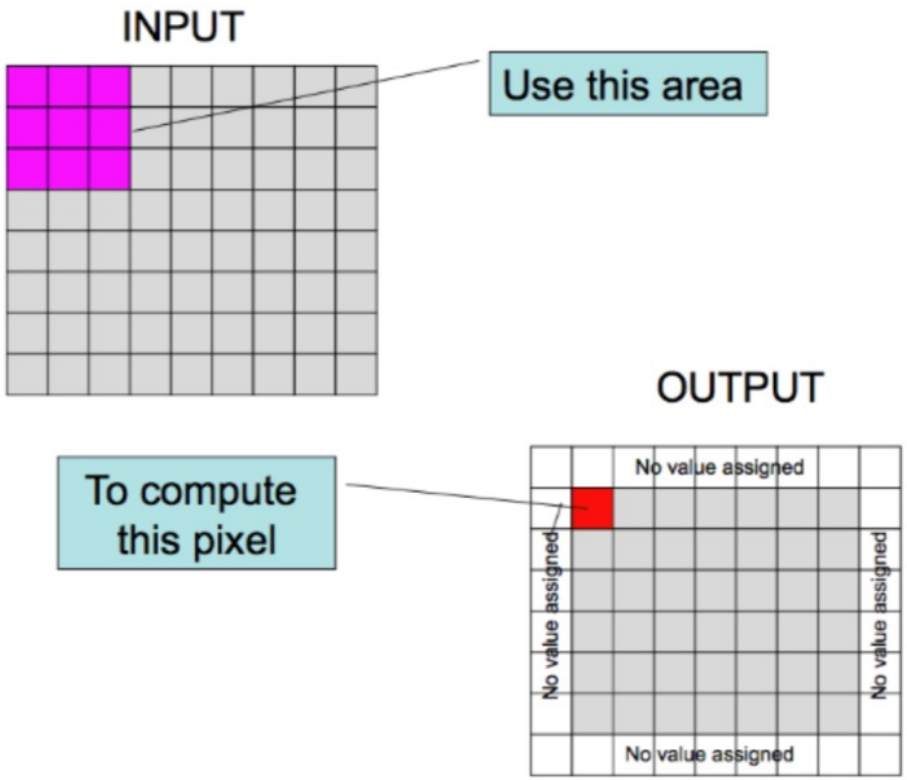
Introduction to Performance Lab

- Filter Operation
  - Convolution
    - Elements of the filter matrix (3X3) are multiplied by image matrix (bmp format) to compute a new value of image
  - Filter files
    - *.filter : size(3), div, values(3x3 matrix)

- Goal
  - Optimizing code performance (Getting the highest resulting score)
  - You are free to modify *.h, *.cpp, and Makefile



How to Optimize Program Performance

- Invoking GCC with option –O1 or higher
- Selecting an appropriate algorithm and data structures or type
- Eliminating unnecessary works
  - Eliminating loop inefficiencies (code motion)
  - Eliminating unnecessary function calls
- Executing multiple instructions simultaneously (loop unroll)

Specifying the Optimization Level

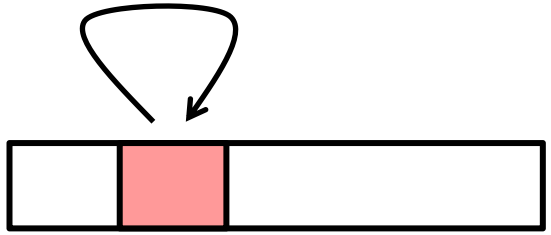
- Invoking GCC with option –O1 or higher will cause it to apply more extensive optimization
  - Level –O2 has become the accepted standard for most software projects that use GCC

Terminal 1 | Makefile | FilterMain.cpp

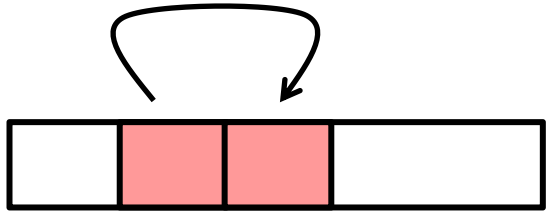
```
1  ##
2  ##
3
4  CXX =g++
5  ##
6  ## Use our standard compiler flags for the course...
7  ## You can try changing these flags to improve performance.
8  ##
9  CXXFLAGS= -g -O2 -fno-omit-frame-pointer -Wall
10
```

Appropriate Algorithm & Data Structure

- Locality
  Programs tend to access the same set of memory locations repetitively over a short period of time
  - Temporal locality: loop
    - Recently referenced items are likely to be referenced again in the near future



  - Spatial locality: array
    - Items with nearby addresses tend to be referenced close together in time



Appropriate Algorithm & Data Structure

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

- Which function has good locality with respect to array a?

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

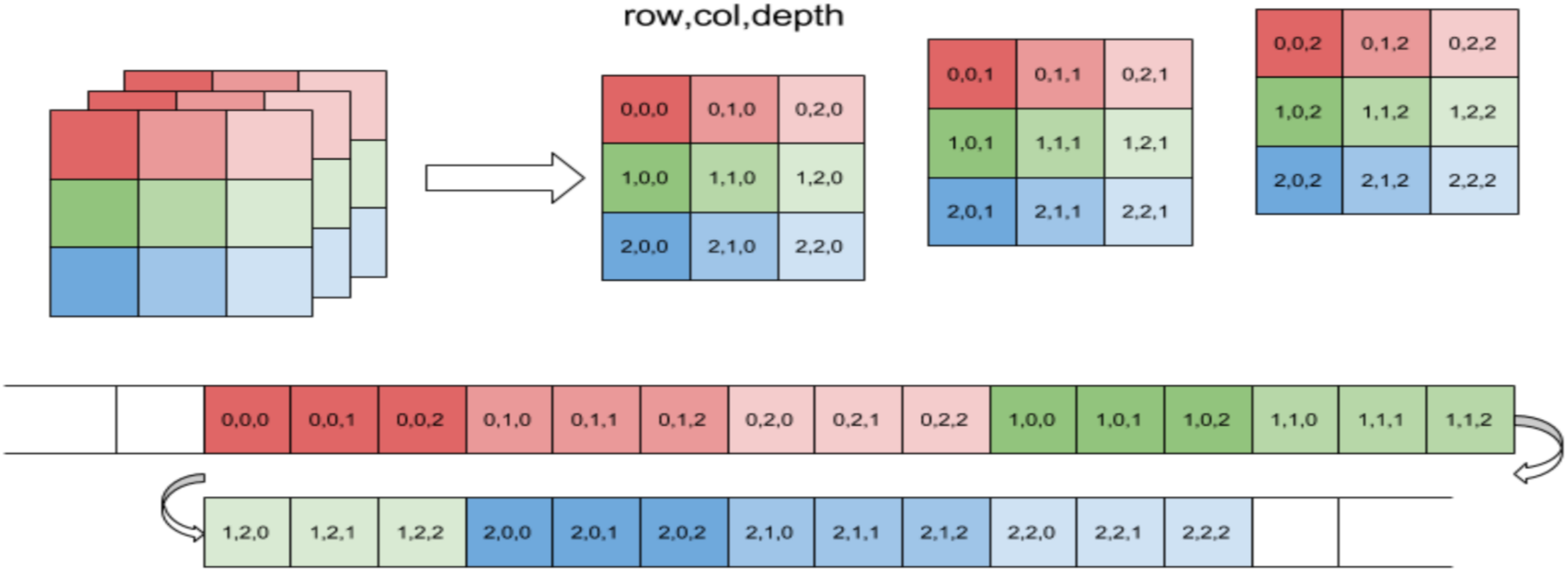
```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            sum += a[i][j];
    return sum;
}
```

			0,0	0,1	0,2	1,0	1,1	1,2	2,0	2,1	2,2

Appropriate Algorithm & Data Structure

- Memory Layout of a 3D Array



Eliminating Unnecessary Work

- Move code out of the loop

```
void lower(char *s)
{
  size_t i;
  for (i = 0; i < strlen(s); i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
      s[i] -= ('A' - 'a');
}
```

```
void lower(char *s)
{
  size_t i;
  size_t len = strlen(s);
  for (i = 0; i < len; i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
      s[i] -= ('A' - 'a');
}
```

```
void set_row(double *a, double *b,
   long i, long n)
{
    long j;
    for (j = 0; j < n; j++)
        a[n*i+j] = b[j];
}
```

```
    long j;
    int ni = n*i;
    for (j = 0; j < n; j++)
        a[ni+j] = b[j];
```

Loop Unrolling

- Executing multiple instructions simultaneously

```
void combine4(vec_ptr v, data_t *dest)
{
  long i;
  long length = vec_length(v);
  data_t *d = get_vec_start(v);
  data_t t = IDENT;
  for (i = 0; i < length; i++)
    t = t OP d[i];
  *dest = t;
}
```

Original loop:
do 1 calc. in 1 iteration

Loop Unrolling

- Executing multiple instructions simultaneously

```
void unroll2a_combine(vec_ptr v, data_t *dest){
    long length = vec_length(v);
    long limit = length-1;
    data_t *d = get_vec_start(v);
    data_t x = IDENT;
    long i;
    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
          x = (x OP d[i]) OP d[i+1];
    }
    /* Finish any remaining elements */
    for (; i < length; i++) {
          x = x OP d[i];
    }
    *dest = x;
}
```

Unroll by 2x1:
do 2 calc. in 1 iteration

Loop Unrolling

- Executing multiple instructions simultaneously

```
void unroll2a_combine(vec_ptr v, data_t *dest){
    long length = vec_length(v);
    long limit = length-1;
    data_t *d = get_vec_start(v);
    data_t x0 = IDENT;
    data_t x1 = IDENT;
    long i;
    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
       x0 = x0 OP d[i];
       x1 = x1 OP d[i+1];
    }
    /* Finish any remaining elements */
    for (; i < length; i++) {
           x0 = x0 OP d[i];
    }
    *dest = x0 OP x1;
}
```

Unroll by 2x2
to reduce sequential dependency