Princeton GPU Hackathon

EcoSLIM

Jun Zhang
U of Arizona

Hoang Tran
Princeton U

Chen Yang
Princeton U

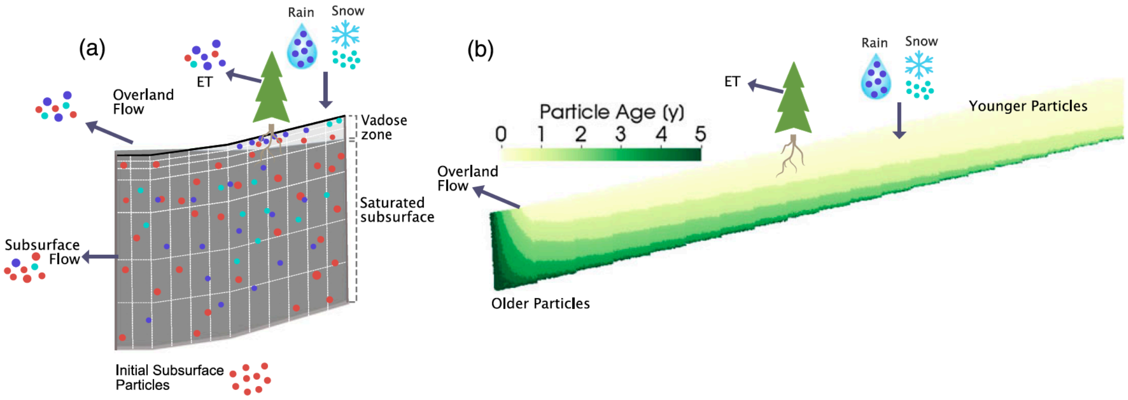
Troy Comi
Princeton U

Carl Ponder
NVIDIA

EcoSLIM

- A Lagrangian, particle tracking code that simulates advective and diffusive movement of water parcels in subsurface.

  Pollock's method (Pollock, 1988, groundwater) with some new features.



(a) Schematic of particle tracking method and (b) example of particle age distribution during a simulation

- Library: MPI, cuRAND, Thrust
- Language: Fortran
- GPU port path: CUDA Fortran
- Function focused: Sorting inactive particles/ compaction of the particle array

Sorting inactive particles

```
real*8,allocatable,pinned::P(:,:)
        ! P = Particle array [np,attributes]
        ! np = Number of particles
        ! P(np,1) = X coordinate [L]
        ! P(np,2) = Y coordinate [L]
        ! P(np,3) = Z coordinate [L]
        ! P(np,4) = Particle residence time [T]
        ! P(np,5) = Saturated particle residence time [T]
        ! P(np,6) = Particle mass; assigned via preciptiation or snowmel
        ! P(np,7) = Particle source (1=IC, 2=rain, 3=snowmelt, 4=irrigat
        ! P(np,8) = Particle Status (1=active, 0=inactive)
        ! P(np,9) = concentration
        ! P(np,10) = Exit status (1=outflow, 2=ET...)

        ! P(np,11) = Length of flow path [L]
        ! P(np,12) = Length of saturated flow path [L]
        ! P(np,13:(12+nind)) = Length of flow path in indicator i [L]
        ! P(np,(13+nind):(12+nind*2)) = particle age in indicator i [T]

        ! P(np,13+nind*2) = Particle Number (This is a unique integer ic
        ! P(np,14+nind*2) = Partical Initial X coordinate [L]
        ! P(np,15+nind*2) = Partical Initial Y coordinate [L]
        ! P(np,16+nind*2) = Partical Initial Z coordinate [L]
        ! P(np,17+nind*2) = Time that particle was added [T]
```

- Some particles will become inactive at the end of each time step. Status of a particle is indicated by P(n,8).
- Now we use serial algorithm on CPU to remove the inactive particles (P(n,8)=0), by replacing an inactive particle using an active particle from the bottom of the P array.
- We want to realize its parallelization on GPU by using stream compaction.

Particle array with dimension of

n particles × (17+nind*2) attributions

	P(:,1)	P(:,2)	P(:,3)	P(:,4)	P(:,5)	P(:,6)	P(:,7)	P(:,8)	....
1								1	
2								0	
3								0	
4								1	
5								1	
6								0	
7								1	
8								1	
9								0	
10								1	
....									

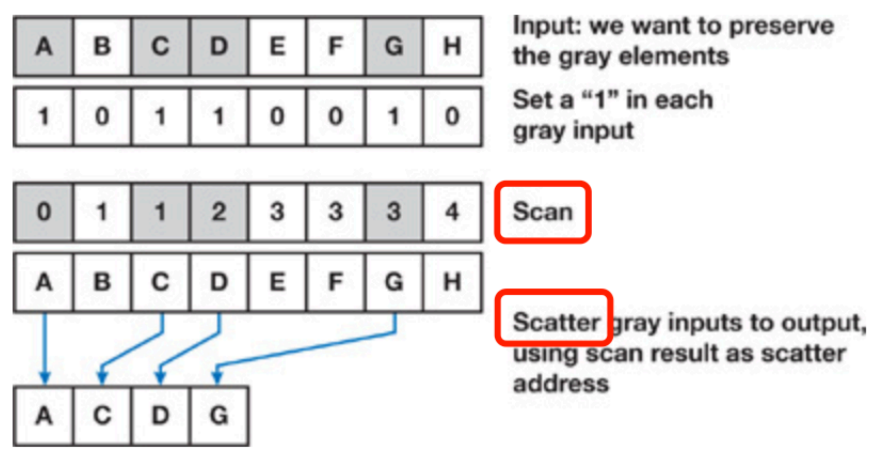


Figure 39-10 Scan and Scatter

Goals

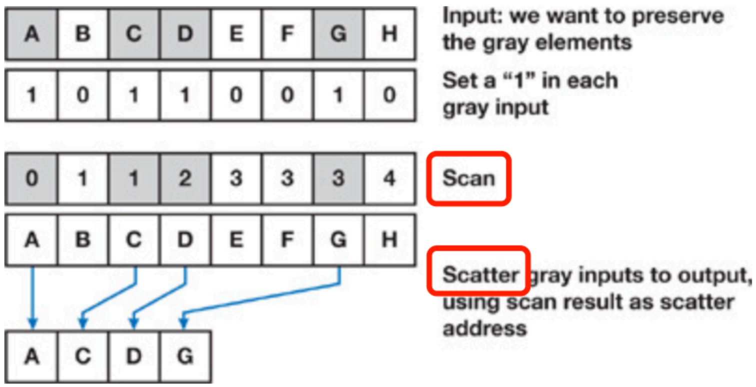


Figure 39-10 Scan and Scatter

```
template <class T, bool isBackward>
__global__ void compactData(T                     *d_out,
                            size_t                *d_numValidElements,
                            const unsigned int *d_indices, // Exclusive Sum-Scan Result
                            const unsigned int *d_isValid,
                            const T               *d_in,
                            unsigned int          numElements)
{
    if (threadIdx.x == 0)

if (iGlobal < numElements && d_isValid[iGlobal] > 0) {
    d_out[d_indices[iGlobal]] = d_in[iGlobal];
}
```

Scatter in cudpp **1D array**

1) For scan: writing a wrapper to call **exclusive_scan in Thrust** which is a CUDA C++ template library;
2) For scatter: modifying the kernel from cudpp to handle 2D particle array P;
3) For scatter: writing a wrapper to call this CUDA C++ kernel in Fortran; Or probably rewrite it in CUDA Fortran;
4) Test and profile scan and scatter using restart files generated in previous simulations;
5) Using cmake, Nsight, etc.