Agenda for Today

1. Q&A
2. Concurrency Bugs
3. Wrap Up
Lettercount Assignment Q&A

Can we count separate chunks of the input in each thread and combine at the end?

Please don't do all the counting without updating shared state.

How expensive are locks?

It's hard to say precisely. Usually you need a lock or you don't, regardless of the performance cost. We can design programs to avoid the need for locks sometimes, and that's worthwhile.

Is it better to have two separate lock/unlock pairs, or have one big lock/unlock?

Usually you want critical sections (the space between lock/unlock pairs) to be as small as possible.
Is there a better way to address regions than making a large switch/case block?

If you have a cell that you're working on, you can find the top left corner of its region with some fairly basic math. At that point you can iterate over the rows and columns of the region, and then convert those back to 1D indices.
Concurrency Bugs
Types of Concurrency Bugs

The reading described three types of concurrency bugs. What are they, and how do they occur?
An Example Program
int row = 0;
int color = RED;

int main() {
    pthread_t ts[3];
    for(int i=0; i<3; i++) {
        pthread_create(&ts[i], NULL, worker, NULL);
    }

    for(int i=0; i<3; i++) {
        pthread_join(threads[i], NULL);
    }
}

void* worker(void* p) {
    int my_row = row;
    row++;

    int my_color = color;
    if(color == RED) color = WHITE;
    else if(color == WHITE) color = BLUE;

    for(int col=0; col<4; col++) {
        put_color(my_row, col, my_color);
    }
    return NULL;
}
int row = 0;
int color = RED;

void* worker(void* p) {
  int my_row = row;
  row++;

  int my_color = color;
  if(color == RED) color = WHITE;
  else if(color == WHITE) color = BLUE;

  for(int col=0; col<4; col++) {
    put_color(my_row, col, my_color);
  }
  return NULL;
}

int main() {
  pthread_t ts[3];
  for(int i=0; i<3; i++) {
    pthread_create(&ts[i], NULL, worker, NULL);
  }

  for(int i=0; i<3; i++) {
    pthread_join(threads[i], NULL);
  }
}
A Better Flag-Drawing Program

int color = RED;

int main() {
    pthread_t ts[3];
    int rows[3];
    for(int i=0; i<3; i++) {
        rows[i] = i;
        pthread_create(&ts[i], NULL, worker, &rows[i]);
    }

    for(int i=0; i<3; i++) {
        pthread_join(threads[i], NULL);
    }
}

void* worker(void* arg) {
    int my_row = *(int*)arg;

    int my_color = color;
    if(color == RED) color = WHITE;
    else if(color == WHITE) color = BLUE;
    for(int col=0, col<4, col++) {
        put_color(my_row, col, my_color);
    }
    return NULL;
}
int color = RED;

int main() {
    pthread_t ts[3];
    int rows[3];
    for(int i=0; i<3; i++) {
        rows[i] = i;
        pthread_create(&ts[i], NULL, worker, &rows[i]);
    }

    for(int i=0; i<3; i++) {
        pthread_join(threads[i], NULL);
    }
}

int main() {
    pthread_t ts[3];
    int rows[3];
    for(int i=0; i<3; i++) {
        rows[i] = i;
        pthread_create(&ts[i], NULL, worker, &rows[i]);
    }

    for(int i=0; i<3; i++) {
        pthread_join(threads[i], NULL);
    }
}

void* worker(void* arg) {
    int row = *(int*)arg;

    int my_color = color;
    if(color == RED) color = WHITE;
    else if(color == WHITE) color = BLUE;

    for(int col=0, col<4, col++) {
        put_color(my_row, col, my_color);
    }
    return NULL;
}
A Better Flag-Drawing Program

```c
int color = RED;
pthread_mutex_t color_lock = PTHREAD_MUTEX_INITIALIZER;

int main() {
    pthread_t ts[3];
    int rows[3];
    for (int i = 0; i < 3; i++) {
        rows[i] = i;
        pthread_create(&ts[i], NULL, worker, &rows[i]);
    }

    for (int i = 0; i < 3; i++) {
        pthread_join(threads[i], NULL);
    }
}

void* worker(void* arg) {
    int row = *(int*)arg;
    pthread_mutex_lock(&color_lock);
    int my_color = color;
    if (color == RED) color = WHITE;
    else if (color == WHITE) color = BLUE;
    pthread_mutex_unlock(&color_lock);
    for (int col = 0, col < 4, col++) {
        put_color(my_row, col, my_color);
    }
    return NULL;
}
```
What went wrong?

int color = RED;
pthread_mutex_t color_lock = PTHREAD_MUTEX_INITIALIZER;

int main() {
    pthread_t ts[3];
    int rows[3];
    for(int i=0; i<3; i++) {
        rows[i] = i;
        pthread_create(&ts[i], NULL, worker, &rows[i]);
    }

    for(int i=0; i<3; i++) {
        pthread_join(threads[i], NULL);
    }
}

void* worker(void* arg) {
    int row = *(int*)arg;
    pthread_mutex_lock(&color_lock);
    int my_color = color;
    if(color == RED) color = WHITE;
    else if(color == WHITE) color = BLUE;
    pthread_mutex_unlock(&color_lock);
    for(int col=0; col<4; col++) {
        put_color(my_row, col, my_color);
    }
    return NULL;
}
# One Possible Fix

```c
int main() {
    pthread_t ts[3];
    int rows[3];
    for(int i=0; i<3; i++) {
        rows[i] = i;
        pthread_create(&ts[i], NULL, worker, &rows[i]);
    }
    for(int i=0; i<3; i++) {
        pthread_join(threads[i], NULL);
    }
}

void* worker(void* arg) {
    int row = *(int*)arg;
    int my_color;
    if (row == 0) my_color = RED;
    else if (row == 1) my_color = WHITE;
    else my_color = BLUE;
    for(int col=0; col<4; col++) {
        put_color(my_row, col, my_color);
    }
    return NULL;
}
```
Deadlock
Deadlock

What are the four causes of deadlock? How can we prevent each cause?

No Preemption - There's no way to force a thread to release a lock. Hard to get rid of this cause.

Mutual Exclusion - A thread can have exclusive control of a resource. To get rid of this, we'd need to get rid of locks.

Circular Wait - Thread 2 holds a lock that thread 1 needs, and thread 1 holds a lock that thread 2 needs. The fix is to acquire locks in some fixed order.

Hold and Wait - A thread can hold a lock while waiting for another. We could fix this by trying to acquire all locks at once. Condition variables also can resolve this in some cases.
Wrap Up
Reminders

The End of Daylight Savings Time
   Set your clocks back an hour on Sunday morning!

Lab
   Sudoku lab is due on Wednesday, November 10th (next week)

Reading
   Read about distributed systems for Monday

My Office Hours
   Monday 10:30am–noon, Tuesday 2:30–4:30pm, Friday 10:30am–noon

Mentor Office Hours
   Sunday 1–4pm, Tuesday 8–9pm, Wednesday 8–9pm