Modern Computer Architecture

Lecture 10
Multi-Core: pthreads, OpenMP
Segmented Scan
• Multi-Core:
  – pthreads
  – OpenMP
• Segmented Scan
Multi- and Many-Core Systems I
- Tightly Coupled
- Shared Memory
- Multi-Threading
- Example: pthreads, OpenMP

Multi- and Many-Core Systems II
- Loosely Coupled
- No Shared Memory
- Message Passing
- Example: Unix Processes, MPI, Cloud

Not part of this Lecture. See, e.g., Parallel Programming.
Thread
- represents independent control flow

In hardware:
- Program Counter
- Registers
- Thread Local Storage (TLS)
  - `__thread` int variable;
  - `__declspec(thread)` int variable; (Visual Studio)
- State for communication (signals, semaphores, etc.)

Process
- represents a program managed by OS:
  - memory map
  - heap
  - table of open files
  - table of open network sockets
  - shared libraries
  - environment variables
  - security credentials (user name etc.)

traditionally: each process had one thread
today: multiple parallel threads
Unix Processes

- Creation of processes is expensive
- A lot of state has to be duplicated
- Happens e.g. in web-server
- Cumbersome for different processes to communicate
- Slow to check security credentials at inter-process communication
- many programs create threads as “light-weight processes”
- problem: many library functions were not designed to run in multi-threaded environment (e.g. use global variables)
- property: “thread-safe”, usually pass global state in additional parameters

```c
[one process]
ppid = fork ();
if (ppid < 0) {
    my_handle_error ();
} else if (ppid == 0) {
    [child process]
    my_child_function ();
} else {
    [parent process]
    my_parent_function ();
}
```
- pthread_create (thread, attr, start_routine, arg)

- pthread_exit (status)

- pthread_cancel (thread)

- pthread_attr_init (attr)

- pthread_attr_destroy (attr)
Thread Attributes

- Detached or joinable state
- Scheduling inheritance
- Scheduling policy
- Scheduling parameters
- Scheduling contention scope
- Stack size
- Stack address
- Stack guard (overflow) size

Linux allows to use different scheduling policies:
- maximum throughput (server)
- minimum latency (desktop PC)
- real-time programs
Scheduling Strategies

- First In – First Out (FIFO), First-Come First-Served (FCFS)
- Shortest-Job-Next (SJN), Shortest Job First (SJF), Shortest Processing Time (SPT)
- Shortest-Remaining-Time (SRT) oder Shortest-Remaining-Processing-Time (SRPT).
- Earliest Due Date (EDD), Earliest Deadline First (EDF)
- Priority Scheduling
  - Rate Monotonic Scheduling (RMS)
  - Deadline Monotonic Scheduling (DMS)
  - Multilevel Feedback Queue Scheduling, Shortest-Elapsed-Time (SET)
- Round Robin, Time Slice, Weighted Round Robin (WRR)
```c
#include <pthread.h>
#include <stdio.h>

#define NUM_THREADS 5

void *
PrintHello(void *threadid)
{
    long tid; tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc; long t;
    for(t=0; t<NUM_THREADS; t++) {
        printf("In main: creating thread %ld\n", t);
        rc = pthread_create(&threads[t],
                            NULL, PrintHello,
                            (void *)t);
        if (rc) {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    /* Last thing that main() should do */
    pthread_exit(NULL);
}
```
Thread Joining and Detaching

- `pthread_join (threadid, status)`
  Waits for child processes to exit.

- `pthread_detach (threadid)`
  Note: it’s not possible to re-attach a detached thread.

For newly created threads:
- `pthread_attr_setdetachstate (attr, detachstate)`
- `pthread_attr_getdetachstate (attr, detachstate)`
• `pthread_attr_getstacksize (attr, stacksize)`

• `pthread_attr_setstacksize (attr, stacksize)`

• `pthread_attr_getstackaddr (attr, stackaddr)`

• `pthread_attr_setstackaddr (attr, stackaddr)`
Mutex Variables

- `pthread_mutex_init (mutex,attr)`
- `pthread_mutex_destroy (mutex)`
- `pthread_mutex_lock (mutex)`
- `pthread_mutex_trylock (mutex)`
- `pthread_mutex_unlock (mutex)`
Debugging Multi-Threaded Programs

GDB:

• info threads
  
  3 process 35 thread 27  0x34e5 in sigpause ()
  2 process 35 thread 23  0x34e5 in sigpause ()
  * 1 process 35 thread 13  main (argc=1, argv=0x7fffffffe8)
    at threadtest.c:68

• break filename:line thread number

• Challenge: reproduce sequence of events that lead to program failure
• “Parallel Chaos”
• Fast Thread Creation and Synchronization
• Variable number of threads

```c
#define N 100000
int main(int argc, char *argv[])
{
    int i, a[N];

    #pragma omp parallel for
    for (i = 0; i < N; i++) {
        a[i]= 2 * i;
    }

    return 0;
}
```
#include <omp.h>

int main(void)
{
    int id,i;

    omp_set_num_threads(4);

    #pragma omp parallel for private(id)
    for (i = 0; i < 4; ++i) {
        id = omp_get_thread_num();

        printf("Hello World from thread %d\n", id);
    }

    #pragma omp barrier
    if (id == 0) {
        printf("There are %d threads\n", omp_get_num_threads());
    }

    return 0;
}
$ gcc-4.2 -fopenmp -o pg pg.c
$ ./pg

Hello World from thread 3
Hello World from thread 0
Hello World from thread 1
Hello World from thread 2

There are 4 threads

#pragma omp parallel
{
  ...
}

arbitrary
order
barrier
```c
#pragma omp parallel
{
    #pragma omp for private(j)
    for(i=0; i<100; i++)
    {
        for(j=0; j<10; j++)
        {
            T[omp_get_thread_num()]+=1;
        
        #pragma omp critical
            NumOfIters++;
    
        #pragma omp end critical
    }
}
}```
```c
#pragma omp parallel
{
    #pragma omp for private(j) reduction(+: NumOfIters)
    for(i=0; i<100; i++)
    {
        for(j=0; j<10; j++)
        {
            T[omp_get_thread_num()]++;
            NumOfIters++;  
        }
    }
}
```
• Part 2: Segmented Scan
• http://www.me.berkeley.edu/ME290R/S2009/lectures/lec15.PDF
• Shubhabrata Sengupta, Mark Harris, Yao Zhang, John D. Owens: Scan primitives for GPU computing. Graphics Hardware 2007: 97-106
Applications of Segmented Scan

- Segmented Scan is inherently a sequential operation
- It can be implemented efficiently even on highly parallel architectures (GPU, Connection Machine, Cray, etc.)
- Segment flags can be used to store irregular structures, like graphs and lists.
- Parallelization of Algorithms:
  - quick-sort, radix-sort
  - graph: e.g. minimum spanning-tree
  - sparse matrices
Element-wise Vector Operations

- **Prerequisites:**
  - A tuple of input vectors of the same length (typically two)
  - Example: \( \mathbf{a} = [1 \ 2 \ 3 \ 4] \), \( \mathbf{b} = [2 \ 4 \ 6 \ 8] \)
  - An operator acting on the element types of the vectors
  - Example: +

- **Element-wise operation:** .
  - Semantics: \( \cdot^\circ [v]_i = (\cdot [v])_i \)
  - With two input operands: \( a_i \cdot^\circ b_i := (a\cdot b)_i \)
  - Example: .+
  - Example: \( \mathbf{a} .+ \mathbf{b} = [1+2 \ 2+4 \ 3+6 \ 4+8] = [3 \ 6 \ 9 \ 12] \)
Memory Operations

- Traditional vector load and store has scalar address
- Semantics Element-wise load/store of consecutive addresses

Example: `_mm_load_ps (addr)`

```
PE_1  ---  bank_1  addr+0
PE_2  ---  bank_2  addr+1
    ...          ...
PE_N  ---  bank_M  addr+M-1
```
• Provide a vector address
• Potential slowdown:
  – Bank-conflict: two PEs may want to access data on same bank
  – Interconnection network may not permit arbitrary permutation
• Mask by boolean vector (e.g. select-permute)
A (data vector) = \[ a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a_6 \ a_7 \]

I (index vector) = \[ 2 \ 5 \ 4 \ 3 \ 1 \ 6 \ 0 \ 7 \]

\[
C \leftarrow \text{permute}(A, I) = \begin{bmatrix} a_6 & a_4 & a_0 & a_3 & a_2 & a_1 & a_5 & a_7 \end{bmatrix}
\]
Communications Operations

- Broadcast scalar into vector
- Select scalar from vector
- Permute vector
- Shift elements of vector
  - By arbitrary number of elements (crossbar switch)
  - By power of two (skip network)
Scan Operation

Prerequisites:

- A vector \([a_i]\) of type \([T]\)
- A binary operator \(\circ\)
- \((T, \circ)\) form a semi-group with neutral element \(0\circ\)
  - \(\circ\) is associative: i.e. \((a \circ b) \circ c = a \circ (b \circ c)\)
  - \(a \circ 0\circ = 0\circ \circ a = a\)

- Examples: addition, multiplication, max, min, AND, OR, XOR, etc.

Scan / inclusive scan:

\[
\text{scan} : \text{op} \times [T] \rightarrow [T] \\
[b_i] = \text{scan} \circ [a_i] \\
b_{-1} = 0\circ \\
b_i = b_{i-1} \circ a_i
\]

Exclusive scan / prescan:

\[
\text{prescan} : \text{op} \times [T] \rightarrow [T] \\
[b_i] = \text{prescan} \circ [a_i] \\
b_0 = 0\circ \\
b_i = b_{i-1} \circ a_{i-1}
\]
• Inclusive Scan:
• \( \text{scan} + [x_0, x_1, x_2, x_3] = [x_0, x_0+x_1, x_0+x_1+x_2, x_0+x_1+x_2+x_3] \)

\[
\begin{array}{c|cccccccc}
  & 5 & 1 & 3 & 4 & 3 & 9 & 2 & 6 \\
\hline
\text{A} = [ & 5 & 1 & 3 & 4 & 3 & 9 & 2 & 6 ] \\
\text{scan + A} = [ & 5 & 6 & 9 & 13 & 16 & 25 & 27 & 33 ] \\
\end{array}
\]

• Exclusive Scan:

\[
\begin{array}{c|cccccccc}
  & 5 & 1 & 3 & 4 & 3 & 9 & 2 & 6 \\
\hline
\text{A} = [ & 5 & 1 & 3 & 4 & 3 & 9 & 2 & 6 ] \\
\text{prescan + A} = [ & 0 & 5 & 6 & 9 & 13 & 16 & 25 & 27 ] \\
\end{array}
\]
Up-Sweep

\[
\text{for } d = 0 \text{ to } \log_2 n-1 \text{ do } \\
\text{ for all } k = 0 \text{ to } n-1 \text{ by } 2^{d+1} \text{ in parallel do } \\
\quad x[k+2^{d+1}-1] \leftarrow x[k+2^d-1] + x[k+2^{d+1}-1]
\]

Down-Sweep

\[
\text{x}[n-1] \leftarrow 0 \\
\text{for } d = \log_2 n-1 \text{ down to } 0 \text{ do } \\
\text{ for all } k = 0 \text{ to } n-1 \text{ by } 2^{d+1} \text{ in parallel do } \\
\quad t \leftarrow x[k+2^d-1] \\
\quad x[k+2^d-1] \leftarrow x[k+2^{d+1}-1] \\
\quad x[k+2^{d+1}-1] \leftarrow t + x[k+2^{d+1}-1]
\]
Scan and Segmented Scan

- **Scan-Operation:**
  - Apply operation sequentially to elements of a vector
  - Result is a vector of element-wise intermediate results
  - Example: \( A \xrightarrow{\text{ scan}} \text{scan}(A,+)) \)

<table>
<thead>
<tr>
<th>A=</th>
<th>5</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>3</th>
<th>9</th>
<th>2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>16</td>
<td>25</td>
<td>27</td>
</tr>
</tbody>
</table>

- **Segmented Scan:**
  - new parameter: segment descriptor/flags
  - “restart” operation at beginning of each segment

<table>
<thead>
<tr>
<th>B=</th>
<th>5</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>3</th>
<th>9</th>
<th>2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sb=</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>7</td>
<td>10</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Up-Sweep

for $d = 0$ to $\log_2 n - 1$ do
  for all $k = 0$ to $n-1$ by $2^{d+1}$ in parallel do
    if $f[k+2^{d+1}-1]$ is not set then
      $x[k+2^{d+1}-1] \leftarrow x[k+2^d-1] + x[k+2^{d+1}-1]$
      $f[k+2^{d+1}-1] \leftarrow f[k+2^d-1] | f[k+2^{d+1}-1]$
  
Down-Sweep

$x[n-1] \leftarrow 0$

for $d = \log_2 n - 1$ down to 0 do
  for all $k = 0$ to $n-1$ by $2^{d+1}$ in parallel do
    $t \leftarrow x[k+2^d-1]$
    $x[k+2^d-1] \leftarrow x[k+2^{d+1}-1]$
    if $f[k+2^d]$ is set then
      $x[k+2^{d+1}-1] \leftarrow 0$
    else if $f[k+2^d-1]$ is set then
      $x[k+2^{d+1}-1] \leftarrow t$
    else
      $x[k+2^{d+1}-1] \leftarrow t + x[k+2^{d+1}-1]$
    Unset flag $f[k+2^d-1]$
- Example scan(+[4 2 1] [3 0 2] [1 5]))
- \( V = [4 2 1 3 0 2 1 5] \)
- \( S = [T F F T F F T F] \)

<table>
<thead>
<tr>
<th>input</th>
<th>T</th>
<th>4</th>
<th>F</th>
<th>2</th>
<th>F</th>
<th>1</th>
<th>T</th>
<th>3</th>
<th>F</th>
<th>0</th>
<th>F</th>
<th>2</th>
<th>T</th>
<th>1</th>
<th>F</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>up-</td>
<td>T</td>
<td>4</td>
<td>T</td>
<td>6</td>
<td>F</td>
<td>1</td>
<td>T</td>
<td>3</td>
<td>F</td>
<td>0</td>
<td>F</td>
<td>2</td>
<td>T</td>
<td>1</td>
<td>T</td>
<td>6</td>
</tr>
<tr>
<td>sweep</td>
<td>T</td>
<td>4</td>
<td>T</td>
<td>6</td>
<td>F</td>
<td>1</td>
<td>T</td>
<td>3</td>
<td>F</td>
<td>0</td>
<td>F</td>
<td>2</td>
<td>T</td>
<td>1</td>
<td>T</td>
<td>6</td>
</tr>
<tr>
<td>down-</td>
<td>T</td>
<td>4</td>
<td>T</td>
<td>6</td>
<td>F</td>
<td>1</td>
<td>T</td>
<td>3</td>
<td>F</td>
<td>0</td>
<td>F</td>
<td>2</td>
<td>T</td>
<td>1</td>
<td>T</td>
<td>6</td>
</tr>
<tr>
<td>sweep</td>
<td>T</td>
<td>4</td>
<td>T</td>
<td>6</td>
<td>F</td>
<td>1</td>
<td>T</td>
<td>3</td>
<td>F</td>
<td>0</td>
<td>F</td>
<td>2</td>
<td>T</td>
<td>1</td>
<td>T</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>4</td>
<td>F</td>
<td>0</td>
<td>F</td>
<td>1</td>
<td>T</td>
<td>6</td>
<td>F</td>
<td>0</td>
<td>T</td>
<td>3</td>
<td>T</td>
<td>1</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>0</td>
<td>T</td>
<td>4</td>
<td>T</td>
<td>6</td>
<td>F</td>
<td>0</td>
<td>T</td>
<td>3</td>
<td>T</td>
<td>3</td>
<td>F</td>
<td>0</td>
<td>T</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Segmented Scan

Up-Sweep

Down-Sweep
Hardware Implementation: Connection Machine

\[ D_1 = \text{Op}(Q_1 + BAQ_2) + \text{Op}(AB + AQ_1 + BQ_1) \]
\[ D_2 = \text{Op}(Q_2 + ABQ_1) \]
\[ D_3 = \text{Op}(BQ_1 + AQ_2) + \text{Op}(A \oplus B \oplus Q_1) \]
Enumerate

- Assign each 'true' element in vector a unique index
- Usage: for scattered load/store operation
- Enumerate([t f f t f t t]) = [0 1 1 1 2 2 3]

Distribute / Copy

- Segmented version of broadcast
- Distribute([a b c] [d e]) = [a a a] [d d]
### Simple Operations

<table>
<thead>
<tr>
<th>Flag</th>
<th>=</th>
<th>[T  F  F  T  F  T  T  T  F]</th>
</tr>
</thead>
<tbody>
<tr>
<td>enumerate(Flag)</td>
<td>=</td>
<td>[0 1 1 1 2 2 3 4]</td>
</tr>
</tbody>
</table>

| A               | = | [5 1 3 4 3 9 2 6]          |
| copy(A)         | = | [5 5 5 5 5 5 5 5]          |

| B               | = | [1 1 2 1 1 2 1 1]          |
| +--distribute(B) | = | [10 10 10 10 10 10 10 10]  |
Scan Vector Primitives

Scalar Instructions:

Arithmetic and Logical Instructions:
+ , − , and , or , = , < , ...

Conditional Instruction:
cond-jump

Indirect-Access Instructions:
move-scalar, move-vector

Vector Instructions:

Elementwise Instructions:
p+ , p− , p-and , p-or , p= , p< , p-select , ...

Permutation Instructions:
permute, select-permute

Scan Instructions:
+-scan, max-scan, min-scan, or-scan, and-scan

Vector-Scalar Instructions:
insert, extract, distribute, length
<table>
<thead>
<tr>
<th>Operation</th>
<th>Other Names</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>index</td>
<td>PARALATION LISP [96]</td>
</tr>
<tr>
<td></td>
<td>iota</td>
<td>APL [59]</td>
</tr>
<tr>
<td>⊕-reduce</td>
<td>reduce</td>
<td>APL, COMMON LISP [107]</td>
</tr>
<tr>
<td></td>
<td>vref</td>
<td>PARALATION LISP</td>
</tr>
<tr>
<td>⊕-distribute</td>
<td>catenate</td>
<td>APL</td>
</tr>
<tr>
<td></td>
<td>concatenate</td>
<td>COMMON LISP</td>
</tr>
<tr>
<td>append</td>
<td>pack</td>
<td>Schwartz [101]</td>
</tr>
<tr>
<td>pack</td>
<td>compress</td>
<td>APL</td>
</tr>
<tr>
<td></td>
<td>irregular compression</td>
<td>Batcher [12]</td>
</tr>
<tr>
<td>split</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flag-merge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inverse-permute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>enumerate</td>
<td>enumerate</td>
<td>Christman [31]</td>
</tr>
<tr>
<td>max-index</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```plaintext
define split(A, Flags){
    I-down ← enumerate(not(Flags));
    I-up ← n-back enumerate(Flags) − 1;
    Index ← if Flags then I-up else I-down;
    permute(A, Index)}
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Flags</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>I-down</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>I-up</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Index</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>permute(A, Index)</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
\[ V = [v_1, v_2, v_3] \]
\[ A = [4, 1, 3] \]
\[ \text{Hpointers} \leftarrow \text{+-scan}(A) = [0, 4, 5] \]
\[ \text{Segment-flag} = [1, 0, 0, 0, 1, 1, 0, 0] \]
\[ \text{distribute}(V, \text{Hpointers}) = [v_1, v_1, v_1, v_1, v_2, v_3, v_3, v_3] \]
Example: Quicksort

1. Check if vector is already sorted
2. Pick a pivot within each segment
3. Compare each segment against pivot
4. Split based on the result of the comparison
5. Insert segment flags to separate split values
6. Continue with step 1.
### Example Quicksort

<table>
<thead>
<tr>
<th>Key</th>
<th>=</th>
<th>[24.6 48.1 5.8 3.1 37.8 9.5 48.1 5.8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment-Flags</td>
<td>=</td>
<td>[T F F F F F F F F F]</td>
</tr>
<tr>
<td>Pivots</td>
<td>=</td>
<td>[24.6 24.6 24.6 24.6 24.6 24.6 24.6 24.6]</td>
</tr>
<tr>
<td>F</td>
<td>=</td>
<td>[= &gt; &lt; &lt; &gt; &lt; &gt; &lt;]</td>
</tr>
<tr>
<td>Key $\leftarrow$ split(Key, F)</td>
<td>=</td>
<td>[5.8 3.1 9.5 5.8 24.6 48.1 37.8 48.1]</td>
</tr>
<tr>
<td>Segment-Flags</td>
<td>=</td>
<td>[T F F F T T F F F]</td>
</tr>
<tr>
<td>Pivots</td>
<td>=</td>
<td>[5.8 5.8 5.8 5.8 24.6 48.1 48.1 48.1]</td>
</tr>
<tr>
<td>F</td>
<td>=</td>
<td>[= &lt; &gt; = = = &lt; =]</td>
</tr>
<tr>
<td>Key $\leftarrow$ split(Key, F)</td>
<td>=</td>
<td>[3.1 5.8 5.8 9.5 24.6 37.8 48.1 48.1]</td>
</tr>
<tr>
<td>Segment-Flags</td>
<td>=</td>
<td>[T T F T T T T F]</td>
</tr>
</tbody>
</table>
Graph Representation

Index = [0 1 2 3 4 5 6 7 8 9 10 11]
vertex = [1 2 3 4 5 6 7 8 9]
segment-descriptor = [1 3 3 2 3]
cross-pointers = [1 0 4 9 2 7 10 5 11 3 6 8]
weights = [w_1 w_1 w_2 w_3 w_2 w_4 w_5 w_4 w_6 w_3 w_5 w_6]