CUDA Parallel Prefix Sum (1A)

-
-

Copyright (c) 2013 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Naive Parallel Scan

for d := 1 **to** $\log_2 n$ **do**

    forall k **in parallel do**

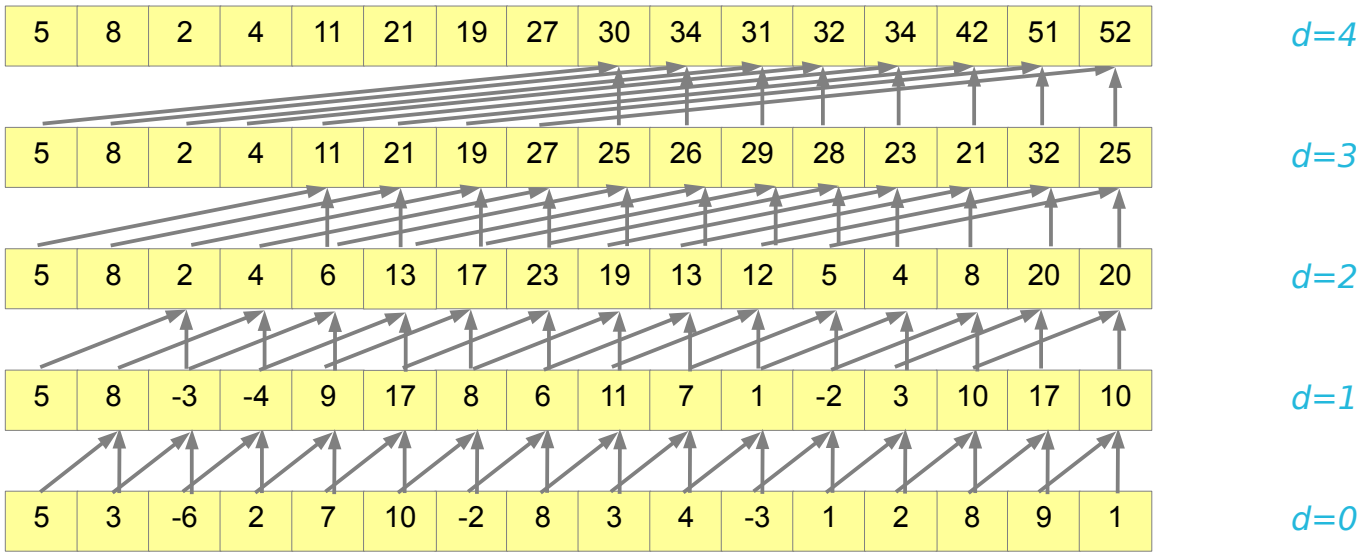
        if $k \geq 2^d$ **then** $x[k] := x[k - 2^{d-1}] + x[k]$

Parallel Prefix Sum

for d := 1 **to** $\log_2 n$ **do**

forall k **in parallel do**

if $k \geq 2^d$ **then** $x[k] := x[k - 2^{d-1}] + x[k]$



Double Buffered Version

for $d := 1$ **to** $\log_2 n$ **do**

  forall k **in parallel do**

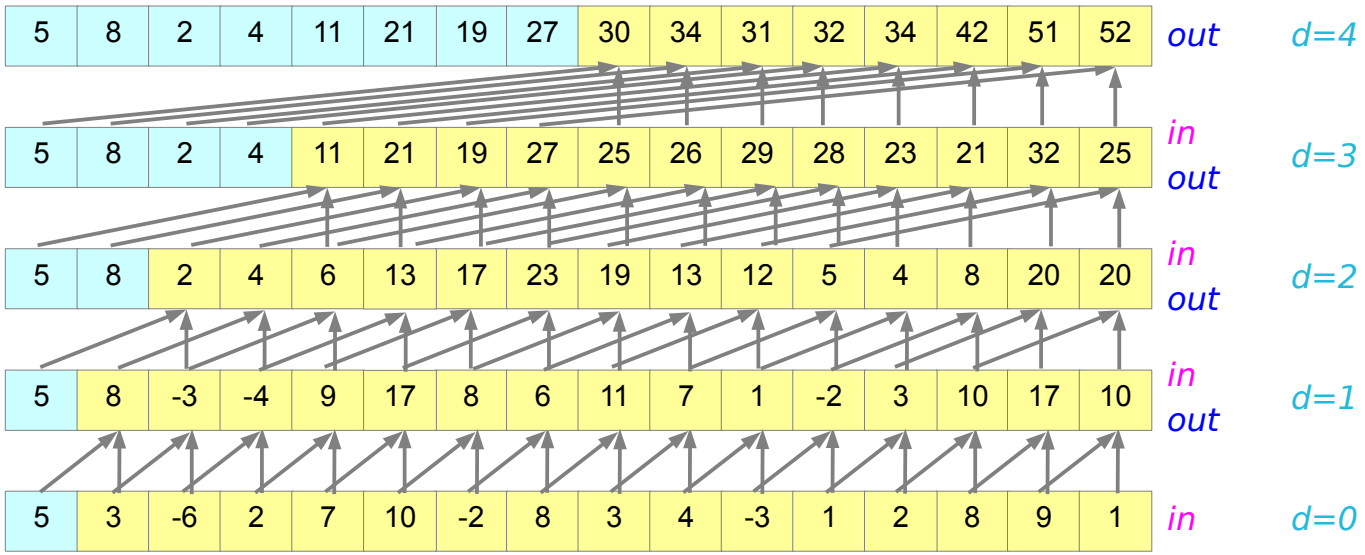
    if $k \geq 2^d$ **then**

      $x[out][k] := x[in][k - 2^{d-1}] + x[in][k]$

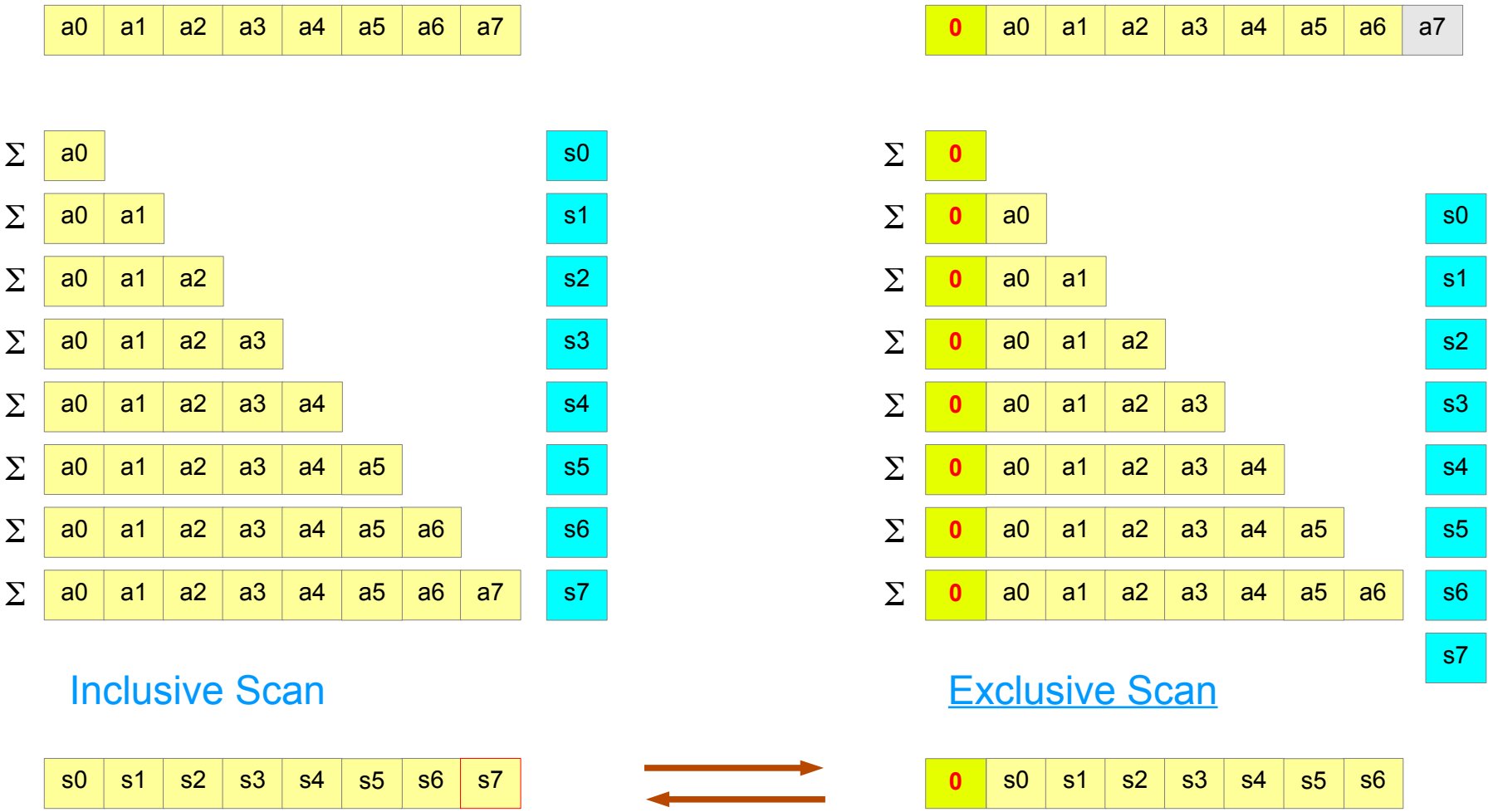
    else

      $x[out][k] := x[in][k]$

  swap(*in*,*out*)



Types of Scan Operations in CUDA



Double Buffered Version – CUDA Scan code

```
__global__ void scan(float *g_odata, float *g_idata, int n)
{
  extern __shared__ float temp[]; // allocated on invocation
  int thid = threadIdx.x;
  int pout = 0, pin = 1;

   // load input into shared memory.
   // This is exclusive scan, so shift right by one and set first elt to 0
  temp[pout*n + thid] = (thid > 0) ? g_idata[thid-1] : 0;
__syncthreads();
  for (int offset = 1; offset < n; offset *= 2)
  {
     pout = 1 - pout; // swap double buffer indices
     pin = 1 - pout;
     if (thid >= offset)
        temp[pout*n+thid] += temp[pin*n+thid - offset];
     else
        temp[pout*n+thid] = temp[pin*n+thid];
__syncthreads();
  }
  g_odata[thid] = temp[pout*n+thid]; // write output   (inclusive scan)
}
```

Double Buffer Pointers: pout and pin

```
pout = 1
pin = 0
```

```
pout = 1 – pout;
pin = 1 - pout;
```

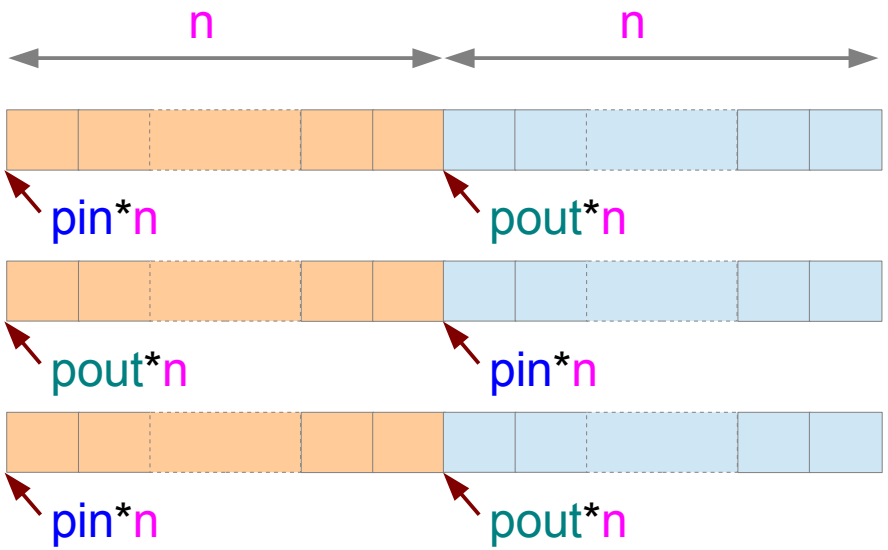
```
pout = 1
pin = 0
```

```
pout = 1 – pout;
pin = 1 - pout;
```

```
pout = 0
pin = 1
```

```
pout = 1 – pout;
pin = 1 - pout;
```

```
pout = 1
pin = 0
```




```
temp[pout*n + thid]
```

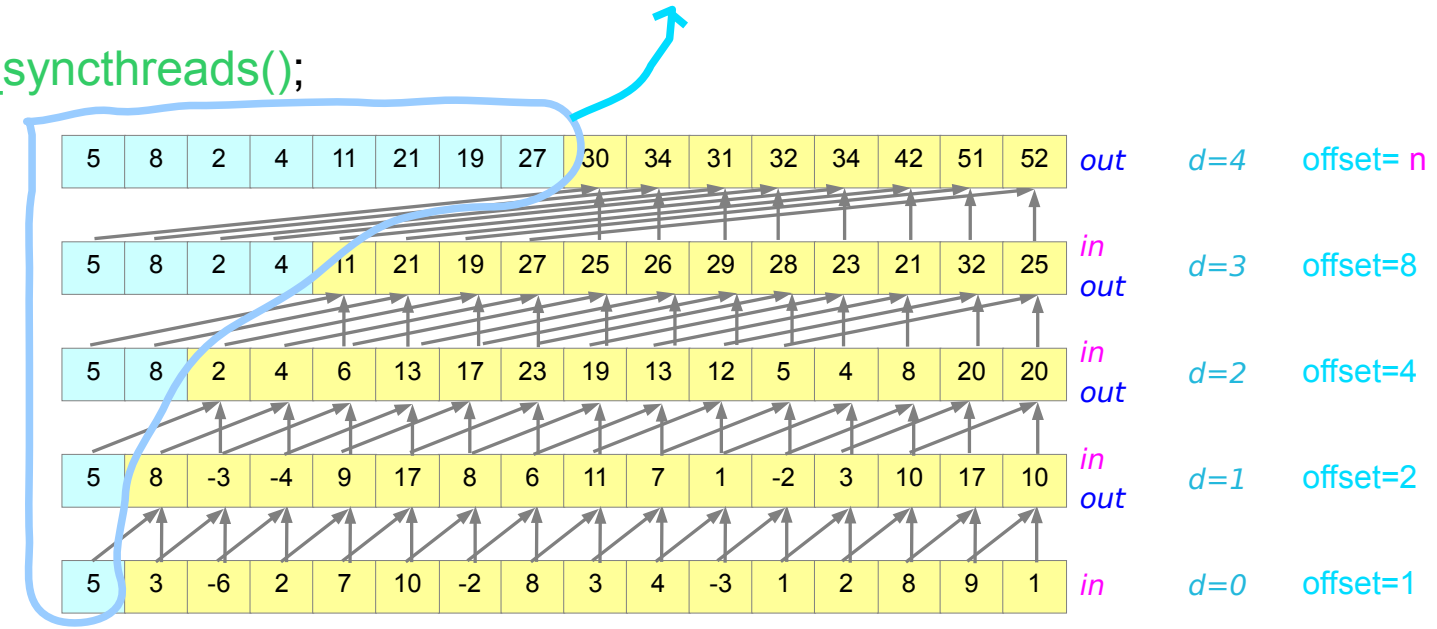
Log2 Steps

```
__syncthreads();
 for (int offset = 1; offset < n; offset *= 2)
 {
    pout = 1 - pout; // swap double buffer indices
    pin = 1 - pout;

    if (thid >= offset)
       temp[pout*n+thid] += temp[pin*n+thid - offset];
    else
       temp[pout*n+thid] = temp[pin*n+thid];

__syncthreads();
 }
```

```
int thid = threadIdx.x;
```



Double Buffered Version

Efficient Parallel Scan

Up-Sweep Phase

for $d := 0$ **to** $\log_2 n - 1$ **do**

  for k **from** 0 **to** $n - 1$ **by** 2^{d+1} **in parallel do**

    $x[k + 2^{d+1} - 1] := x[k + 2^d - 1] + x[k + 2^{d+1} - 1]$

d=4			
d=3	$x[k + 2^4 - 1] := x[k + 2^3 - 1] + x[k + 2^4 - 1]$	$x[k+15] := x[k+7] + x[k+15]$	by 16
d=2	$x[k + 2^3 - 1] := x[k + 2^2 - 1] + x[k + 2^3 - 1]$	$x[k+7] := x[k+3] + x[k+7]$	by 8
d=1	$x[k + 2^2 - 1] := x[k + 2^1 - 1] + x[k + 2^2 - 1]$	$x[k+3] := x[k+1] + x[k+3]$	by 4
d=0	$x[k + 2^1 - 1] := x[k + 2^0 - 1] + x[k + 2^1 - 1]$	$x[k+1] := x[k+0] + x[k+1]$	by 2

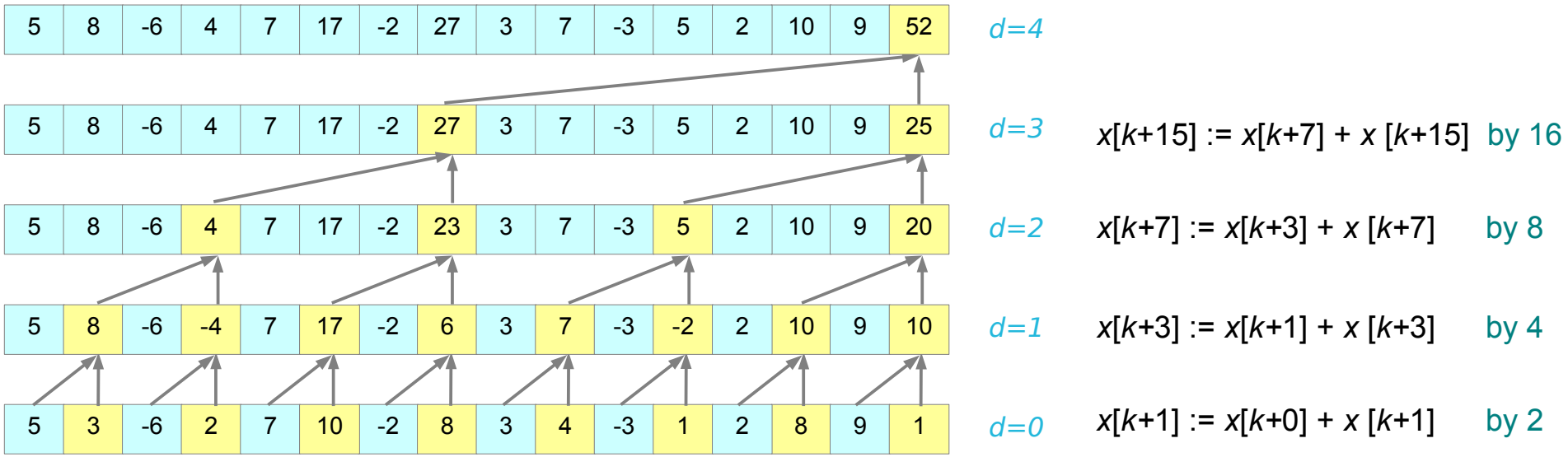
Efficient Parallel Scan

Up-Sweep Phase

for $d := 0$ **to** $\log_2 n - 1$ **do**

for k **from** 0 **to** $n - 1$ **by** 2^{d+1} **in parallel do**

$x[k + 2^{d+1} - 1] := x[k + 2^d - 1] + x[k + 2^{d+1} - 1]$



Efficient Parallel Scan

Down-Sweep Phase

x[n - 1] := 0
for d := $\log_2 n$ **down to** 0 **do**
  for k **from** 0 **to** n – 1 **by** 2^{d+1} **in parallel do**
    t := x[k + 2^d - 1]
    x[k + 2^d - 1] := x [k + 2^{d+1} - 1]
    x[k + 2^{d+1} - 1] := t + x [k + 2^{d+1} - 1]

d=4 t := x[*k+15*], x[*k15*] := x [*k+31*], x[*k+31*] := t + x [*k+31*]

d=3 t := x[*k+7*], x[*k+7*] := x [*k+15*], x[*k+15*] := t + x [*k+15*]

d=2 t := x[*k+3*], x[*k+3*] := x [*k+7*], x[*k+7*] := t + x [*k+7*]

d=1 t := x[*k+1*], x[*k+1*] := x [*k+3*], x[*k+3*] := t + x [*k+3*]

d=0 t := x[*k*], x[*k*] := x [*k+1*], x[*k+1*] := t + x [*k+1*]

Efficient Parallel Scan

Down-Sweep Phase

x[n - 1] := 0

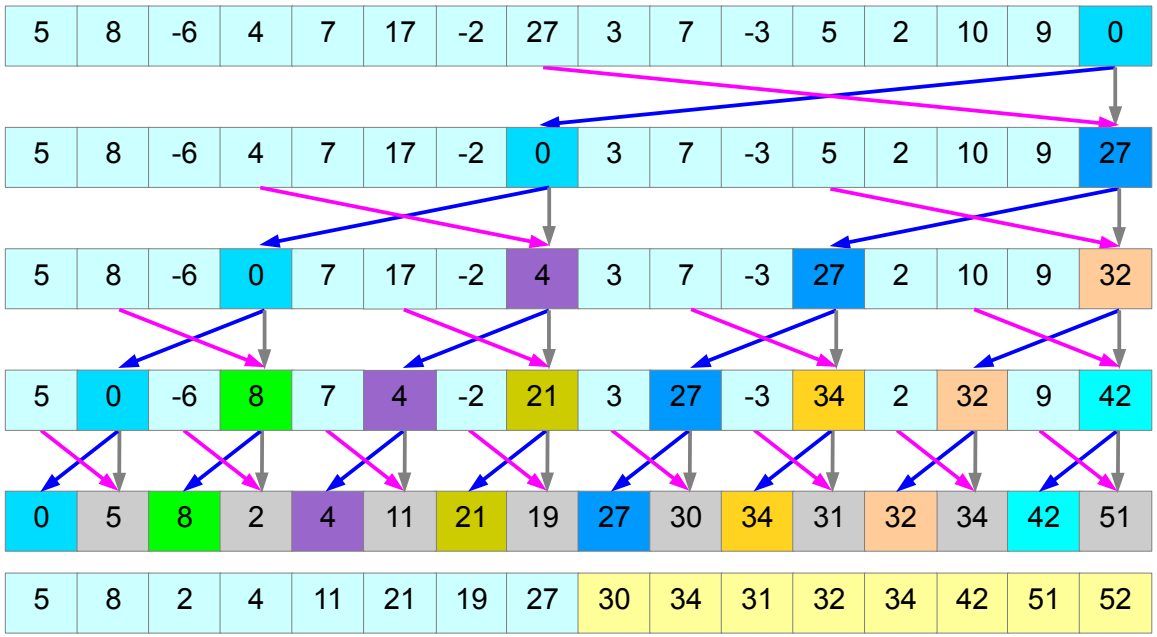
for d := $\log_2 n$ **down to** 0 **do**

  for k **from** 0 **to** n – 1 **by** 2^{d+1} **in parallel do**

    t := x[k + 2^d - 1]

    x[k + 2^d - 1] := x [k + 2^{d+1} - 1]

    x[k + 2^{d+1} - 1] := t + x [k + 2^{d+1} - 1]



	t := x[k+15], x[k15] := x [k+31], x[k+31] := t + x [k+31]
d=4	t := x[k+7], x[k+7] := x [k+15], x[k+15] := t + x [k+15]
d=3	t := x[k+3], x[k+3] := x [k+7], x[k+7] := t + x [k+7]
d=2	t := x[k+1], x[k+1] := x [k+3], x[k+3] := t + x [k+3]
d=1	t := x[k], x[k] := x [k+1], x[k+1] := t + x [k+1]
d=0	

Work Efficient Scan Sum (1)

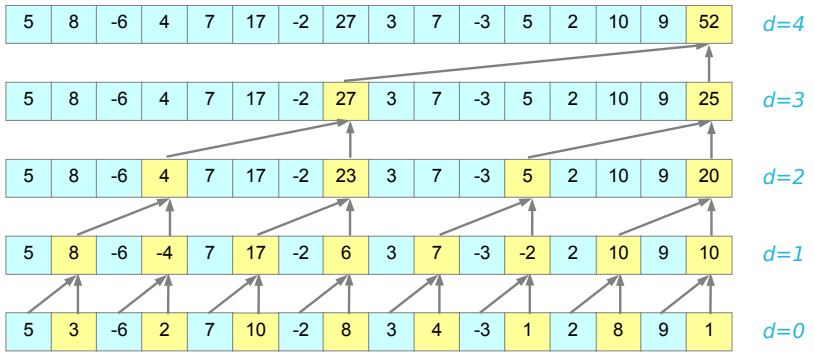
Up-Sweep Phase

```
__global__ void prescan(float *g_odata, float *g_idata, int n)
{
    extern __shared__ float temp[];// allocated on invocation
    int thid = threadIdx.x;
    int offset = 1;

A   temp[2*thid] = g_idata[2*thid]; // load input into shared memory
    temp[2*thid+1] = g_idata[2*thid+1];

    for (int d = n>>1; d > 0; d >>= 1) { // build sum in place up the tree
        __syncthreads();
        if (thid < d) {
B           int ai = offset*(2*thid+1)-1;
            int bi = offset*(2*thid+2)-1;

            temp[bi] += temp[ai];
        }
        offset *= 2;
    }
```



Work Efficient Scan Sum (2)

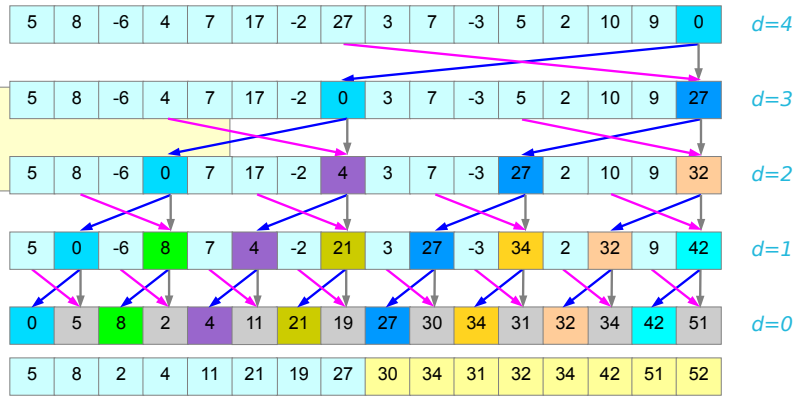
Down-Sweep Phase

```
if (thid == 0) { temp[n - 1] = 0; } // clear the last element
for (int d = 1; d < n; d *= 2) { // traverse down tree & build scan
        offset >>= 1;
        __syncthreads();
        if (thid < d) {
C               int ai = offset*(2*thid+1)-1;
                int bi = offset*(2*thid+2)-1;

                float t = temp[ai];
                temp[ai] = temp[bi];
                temp[bi] += t;
        }
}

__syncthreads();
D   g_odata[2*thid] = temp[2*thid]; // write results to device memory
    g_odata[2*thid+1] = temp[2*thid+1];

}
```



References

[1] en.wikipedia.org
[2] M Harris, http://beowulf.lcs.mit.edu/18.337-2008/lectslides/scan.pdf