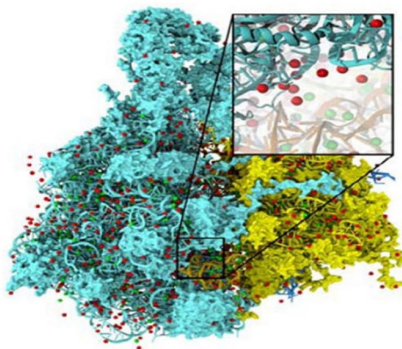


# Electrostatic Potential Map

# Electrostatic Potential Map

---

- This calculation is often used
  - in the placement of ions into a molecular structure for molecular dynamics simulation
  - to identify spatial locations where ions (red dots) can fit in according to physical laws
  - to calculate time-averaged electric field potential maps during molecular dynamics simulation
  - for the simulation process, as well as the visualization and analysis of simulation results



# VMD

---

- The examples will be taken from [VMD](#)
- A molecular dynamics application
  - designed for displaying, animating, and analyzing biomolecular systems

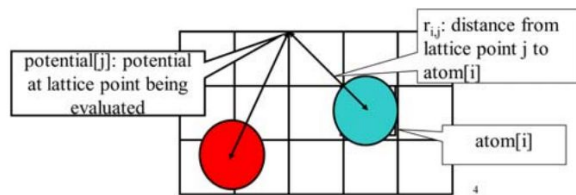
# Computation

---

- There are several methods for calculating electrostatic potential maps
- Direct Coulomb Summation (DCS) is a highly accurate method that is particularly suitable for GPUs
  - The DCS method calculates the electrostatic potential value of each grid point as the sum of contributions from all atoms in the system

# Direct Coulomb Summation (DCS)

- The contribution of atom  $i$  to a lattice point  $j$  is the charge of atom  $i$  divided by the distance from lattice point  $j$  to atom  $i$ 
  - Since this needs to be done for all grid points and all atoms, the number of calculations is proportional to the product of the total number of atoms in the system and the total number of grid points



# Sequential Scatter Approach

---

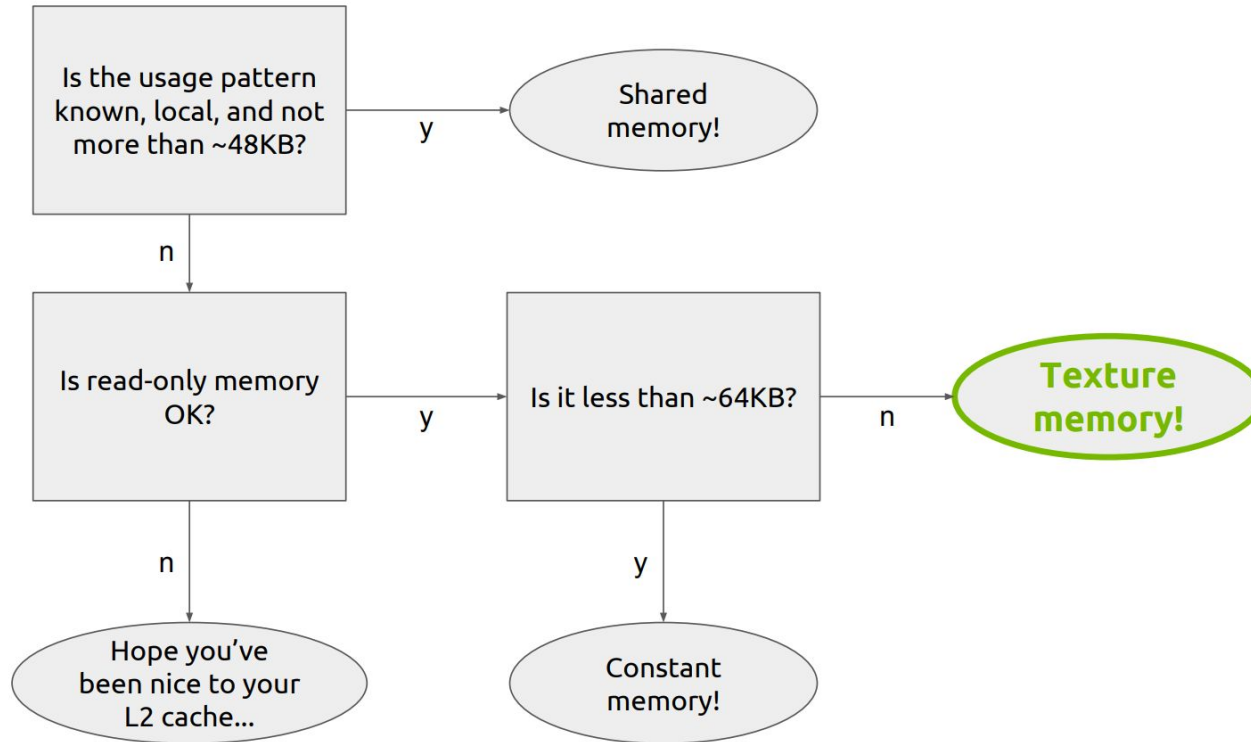
- The outer three levels iterate over the  $z$ ,  $y$ , and  $x$  dimensions of the grid point space
  - For each grid point, the innermost loop iterates over all atoms, calculating the contribution of electrostatic potential energy from all atoms to the grid point

# Sequential Scatter Approach: Optimized

---

- First, the innermost loop has been exchanged into the outermost loop
  - Thus, the code iterates over all atoms
  - For each atom, the inner loops scatter the contribution of the atom to all the grid points
- This enables the pre-computation of some values at the outer loops

# Memory Optimization Flowchart





# Parallel Scatter Approach

---

- It makes direct use of constant memory
  - Atoms are divided into CHUNKS
  - If they do not fit (64 KB), multiple kernel calls must be performed
  - The massive reuse of these constant memory elements across threads makes the constant cache extremely effective
- CPU transfers the chunk into the device's constant memory
  - Invokes the DCS kernel to calculate the contribution of the current chunk to the current slice
  - Prepares to transfer the next chunk
- It requires an atomic operation to update data

# Parallel Gather Approach

---

- We can instead use a gather approach in which each thread calculates the accumulated contributions of all atoms to one grid point
  - This is a preferred approach since each thread will be written into its own grid point, and there is no need to use atomic operations
- We form a 2D thread grid that matches the 2D potential grid point organization
  - Within each thread grid, the thread blocks are organized to calculate the electrostatic potential of tiles of the grid structure
  - Tiling can be used to push performance
- Each thread does 11 floating-point operations for every 4 memory elements accessed

# Problems and Optimizations

- Thread Coarsening
- Memory Coalescing

| Optimization                              | Benefit to compute cores                                      | Benefit to memory   | Strategies   |
|---|---|---|--|
| Maximizing occupancy                      | More work to hide pipeline latency                            | More parallel memory accesses to hide DRAM latency                      | Tuning usage of SM resources such as threads per block, shared memory per block, and registers per thread  |
| Enabling coalesced global memory accesses | Fewer pipeline stalls waiting for global memory accesses      | Less global memory traffic and better utilization of bursts/cache lines | Transfer between global memory and shared memory in a coalesced manner and performing uncoalesced accesses in shared memory (e.g., corner turning)<br>Rearranging the mapping of threads to data<br>Rearranging the layout of the data |
| Minimizing control divergence             | High SIMD efficiency (fewer idle cores during SIMD execution) | —   | Rearranging the mapping of threads to work and/or data<br>Rearranging the layout of the data   |
| Tiling of reused data                     | Fewer pipeline stalls waiting for global memory accesses      | Less global memory traffic  | Placing data that is reused within a block in shared memory or registers so that it is transferred between global memory and the SM only once  |
| Privatization (covered later)             | Fewer pipeline stalls waiting for atomic updates              | Less contention and serialization of atomic updates                     | Applying partial updates to a private copy of the data and then updating the universal copy when done  |
| Thread coarsening                         | Less redundant work, divergence, or synchronization           | Less redundant global memory traffic                                    | Assigning multiple units of parallelism to each thread to reduce the price of parallelism when it is incurred unnecessarily  |

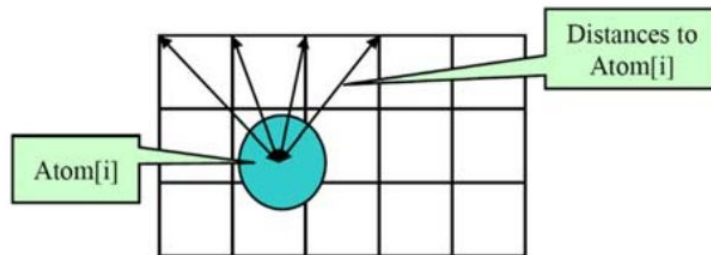
# Thread Coarsening

---

- Kernels use constant memory
  - Still, four constant memory access instructions for every nine floating-point operations are performed!
- We can use the thread coarsening technique to fuse several threads together
  - The `atoms[]` data can be fetched once from the constant memory, stored in registers, and used for multiple grid points

# Thread Coarsening

- The  $dy$  and  $dz$  distances between points on the same  $yz$  plane and any other atoms are the same
  - We can compute them once and save the results into a register
  - Also, the charge from the same atoms can be stored in a register
- The COARSEN\_FACTOR defines how many points are computer by each thread



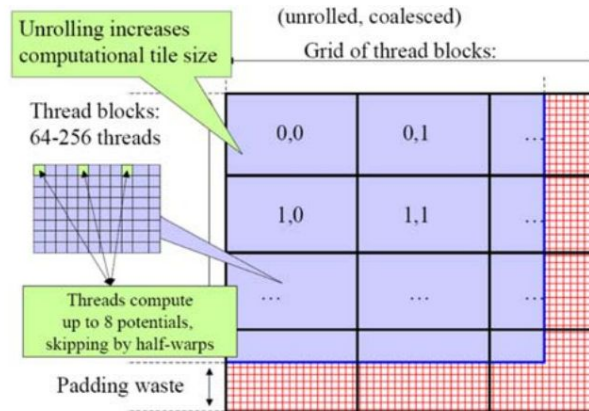
# Thread Coarsening

---

- 4 constant memory accesses and 29 floating point operations
  - A sizable reduction wrt to the previous case
- The cost of the optimization is that more registers are used by each thread
- This can potentially reduce the number of threads that can be accommodated by each SM
  - However, since the number of registers stays within the permissible limit, it does not limit the occupancy of GPU execution resources

# Memory Coalescing

- Threads perform memory writes inefficiently
  - Each thread writes four neighboring grid points
- The write pattern of adjacent threads in each warp will result in uncoalesced global memory writes
- This problem can be solved by assigning adjacent grid points to adjacent threads in each block



# Approximate Solutions: Cutoff Binning

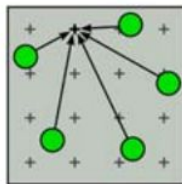
---

- Each grid point receives contributions from all atoms
  - It does not scale well to very large energy grid systems in which the number of atoms increases proportionally to the volume of the system
- The amount of computation increases with the square of the volume
  - For large-volume systems, such an increase makes the computation excessively long



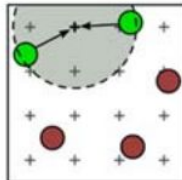
# Approximate Solutions: Cutoff Binning

- We know that each grid point needs to receive accurate contributions from atoms that are close to it
  - The atoms that are far away from a grid point will have a tiny contribution to the energy value at the grid point because the contribution is inversely proportional to the distance



**(A) Direct summation**

At each grid point, sum the electrostatic potential from all charges

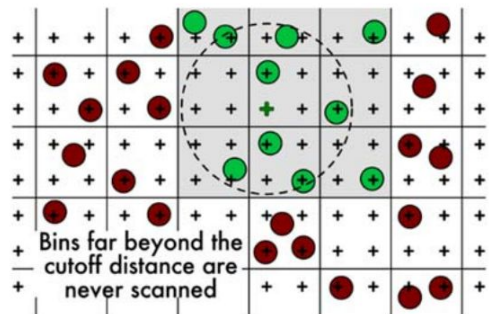


**(B) Cutoff summation**

Electrostatic potential from nearby charges summed; spatially sort charges first

# Cutoff Binning and Electrostatic Potential Map

- The key idea of the algorithm is to first sort the input atoms into bins according to their coordinates
  - Each bin corresponds to a box in the energy grid space, and it contains all atoms whose coordinates fall into the box.
- These bins are implemented as multidimensional arrays: the x, y, and z dimensions, as well as the fourth dimension that is a vector of atoms in the bin
  - We define a “neighborhood” of bins for a grid point as the collection of bins that contain all the atoms that can contribute to the energy value of the grid point



# Cutoff Binning and Load Balancing

---

- One subtle issue with binning is that bins may end up with different numbers of atoms
  - Since the atoms are statistically distributed in the grid system, some bins may have lots of atoms, and some bins may end up with no atoms at all
- To guarantee memory coalescing, it is important that all bins be of the same size and aligned at appropriate coalescing boundaries
- The binning process maintains an overflow list
  - In processing an atom, if the atom's home bin is full, the atom is added to the overflow list instead
  - The host executes a sequential cutoff algorithm on the atoms in the overflow list to complete the missing contributions from these overflow atoms

# Thanks for your attention!

**Gianmarco Accordi**  
*[gianmarco.accordi@polimi.it](mailto:gianmarco.accordi@polimi.it)*