val17;
        data_t B_val21 = B[k_lda + (i + 21)];
        C[(i + 18) + j_lda] += A_val1 * B_val18;
        data_t B_val22 = B[k_lda + (i + 22)];
        C[(i + 19) + j_lda] += A_val1 * B_val19;
        data_t B_val23 = B[k_lda + (i + 23)];
        C[(i + 20) + j_lda] += A_val1 * B_val20;
        data_t B_val24 = B[k_lda + (i + 24)];
        C[(i + 21) + j_lda] += A_val1 * B_val21;
        data_t B_val25 = B[k_lda + (i + 25)];
        C[(i + 22) + j_lda] += A_val1 * B_val22;
        data_t B_val26 = B[k_lda + (i + 26)];
        C[(i + 23) + j_lda] += A_val1 * B_val23;
        data_t B_val27 = B[k_lda + (i + 27)];
        C[(i + 24) + j_lda] += A_val1 * B_val24;
        data_t B_val28 = B[k_lda + (i + 28)];
        C[(i + 25) + j_lda] += A_val1 * B_val25;
        data_t B_val29 = B[k_lda + (i + 29)];
        C[(i + 26) + j_lda] += A_val1 * B_val26;
        data_t B_val30 = B[k_lda + (i + 30)];
        C[(i + 27) + j_lda] += A_val1 * B_val27;
        data_t B_val31 = B[k_lda + (i + 31)];

        C[(i + 28) + j_lda] += A_val1 * B_val28;
        C[(i + 29) + j_lda] += A_val1 * B_val29;
        C[(i + 30) + j_lda] += A_val1 * B_val30;
        C[(i + 31) + j_lda] += A_val1 * B_val31;

    }

  }
```

```
    }


}
```