Discussion #4:

Hints for Lab #2, Cannon's, and HW #2

Daniel Tan, Jason Lau, and Jason Cong

Lab #1: We Achieved 200 GFLOPS

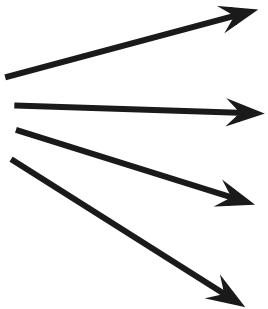
```
#pragma omp parallel for \
    schedule(static) num_threads(4)

for (int ii = 0; ii < kI; ii += 64)
for (int jj = 0; jj < kJ; jj += 1024)
for (int kk = 0; kk < kK; kk += 8)
for (int i = 0; i < 64; i++)
  for (int j = 0; j < 1024; j++) {
    float reg = c[i + ii][j + jj];
    for (int k = 0; k < 8; k++)
      reg +=
        a[i+ii][k+kk] *
        b[k+kk][j+jj];
    c[i + ii][j + jj] = reg;
  }
```

Direct Reimplementation?

Possible!

Lab #2: How to Reimplement

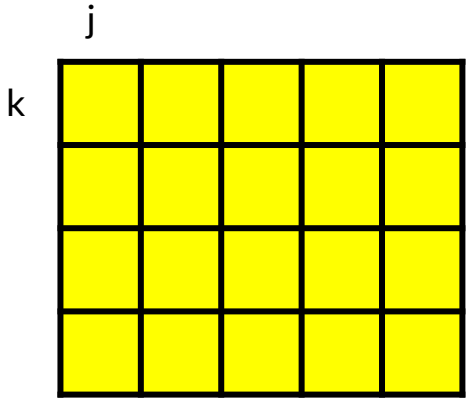
```
#pragma omp parallel for \
    schedule(static) num_threads(4)

for (int ii = 0; ii < kI; ii += 64)
for (int jj = 0; jj < kJ; jj += 1024)
for (int kk = 0; kk < kK; kk += 8)
for (int i = 0; i < 64; i++)
  for (int j = 0; j < 1024; j++) {
    float reg = c[i + ii][j + jj];
    for (int k = 0; k < 8; k++)
      reg +=
        a[i+ii][k+kk] *
        b[k+kk][j+jj];
    c[i + ii][j + jj] = reg;
  }
```

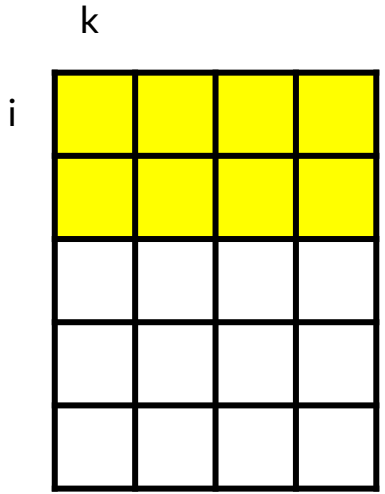


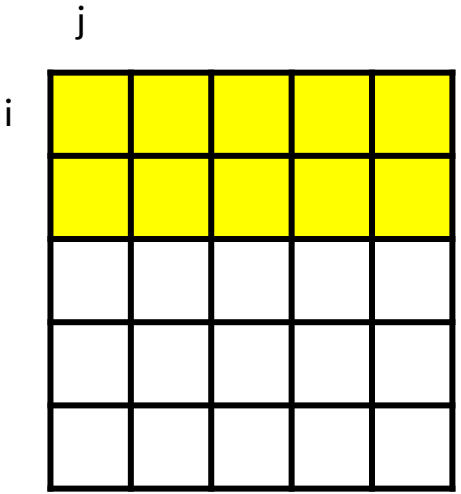
```
for (int ii = 0; ii < kI / 4; ii += 64)

for (int ii = kI / 4; ii < 2 * (kI / 4); ii += 64)

for (int ii = 2 * (kI / 4); ii < 3 * (kI / 4); ...

for (int ii = 3 * (kI / 4); ii < 4 * (kI / 4); ...
```

Lab #2: How to Reimplement

```
#pragma omp parallel for \
    schedule(static) num_threads(4)

for (int ii = 0; ii < kI; ii += 64)
for (int jj = 0; jj < kJ; jj += 1024)
for (int kk = 0; kk < kK; kk += 8)
for (int i = 0; i < 64; i++)
  for (int j = 0; j < 1024; j++) {
    float reg = c[i + ii][j + jj];
    for (int k = 0; k < 8; k++)
      reg +=
        a[i+ii][k+kk] *
        b[k+kk][j+jj];
    c[i + ii][j + jj] = reg;
  }
```







Lab #2: How to Reimplement

```
#pragma omp parallel for \
    schedule(static) num_threads(4)

for (int ii = 0; ii < kI; ii += 64)
for (int jj = 0; jj < kJ; jj += 1024)
for (int kk = 0; kk < kK; kk += 8)
for (int i = 0; i < 64; i++)
  for (int j = 0; j < 1024; j++) {
    float reg = c[i + ii][j + jj];
    for (int k = 0; k < 8; k++)
      reg +=
        a[i+ii][k+kk] *
        b[k+kk][j+jj];
    c[i + ii][j + jj] = reg;
  }
```

MPI_Scatter

MPI_Bcast

MPI_Gather

Lab #2: How to Get to [A] Range?

```
#pragma omp parallel for \
    schedule(static) num_threads(4)

for (int ii = 0; ii < kI; ii += 64)
for (int jj = 0; jj < kJ; jj += 1024)
for (int kk = 0; kk < kK; kk += 8)
for (int i = 0; i < 64; i++)
  for (int j = 0; j < 1024; j++) {
    float reg = c[i + ii][j + jj];
    for (int k = 0; k < 8; k++)
      reg +=
        a[i+ii][k+kk] *
        b[k+kk][j+jj];
    c[i + ii][j + jj] = reg;
  }
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *p1 = malloc(10*sizeof *p1);
    printf("default-aligned addr:   %p\n", (void*)p1);
    free(p1);

    int *p2 = aligned_alloc(1024, 1024*sizeof *p2);
    printf("1024-byte aligned addr: %p\n", (void*)p2);
    free(p2);
}
```

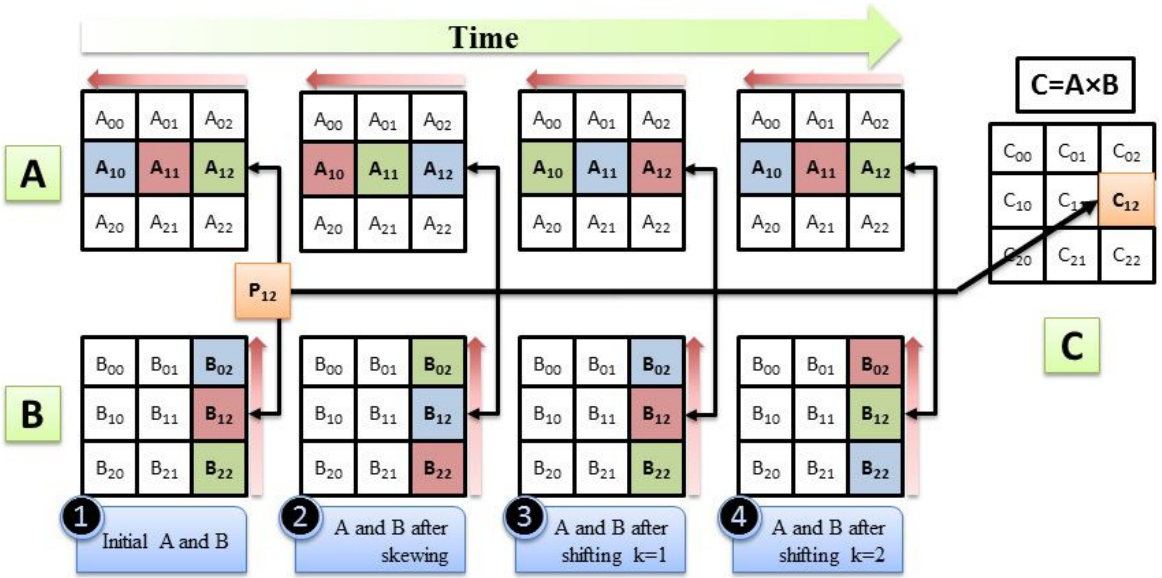
Possible output:

```
default-aligned addr:   0x1e40c20
1024-byte aligned addr: 0x1e41000
```

lab2::aligned_alloc

Cannon's Algorithm

https://www.researchgate.net/figure/Calculating-one-block-of-result-matrix-using-Cannons-algorithm_fig10_317287905



Number of blocks(p) = 9

Number of concurrent processes=9

$P_{00}, P_{01}, P_{02}, P_{10}, P_{11}, P_{12}, P_{20}, P_{21}, P_{22}$

$$C_{1,2}=A_{10}B_{02}+A_{11}B_{12}+A_{12}B_{22}$$

Cannon's Algorithm

https://en.wikipedia.org/wiki/Cannon%27s_algorithm

```
// PE(i , j)
k := (i + j) mod N;
a := a[i][k];
b := b[k][j];
c[i][j] := 0;
for (l := 0; l < N - 1; l++) {
    c[i][j] := c[i][j] + a * b;
        concurrently {
            send a to PE(i, (j + N - 1) mod N);
            send b to PE((i + N - 1) mod N, j);
        } with {
            receive a' from PE(i, (j + 1) mod N);
            receive b' from PE((i + 1) mod N, j );
        }
    a := a';
    b := b';
}
```

Memory Benefit

Shall I implement it?

🤔

Cannon's Algorithm

https://en.wikipedia.org/wiki/Cannon%27s_algorithm

```
// PE(i , j)
k := (i + j) mod N;
a := a[i][k];
b := b[k][j];
c[i][j] := 0;
for (l := 0; l < N - 1; l++) {
    c[i][j] := c[i][j] + a * b;
        concurrently {
            send a to PE(i, (j + N - 1) mod N);
            send b to PE((i + N - 1) mod N, j);
        } with {
            receive a' from PE(i, (j + 1) mod N);
            receive b' from PE((i + 1) mod N, j );
        }
    a := a';
    b := b';
}
```

5pts

Correct Implementation

Homework #2

2. Given an integer array `a[ ]` of `N` elements of value between `1` to `m` as the input, please write an efficient OpenMP function to generate the histogram `h` for array `a[ ]` such that `h[i]` is the number of elements in `a[ ]` with value `i (1 ≤ i ≤ m)`. The function header is: `void histogram(int *a, int *h)`.

- Atomic
-
-

```
# pragma omp parallel for
for(int i = 0; i < N; i++) {
    #pragma omp atomic
    h[a[i]] += 1;
}
```

Homework #2

2. Given an integer array `a[ ]` of `N` elements of value between `1` to `m` as the input, please write an efficient OpenMP function to generate the histogram `h` for array `a[ ]` such that `h[i]` is the number of elements in `a[ ]` with value `i (1 ≤ i ≤ m)`. The function header is: `void histogram(int *a, int *h)`.

-
- Lock on each of `h[i]`
-

```
#pragma omp parallel for
    for (int i = 0; i < N; i++) {
        omp_set_lock(&locks[a[i]]);
        h[a[i]]++;
        omp_unset_lock(&locks[a[i]]);
    }
```

Homework #2

2. Given an integer array `a[ ]` of `N` elements of value between `1` to `m` as the input, please write an efficient OpenMP function to generate the histogram `h` for array `a[ ]` such that `h[i]` is the number of elements in `a[ ]` with value `i (1 ≤ i ≤ m)`. The function header is: `void histogram(int *a, int *h)`.

-
-
- Local copy of `h[i]`

```
#pragma omp for nowait
for (int i = 0; i < N; ++i)
  ++hPrivate[a[i]];

#pragma omp critical
{
  for (int i = 0; i < m+1; ++i)
    h[i] += hPrivate[i]
}
```

Homework #2

3. Please write an OpenMP program to compute the numerical value of the integration of the function x/(1+x^3) between 0 and 1 using 16 threads.

- Atomic
-

```
for (float x = delta; x <= 1; x+=delta)
  #pragma omp atomic
  res += delta*(sqrt(x) / (1 + x*x*x));
```

Homework #2

3. Please write an OpenMP program to compute the numerical value of the integration of the function x/(1+x^3) between 0 and 1 using 16 threads.

-
- Local copy of sum: reduction

```
#pragma omp parallel for reduction(+: res)
for (float x = delta; x <= 1; x+=delta)
  res += delta*(sqrt(x) / (1 + x*x*x));
```

Homework #2

4. Given the following OpenMP program running on four CPU cores using four threads, assuming that the computation of function f(i, j) takes one minute on a single CPU core. Please estimate the completion time.

```
#pragma omp parallel for
for (int i = 0; i < 12; i++)
        for (int j = i; j < 12; j++)
                a[i][j] = f(i, j);
```

- **default:** 33 mins
- **dynamic, 2:** 23 mins
- **guided, 1:** 33 mins

Homework #2

5. If we want to multiply two integer matrices of 1024x1024 each, please compute the best tile size. (Cache size = 64 KB = 16,384 integers)

- sqrt(16384/3) = 73
- Easier to implement if round down to 64 :-)

Homework #2

5. Estimate the peak performance (in terms of GOP/sec) for the tiled matrix multiplication program as discussed in Lecture 5.

- Performance = operations / (time for operations + time for memory)

```
BlockSize = n / N   (N = 1024 / 64 = 16)
for i = 1 to N
   for j = 1 to N
      {read block C(i,j) into fast memory}
      for k = 1 to N
            {read block A(i,k) into fast memory}
            {read block B(k,j) into fast memory}
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
      {write block C(i,j) back to slow memory}
```

Homework #2

6. Assuming that the on-chip cache of a processor can hold 32,000 integers. If we want to sort 8,192 integers using the merge sort, please compute the computational intensity of the algorithm.

- Intensity (q) = operations (f) / memory access (m)
- **Assumption:** We count comparison as valid arithmetic operations
- **Assumption:** We use 2x merge sort (rotating array)
- **Assumption:** We write the array back to the memory after it finishes
  - You can make your own :-)

Homework #2

7. **Loop Permutation? Loop Distribution? Loop Fusion? Loop Peeling? Loop Shifting? Loop Unrolling? Loop Strip-Mining? Loop Unroll-and-Jam? Loop Tiling? Loop Parallelization? Loop Vectorization?**

```
// initialize a[][], b[][] as the 1-hop distance matrix
for (k = 0; k < N; k++) {
     for (i = 0; i < N; i++)
          for (j = 0; j < N; j++)
               a[i][j] = min(a[i][j], b[i][k] + b[k][j]);
     // copy a[][] to b[][]
}
```

Q&A