Compiler Optimization

- Compiler can help in reducing cache miss
- ***Merging Arrays***: Improve spatial locality by single array of compound elements vs. 2 arrays.
- ***Loop Interchange***: Change nesting of loops to access data in the order stored in memory.
- ***Loop Fusion***: Combine 2 or more independent loops that have the same looping and some variables overlap.
- ***Blocking***: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows

Merging Arrays

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of stuctures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Reduces conflict between val and key and reduces compulsory misses if they are accessed in the same pattern

Interchanging Loops

° Assume row-major matrix allocation

```
/* Before */
 for (j = 0; j < 100; j = j+1)
     for (i = 0; i < 5000; i = i+1)
          x[i][j] = 2 * x[i][j];

/* After */
 for (i = 0; i < 5000; i = i+1)
     for (j = 0; j < 100; j = j+1)
          x[i][j] = 2 * x[i][j];
```

Sequential accesses instead of striding through memory every 100 words in this case improves spatial locality (reduces compulsory misses)

Loop Fusion

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
       a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
       d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {    a[i][j] = 1/b[i][j] * c[i][j];
       d[i][j] = a[i][j] + c[i][j];}
```

Blocking

```
for (i = 0; i < 12; i = i+1)
  for (j = 0; j < 12; j = j+1)
       {r = 0;
        for (k = 0; k < 12; k = k+1){
              r = r + y[i][k]*z[k][j];};
        x[i][j] = r;
       };
/* After Blocking */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B,N); j = j+1)
       {r = 0;
        for (k = kk; k < min(kk+B,N); k = k+1) {
              r = r + y[i][k]*z[k][j];};
        x[i][j] = x[i][j] + r;
       };
```