Deterministic Annealing in CUDA - UPDATE

A. C. O. Santos[1], T. Tomei[1], A. Sugunan[2]

[1]SPRACE, [2]TIFR

Table of Contents

1. Introduction and Goals

2. Deterministic Annealing

3. CUDA Implementation

4. Conclusions

Introduction and Goals: High luminosity collateral effect - Pileup

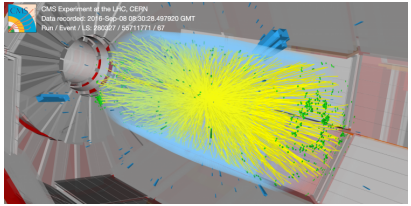


Figure: https://cms.cern/

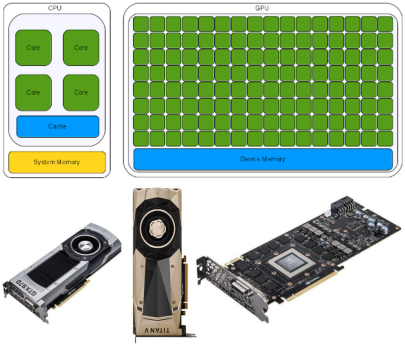


Figure: 10.1371/journal.pone.0097277

Goal: Primary Vertex Position - Deterministic Annealing

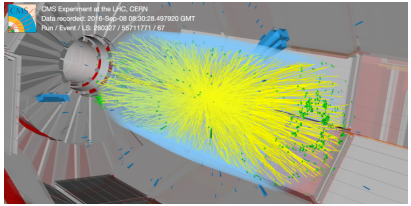


Figure: https://cms.cern/

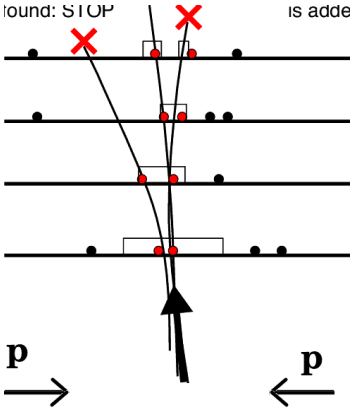


Figure: z-coordinate of their point of closest approach to the beam-line, arXiv:0902.1860v2

Deterministic Annealing

Consider index i related to tracks and k with vertex (always). Then we have

$$z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \qquad p_{ik} = \frac{e^{-\beta E_{ik}}}{\sum_{k'} \rho_{k'} e^{-\beta E_{ik'}}}, \qquad E_{ik} = \frac{(z_i - z_k)^2}{\sigma_i^2}.$$

Alternatively, we have the free energy

$$F_\beta(E_{ik}) = -\frac{1}{\beta} \sum_i \log \sum_k \exp\left(-\beta E_{ik}\right), \quad \beta = 1/T. \tag{1}$$

Constrained to

$$\begin{cases} \sum_k \rho_k p_{ik} = 1, \\ \sum_k \rho_k = 1 \end{cases}. \tag{2}$$

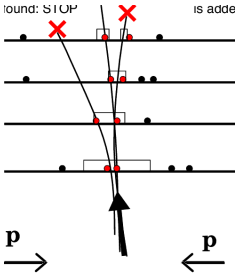
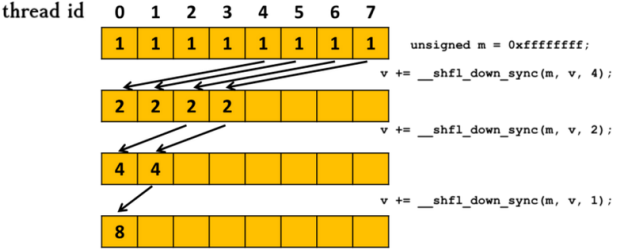


Figure: z-coordinate of their point of closest approach to the beam-line, arXiv:0902.1860v2

Cuda - Problems/Optimization - Memory Access - `shfl_down_sync()`

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$



Part of a warp-level parallel reduction using shfl_down_sync().

Figure: Ref: https://developer.nvidia.com/blog/using-cuda-warp-level-primitives/

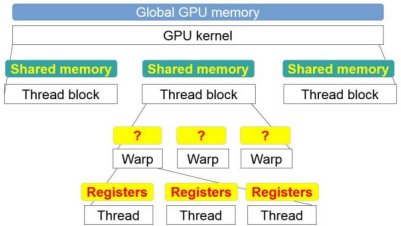
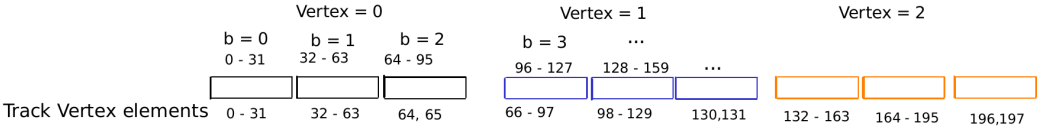


Figure: https://developer.nvidia.com/blog/register-cache-warp-cuda/

Cuda - Problems/Optimization - Parallel Reduction w/ Shuffle

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$

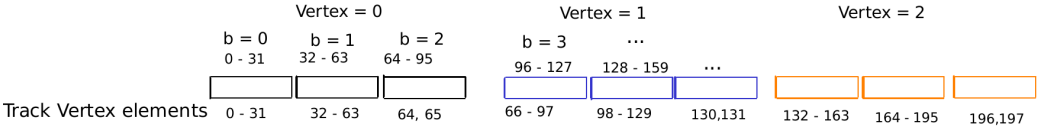
Suppose we have 66 tracks and 3 vertex



Cuda - Problems/Optimization - Parallel Reduction w/ Shuffle

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$

Suppose we have 66 tracks and 3 vertex

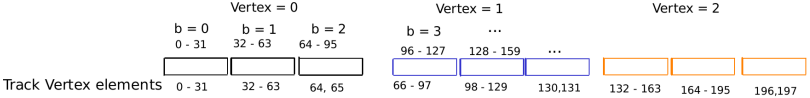


$N_{WS1} * \text{Track} + N_{WS2} * \text{Vertex} + tid$

Cuda - Problems/Optimization - Parallel Reduction w/ Shuffle

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$

Suppose we have 66 tracks and 3 vertex



N_{wf1} * Track + N_{wf2} * Vertex + tid

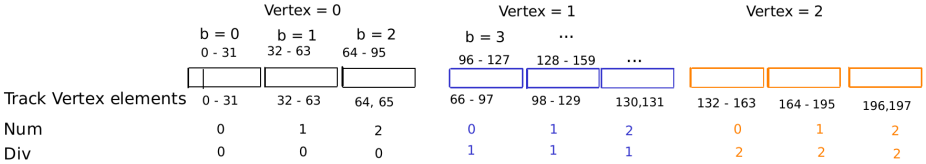
N_{wf2} = num = bid%N_warps_per_vertex

N_{wf1} = div = bid/N_warps_per_vertex;

Cuda - Problems/Optimization - Parallel Reduction w/ Shuffle

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$

Suppose we have 66 tracks and 3 vertex



$N_{wsf1} * \text{Track} + N_{wsf2} * \text{N\_warp} + \text{tid}$

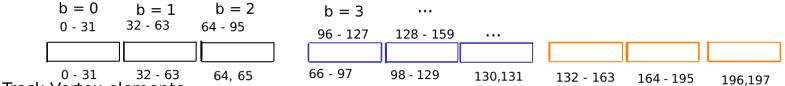
$N_{wsf2} =$ num = bid%N_warps_per_vertex

$N_{wsf1} =$ div = bid/N_warps_per_vertex;

Cuda - Problems/Optimization - Parallel Reduction w/ Shuffle

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$

Suppose we have 66 tracks and 3 vertex




```
int num = bid%N_warps_per_vertex;
int div = bid/N_warps_per_vertex;
int Did = num * warp + div * N_tracks + tid ;

int Lid = (N_warps_per_vertex -1 )% N_warps_per_vertex ;  // Last warp of a group of threads
int Rid = N_tracks%warp ;  // Last warp of a group of threads

    if (num != Lid){

            out[gid] = in[Did];
          }
    else {
        if (tid<Rid){   //  !!!!!!  Warning Warp Divergence !!!!!!

             out[gid] = in[Did];
           }
        else{
            out[gid] = 0.0;
         }
     }
```

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$

Suppose we have 66 tracks and 3 vertex

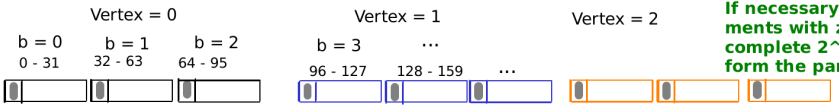
__shfl_down_sync Reduction/Summation

```
// Results of the summation over each warp will be in the first elements of the warp
for (int offset =  blockDim.x/2  ; offset > 0; offset /= 2) //

{

 out[gid] +=  __shfl_down_sync(0xffffffff, out[gid], offset);

 __syncthreads();              // Wait for all shuffle reductions per loop

}
```

Cuda - Problems/Optimization - Parallel Reduction w/ Shuffle

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$

Suppose we have 66 tracks and 3 vertex

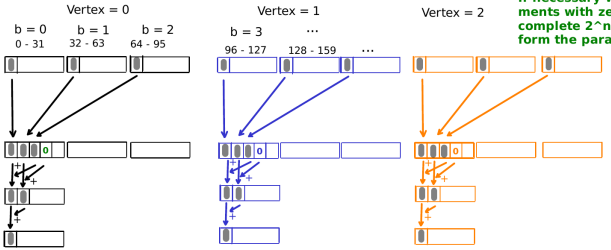


If necessary we need to fill elements with zeros what left to complete 2^n elements to perform the parallel reduction

Cuda - Problems/Optimization - Parallel Reduction w/ Shuffle

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$

Suppose we have 66 tracks and 3 vertex




```
int Sid = (tid + (num * warp) + (div * N_warps_per_vertex )) * warp; //
int warp_elem = N_warps_per_vertex * warp * (div+1); //  variable with maximum element value in each big block

if (Sid < warp_elem){
  __syncthreads();

      aux[gid] = out[Sid]; // Saver option

      }
```

Cuda - Problems/Optimization - Parallel Reduction w/ Shuffle

$$z_0 = \frac{\sum_i p_i z_i/\sigma_i^2}{\sum_i p_i/\sigma_i^2}, \qquad T_0 = 2\frac{\sum_i \frac{p_i}{\sigma_i^2}\left(\frac{z_i - z_k}{\sigma_i}\right)^2}{\sum_i \frac{p_i}{\sigma_i^2}}, \quad z_k = \frac{\sum_i p_i p_{ik} z_i/\sigma_i^2}{\sum_i p_i p_{ik}/\sigma_i^2}, \quad \rho_k, \quad T_c.$$

Suppose we have 66 tracks and 3 vertex

```
int N_2W = comp_list_2n_blocks(N_warps_per_vertex); // Calculates what is left to 2**n
int Pid = num * warp + tid; // variable index to limit the parallel reduction

if (Pid < N_2W/2){

  __syncthreads();

 for (int offset = N_2W/2 ; offset >0; offset /= 2)  {

  aux[gid] += aux[gid+offset];
 }

}
```

If you want print sum results in order

```
if (gid < N_vertex){
__syncthreads();

  summ[gid]= aux[gid*warp*N_warps_per_vertex];
}
else{
  summ[gid]= 0.0;
}
```

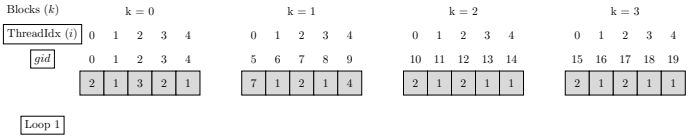
Conclusions

- A first CUDA version for *DA*;
- Proceed with the full version of the *DA* STARTED;
- Extrapolate for even and odd number of vertex DONE;
- Optimize memory transfers PLANNED;
- Streams PLANNED;
- CMSSW PLANNED.

The End

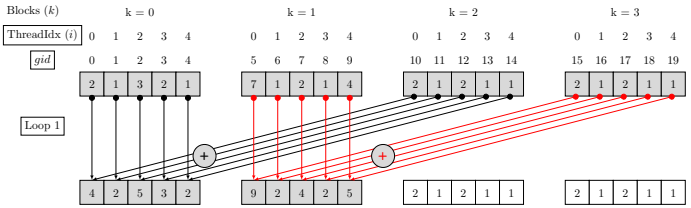
Cuda - Problems/Optimization - Parallel Reduction

$$p_{ik} = \frac{e^{-\beta E_{ik}}}{\sum_{k'} \rho_{k'} e^{-\beta E_{ik'}}}, \quad E_{ik} = \frac{(z_i - z_k)^2}{\sigma_i^2}, \quad \texttt{off} = \texttt{blockDim.x} * \texttt{gridDim.x}/2 + \texttt{gid} \rightrightarrows \texttt{sum}[\texttt{gid}] += \texttt{sum}[\texttt{off}].$$



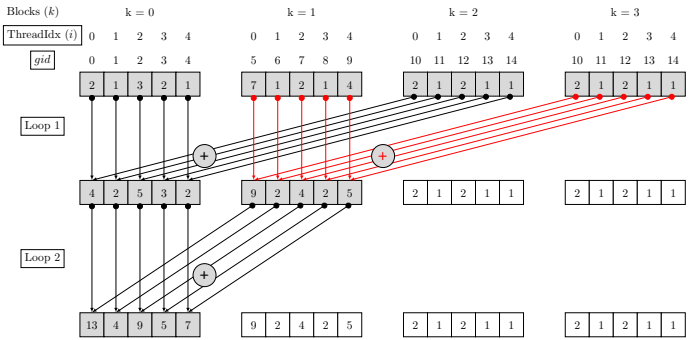
Cuda - Problems/Optimization - Parallel Reduction

$$p_{ik} = \frac{e^{-\beta E_{ik}}}{\sum_{k'} \rho_{k'} e^{-\beta E_{ik'}}}, \quad E_{ik} = \frac{(z_i - z_k)^2}{\sigma_i^2}, \quad \texttt{off} = \texttt{blockDim.x} * \texttt{gridDim.x}/2 + \texttt{gid} \rightrightarrows \texttt{sum[gid]} += \texttt{sum[off]}.$$



Cuda - Problems/Optimization - Parallel Reduction

$$p_{ik} = \frac{e^{-\beta E_{ik}}}{\sum_{k'} \rho_{k'} e^{-\beta E_{ik'}}}, \quad E_{ik} = \frac{(z_i - z_k)^2}{\sigma_i^2}, \quad \texttt{off} = \texttt{blockDim.x} * \texttt{gridDim.x}/2 + \texttt{gid} \rightrightarrows \texttt{sum[gid]} += \texttt{sum[off]}.$$



Cuda - Problems/Optimization - Parallel Reduction

$$p_{ik} = \frac{e^{-\beta E_{ik}}}{\sum_{k'} \rho_{k'} e^{-\beta E_{ik'}}}, \quad E_{ik} = \frac{(z_i - z_k)^2}{\sigma_i^2}, \quad \texttt{off} = \texttt{blockDim.x} * \texttt{gridDim.x}/2 + \texttt{gid} \rightrightarrows \texttt{sum}[\texttt{gid}] += \texttt{sum}[\texttt{off}].$$

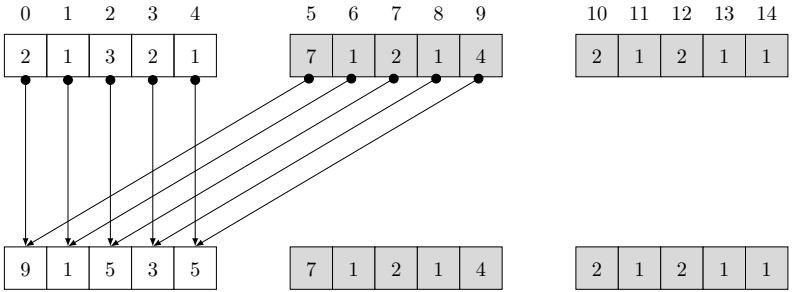


Figure: Parallel reduction for odd n. vertex.

Cuda - Problems/Optimization - Parallel Reduction

$$p_{ik} = \frac{e^{-\beta E_{ik}}}{\sum_{k'} \rho_{k'} e^{-\beta E_{ik'}}}, \quad E_{ik} = \frac{(z_i - z_k)^2}{\sigma_i^2}, \quad \texttt{off} = \texttt{blockDim.x} * \texttt{gridDim.x}/2 + \texttt{gid} \rightrightarrows \texttt{sum[gid]} += \texttt{sum[off]}.$$

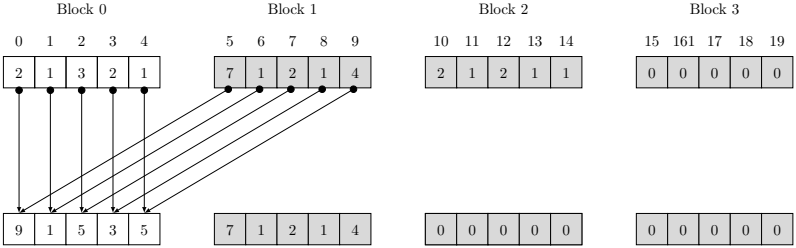


Figure: Parallel reduction for odd n. vertex. Possibility round up to archive 2^n.

Cuda - Problems/Optimization - Parallel Reduction

$$p_{ik} = \frac{e^{-\beta E_{ik}}}{\sum_{k'} \rho_{k'} e^{-\beta E_{ik'}}}, \quad E_{ik} = \frac{(z_i - z_k)^2}{\sigma_i^2}, \quad \texttt{off} = \texttt{blockDim.x} * \texttt{gridDim.x}/2 + \texttt{gid} \rightrightarrows \texttt{sum[gid]} += \texttt{sum[off]}.$$

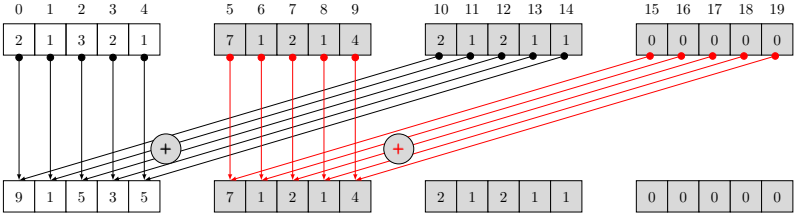


Figure: Parallel reduction for odd n. vertex. Possibility round up to archive 2^n.

Results with CUDA Implementation

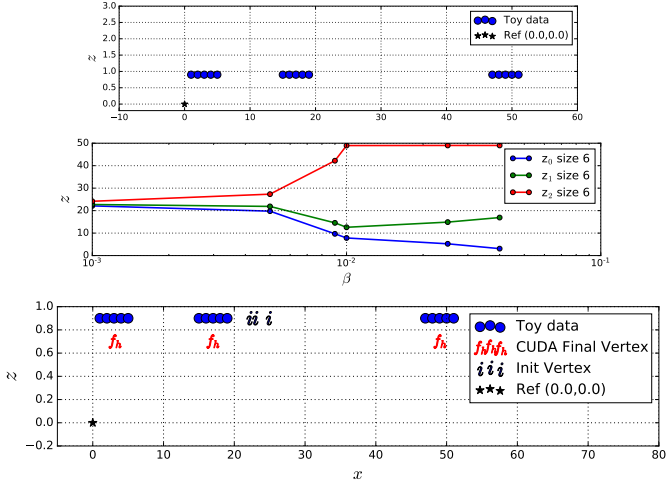


Figure: Toy data (on top). Middle figure corresponds to the vertex position evolution through annealing step. Bottom figure shows the data and the final vertex position after six annealing steps.

Conclusions

- A first CUDA version for *DA*;
- Extrapolate for even and odd number of vertex;
- Optimize memory transfers;
- Streams;

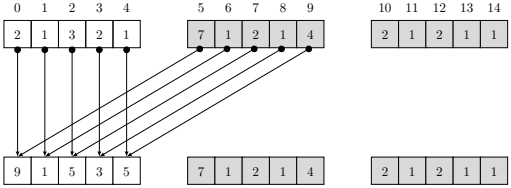


Figure: Parallel reduction for odd n. vertex.

Conclusions

- Extrapolate for even and odd number of vertex;
- Optimize memory transfers;
- Streams;

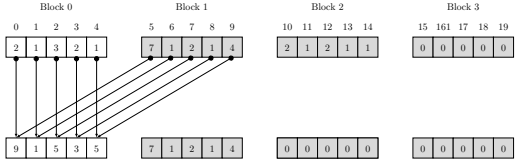


Figure: Parallel reduction for odd n. vertex. Possibility round up to archive 2^n.

Conclusions

- Extrapolate for even and odd number of vertex;
- Optimize memory transfers;
- Streams;

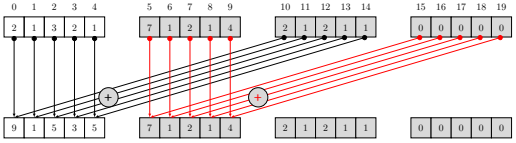


Figure: Parallel reduction for odd n. vertex. Possibility round up to archive 2^n.

Conclusions

- Extrapolate for even and odd number of vertex;
- Optimize memory transfers;
- Streams;

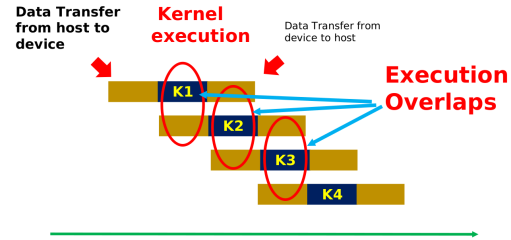


Figure: Cuda programming Masterclass. Kasun Liyanage.

- Avoid `cudaMalloc*()`, `cudaHostAlloc()`, `cudaFree*()`, `cudaHostRegister()`, `cudaHostUnregister()` on every event ;
- Use `cudaMemcpyAsync()`, `cudaMemsetAsync()`, `cudaMemPrefetchAsync()` etc.
- Synchronization needs should be fulfilled with `ExternalWork` extension to `EDProducers`;
- Within `acquire()/produce()`, the current CUDA device is set implicitly and the CUDA stream is provided by the system (with `cms::cuda::ScopedContextAcquire/cms::cuda::ScopedContextProduce`)...

The End

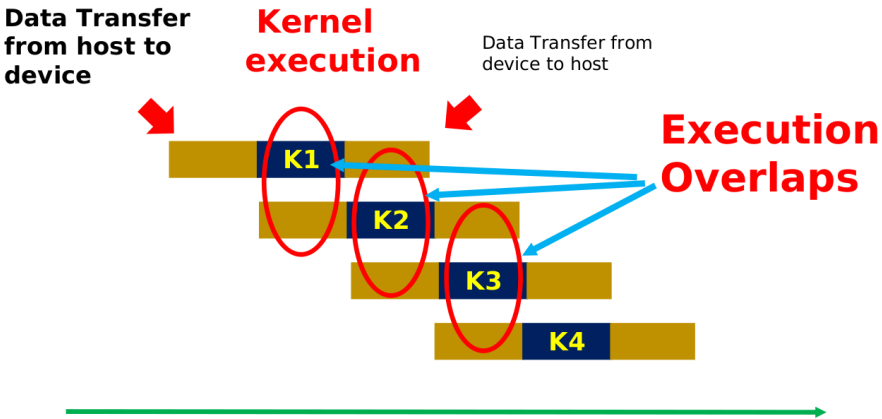


Figure: Cuda programming Masterclass. Kasun Liyanage.