SPRACE

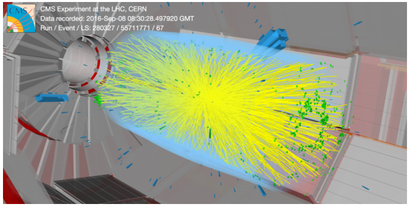
Deterministic Annealing in CUDA - UPDATE

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

[1]SPRACE, [2]TIFR

Table of Contents

Introduction and Goals: High luminosity collateral effect - Pileup



CMS Experiment at the LHC, CERN
Data recorded: 2016-Sep-08 08:30:28.497920 GMT
Run / Event / LS: 280327 / 55711771 / 67

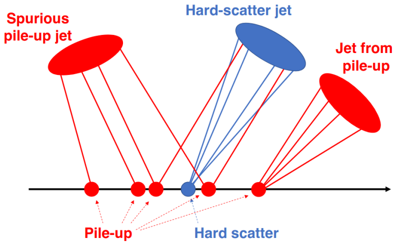


Figure: https://cms.cern/

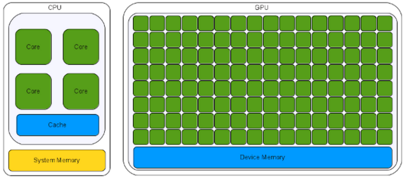
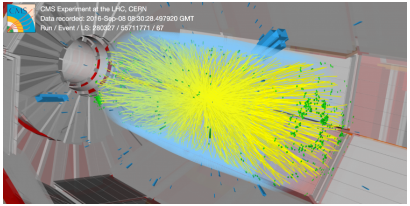


Figure: 10.1371/journal.pone.0097277

Goal: Primary Vertex Position - Deterministic Annealing



CMS Experiment at the LHC, CERN
Data recorded: 2016-Sep-08 08:30:28.497920 GMT
Run / Event / LS: 280327 / 55711771 / 67

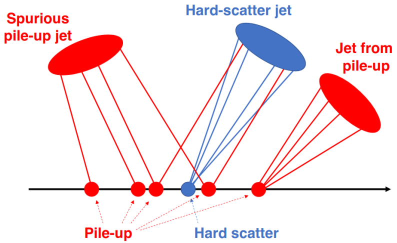


Figure: https://cms.cern/

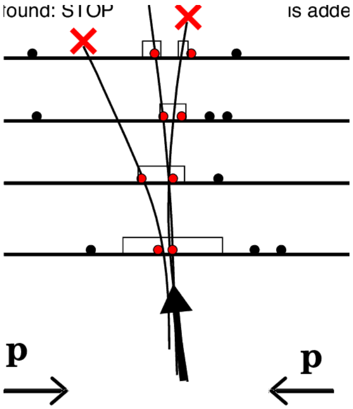


Figure: z-coordinate of their point of closest approach to the beam-line, arXiv:0902.1860v2

Deterministic Annealing

Consider index i related to tracks and k with vertex (always). We want to calculate

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

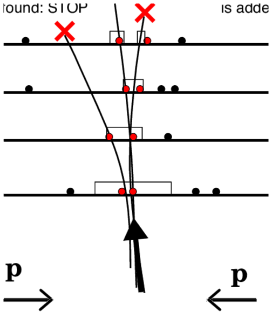
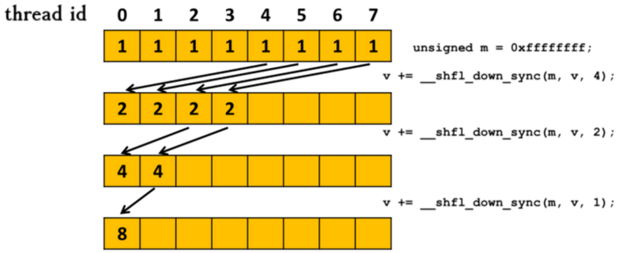


Figure: z-coordinate of their point of closest approach to the beam-line, arXiv:0902.1860v2

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

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



```
unsigned m = 0xffffffff;
v += __shfl_down_sync(m, v, 4);
v += __shfl_down_sync(m, v, 2);
v += __shfl_down_sync(m, v, 1);
```

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

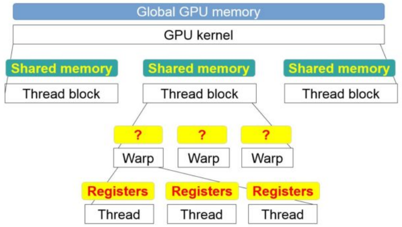


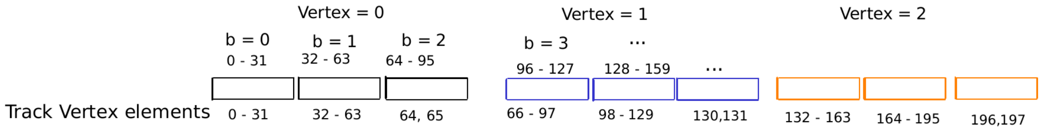
Figure: https://developer.nvidia.com/blog/register-cache-warp-cuda/

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

Cuda - Problems/Optimization - Parallel Reduction w/ Shuffle

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

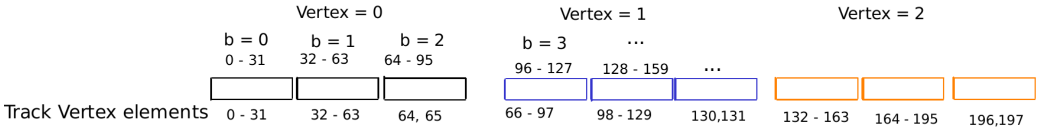
Suppose we have 66 tracks and 3 vertex



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

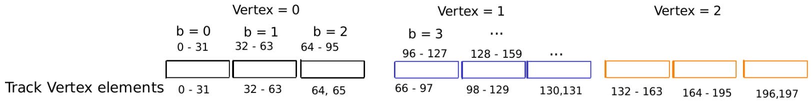


N_{ws1} * Track + N_{ws2} * N_warp + tid

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



$N_{w_{f1}}$ * Track + $N_{w_{f2}}$ * N_warp + tid

$N_{w_{f2}} =$ num = bid%N_warps_per_vertex

$N_{w_{f1}} =$ 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

	Vertex = 0			Vertex = 1			Vertex = 2		
	b = 0	b = 1	b = 2	b = 3	...				
	0 - 31	32 - 63	64 - 95	96 - 127	128 - 159	...			
Track Vertex elements	0 - 31	32 - 63	64, 65	66 - 97	98 - 129	130,131	132 - 163	164 - 195	196,197
Num	0	1	2	0	1	2	0	1	2
Div	0	0	0	1	1	1	2	2	2

N_{ws1} * Track + N_{ws2} * N_warp + tid

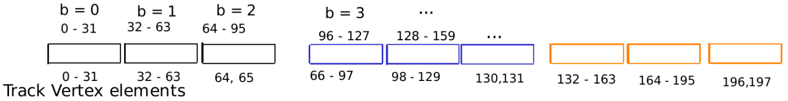
N_{ws2} = **num = bid%N_warps_per_vertex**

N_{ws1} = **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;
               }
      }
```

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

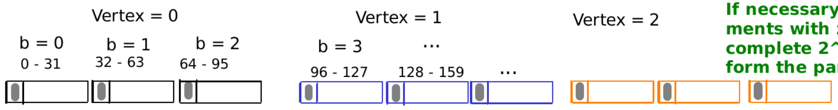
__shfl_down_sync Reduction/Summation

```
// 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

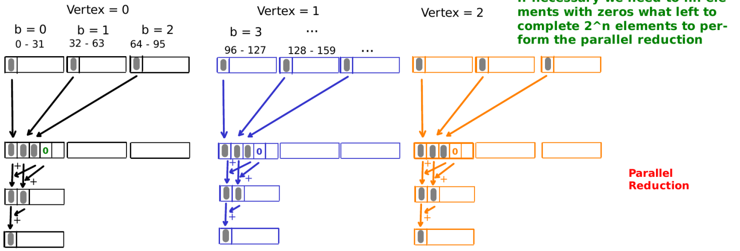


If necessary we need to fill elements with zeros what left to complete 2^n elements to perform the parallel reduction

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



Vertex = 0
b = 0
0 - 31
b = 1
32 - 63
b = 2
64 - 95
Vertex = 1
b = 3
96 - 127
...
128 - 159
...
Vertex = 2
If necessary we need to fill elements with zeros what left to complete 2^n elements to perform the parallel reduction
0
+
Parallel Reduction

```
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

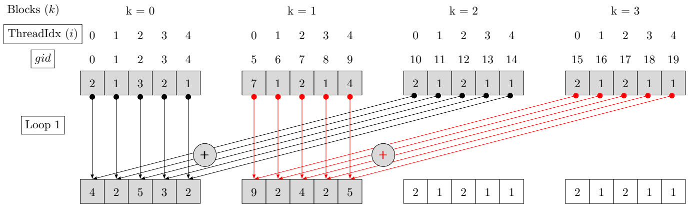
$$p_{ik} = \frac{e^{-\beta E_{ik}}}{\sum_{k'} \rho_{k'} e^{-\beta E_{ik'}}}, \quad E_{ik} = \frac{(z_i - z_k)^2}{\sigma_i^2}, \quad \boxed{\texttt{off} = \texttt{blockDim.x} * \texttt{gridDim.x}/2 + \texttt{gid}} \rightrightarrows \texttt{sum}[\texttt{gid}] += \texttt{sum}[\texttt{off}].$$

Blocks (k)	k = 0					k = 1					k = 2					k = 3				
ThreadIdx (i)	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
gid	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	2	1	3	2	1	7	1	2	1	4	2	1	2	1	1	2	1	2	1	1

Loop 1

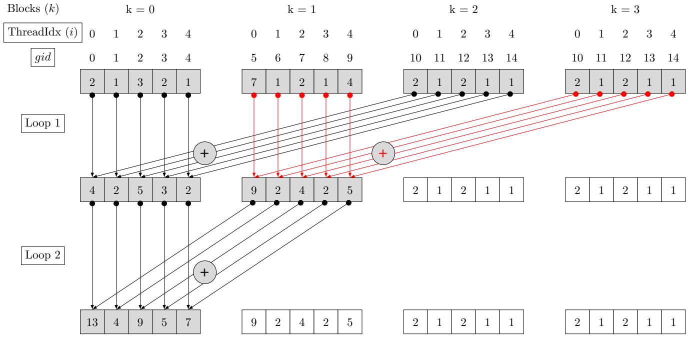
Cuda - Problems/Optimization - Parallel Reduction

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



Cuda - Problems/Optimization - Parallel Reduction

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



Cuda - Problems/Optimization - Parallel Reduction

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

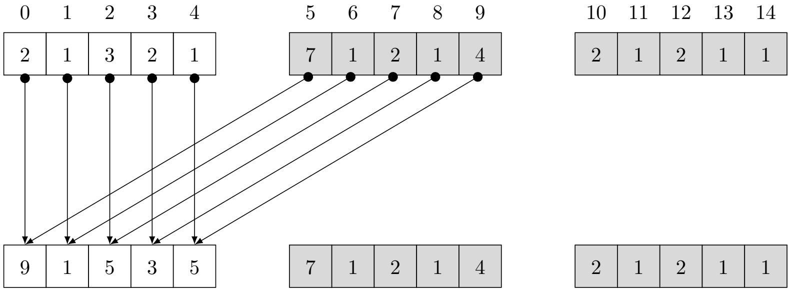


Figure: Parallel reduction for odd n. vertex.

Cuda - Problems/Optimization - Parallel Reduction

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

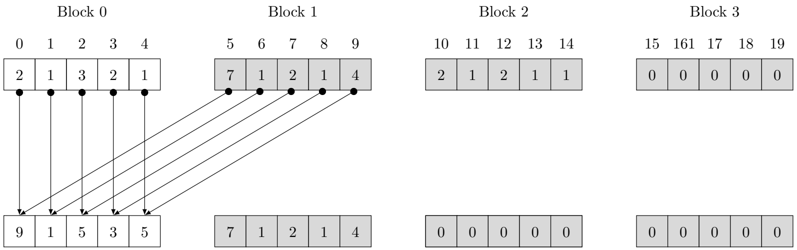


Figure: Parallel reduction for odd n. vertex. Possibility round up to archive 2^n.

Cuda - Problems/Optimization - Parallel Reduction

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

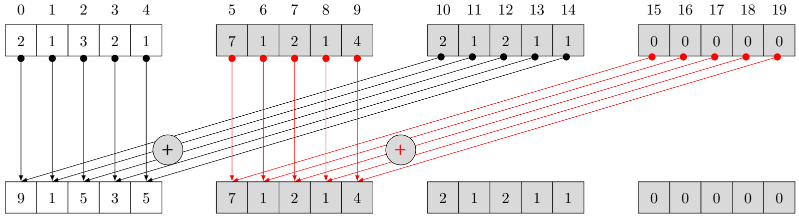
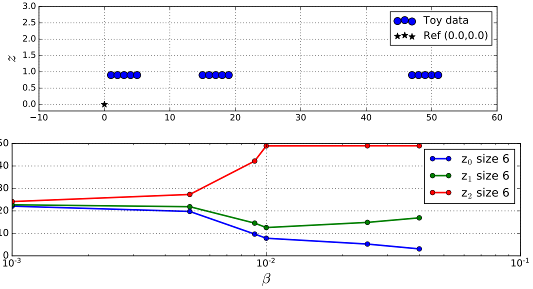


Figure: Parallel reduction for odd n. vertex. Possibility round up to archive 2^n.

Results with CUDA Implementation



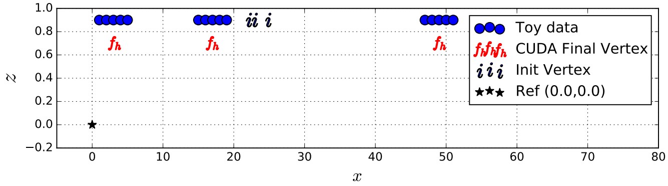


Figure: Toy data (on top). Middle figure corresponds to the vertex position evolution through annealing step. Bottom figure shows the data and the final vertex position after six annealing steps.

Conclusions

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

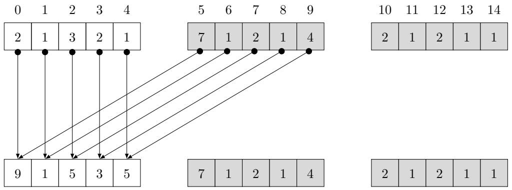


Figure: Parallel reduction for odd n. vertex.

Conclusions

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

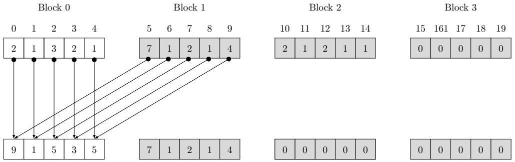


Figure: Parallel reduction for odd n. vertex. Possibility round up to archive 2^n.

Conclusions

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

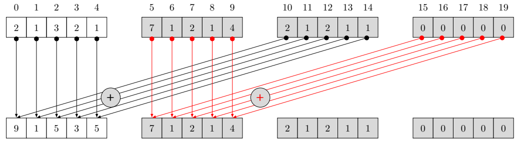


Figure: Parallel reduction for odd n. vertex. Possibility round up to archive 2^n.

Conclusions

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

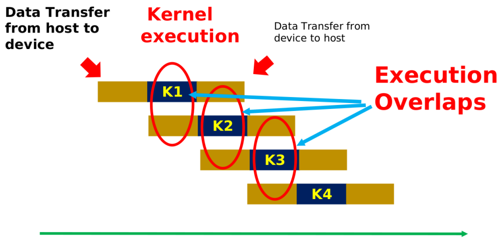

Data Transfer from host to device
Kernel execution
Data Transfer from device to host
Execution Overlaps
K1
K2
K3
K4

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

Cuda - Problems/Optimization - Memory Access - Streams.

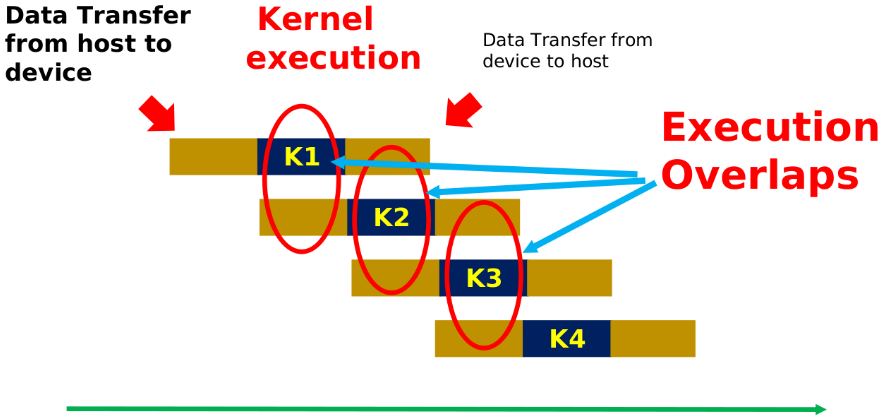

Data Transfer from host to device
Kernel execution
Data Transfer from device to host
Execution Overlaps
K1
K2
K3
K4

Figure: Cuda programming Masterclass. Kasun Liyanage.