CUDA Parallel Prefix Sum (1A)
for \( d := 1 \) to \( \log_2 n \) do

\[
\text{forall } k \text{ in parallel do }
\text{if } k \geq 2^d \text{ then } x[k] := x[k - 2^{d-1}] + x[k]
\]
Parallel Prefix Sum

\[
\text{for } d := 1 \text{ to } \log_2 n \text{ do}
\]

\[
\text{forall } k \text{ in parallel do}
\]

\[
\text{if } k \geq 2^d \text{ then } x[k] := x[k - 2^{d-1}] + x[k]
\]
**Double Buffered Version**

```plaintext
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

Inclusive Scan

Exclusive Scan
Double Buffered Version – CUDA Scan code

```c
__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

\begin{align*}
\text{pout} &= 1 - \text{pout;} \\
\text{pin} &= 1 - \text{pout;} \\
\text{pout} &= 1 - \text{pout;} \\
\text{pin} &= 1 - \text{pout;} \\
\text{pout} &= 1 - \text{pout;} \\
\text{pin} &= 1 - \text{pout;}
\end{align*}

\text{temp[\text{pout}^n + \text{thid}]}

\[ \text{pout} = 1 \]
\[ \text{pin} = 0 \]

\[ \text{pout} = 1 \]
\[ \text{pin} = 0 \]

\[ \text{pout} = 0 \]
\[ \text{pin} = 1 \]
__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();
}
Double Buffered Version
Efficient Parallel Scan

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

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

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

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

Up-Sweep Phase
Efficient Parallel Scan

\[
x[n - 1] := 0
\]

for \( d := \log_2 n \) down to 0 do

for \( k \) from 0 to \( n - 1 \) by \( 2^{d+1} \) in parallel do

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

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

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

\( d=4 \) \hspace{1cm} t := x[k+15], x[k+15] := x[k+31], x[k+31] := t + x[k+31]

\( d=3 \) \hspace{1cm} t := x[k+7], x[k+7] := x[k+15], x[k+15] := t + x[k+15]

\( d=2 \) \hspace{1cm} t := x[k+3], x[k+3] := x[k+7], x[k+7] := t + x[k+7]

\( d=1 \) \hspace{1cm} t := x[k+1], x[k+1] := x[k+3], x[k+3] := t + x[k+3]

\( d=0 \) \hspace{1cm} t := x[k], x[k] := x[k+1], x[k+1] := t + x[k+1]
Efficient Parallel Scan

\[ x[n - 1] := 0 \]

\textbf{for} \( d := \log_2 n \) \textbf{down to} 0 \textbf{do}

\textbf{for} \( k \) \textbf{from} 0 \textbf{to} \( n - 1 \) \textbf{by} \( 2^{d+1} \) \textbf{in parallel do}

\[ t := x[k + 2^d - 1] \]
\[ x[k + 2^d - 1] := x[k + 2^{d+1} - 1] \]
\[ x[k + 2^{d+1} - 1] := t + x[k + 2^{d+1} - 1] \]
Work Efficient Scan Sum (1)

```c
__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;

    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) {
            int ai = offset*(2*thid+1)-1;
            int bi = offset*(2*thid+2)-1;

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

Up-Sweep Phase

A

B

**CUDA Parallel**

Prefix Sum (1A)
if (thid == 0) { \texttt{temp}[n - 1] = 0; } // clear the last element
for (int d = 1; d < n; d *= 2) { // traverse down tree & build scan
    \texttt{offset} >>= 1;
    \_\_\_\_\_sy\texttt{ncth}threads();
    if (thid < d) {
        int \texttt{ai} = \texttt{offset}*(2*thid+1)-1;
        int \texttt{bi} = \texttt{offset}*(2*thid+2)-1;
        float \texttt{t} = \texttt{temp}[\texttt{ai}];
        \texttt{temp}[\texttt{ai}] = \texttt{temp}[\texttt{bi}];
        \texttt{temp}[\texttt{bi}] += \texttt{t};
    }
}
\_\_\_\_\_syncthreads();
\texttt{g\_odata}[2*thid] = \texttt{temp}[2*thid]; // write results to device memory
\texttt{g\_odata}[2*thid+1] = \texttt{temp}[2*thid+1];
References

[1] en.wikipedia.org