GPU Computing

Weijie Zhao

09/13/2022

Scan

- Inclusive scan
- Exclusive scan

- Naïve scan
- Work-efficient scan

```
__global__ void reduce(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;
  }
  __syncthreads();
  if (thid == 0)
    *g_odata = temp[n-1];
}
```

```
__global__ void reduce(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;
  }
  __syncthreads();
  if (thid == 0)
    *g_odata = temp[n-1];
}
```

Dynamic shared memory allocation

reduce<<<1,nT,n>>>(d_out,d_in,n)

Shared memory size per block

Static:
__shared__ float temp[128];

```
__global__ void reduce(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;
  }
  __syncthreads();
  if (thid == 0)
    *g_odata = temp[n-1];
}
```

Dynamic shared memory allocation

reduce<<<1,nT,n>>>(d_out,d_in,n)

Shared memory size per block

Static:
__shared__ float temp[128];

Device/Host Synchronization

```
reduce<<<1,nT,n>>>(d_sum,d_array,n);
for i = 0 to logn do
        sweep_down<<<1,nT,n>>>(d_array,n);

printf(“finished\n”);
cudaMemcpy(h_array,d_array,cudaMemcpyDeviceToHost);
printf(…);
```

Device/Host Synchronization

```
reduce<<<1,nT,n>>>(d_sum,d_array,n);
for i = 0 to logn do
        sweep_down<<<1,nT,n>>>(d_array,n);
cudaDeviceSynchronize();
printf(“finished\n”);
cudaMemcpy(h_array,d_array,cudaMemcpyDeviceToHost);
printf(…);
```

CUDA Kernel Launch

- kernel_name<<<nB,nT,shared_memory_size,stream>>>(…)

- cudaStream_t stream
- cudaStreamCreate(&stream)
- cudaMemcpyAsync(dst,src,size,stream)
- cudaStreamSynchronize(stream)

- Default stream: 0

Multiple GPU Support

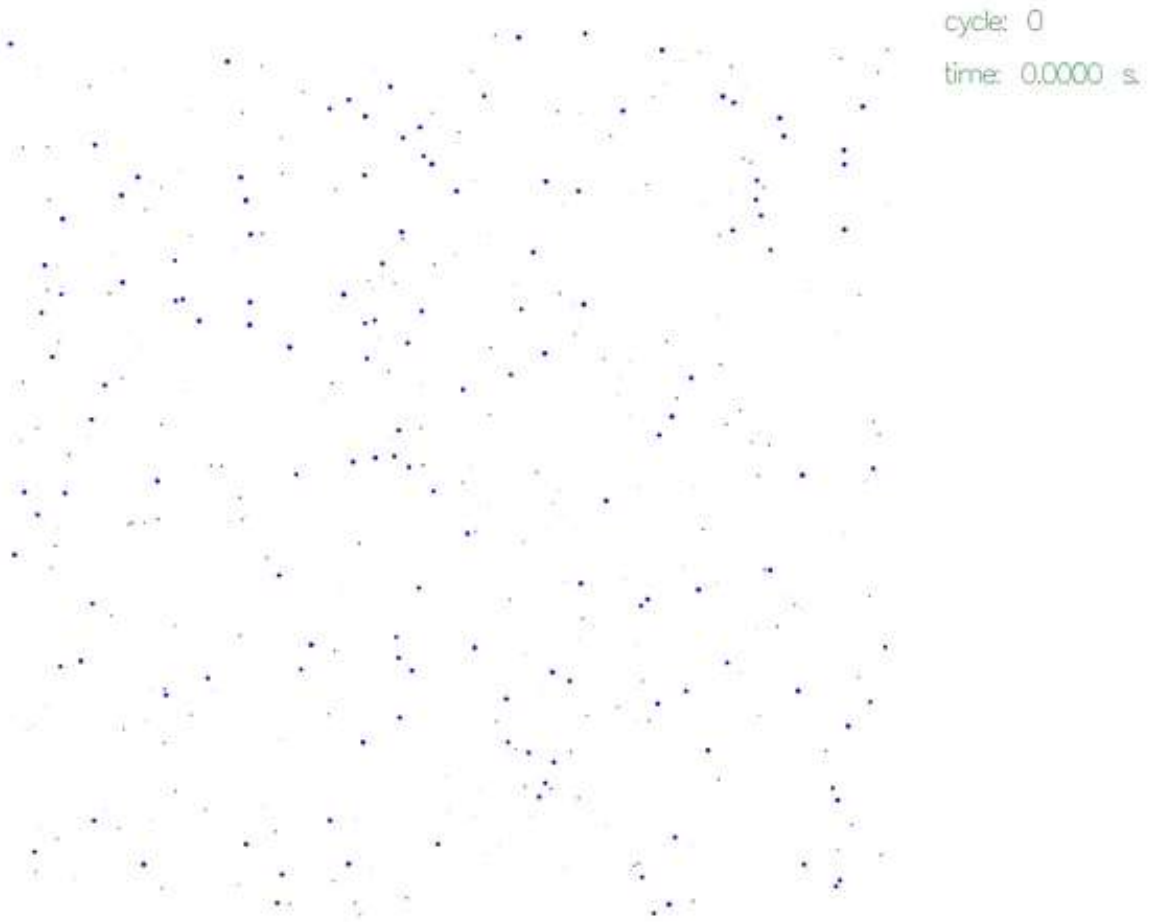
- CUDA_VISIBLE_DEVICES
- cudaError_t cudaSetDevice (int device)

- __host____device__cudaError_t cudaMalloc (void** devPtr, size_t size)
- __host__cudaError_t cudaMemcpyPeer (void* dst, int dstDevice, const void* src, int srcDevice, size_t count)
- __host__cudaError_t cudaMemcpyPeerAsync (void* dst, int dstDevice, const void* src, int srcDevice, size_t count, cudaStream_t stream = 0)

Cluster Computing

N-Body Problem

- Given N objects
  - Mass
  - Velocity
- Compute the status of each object
- Universal Gravitation
  - O(N^2) forces

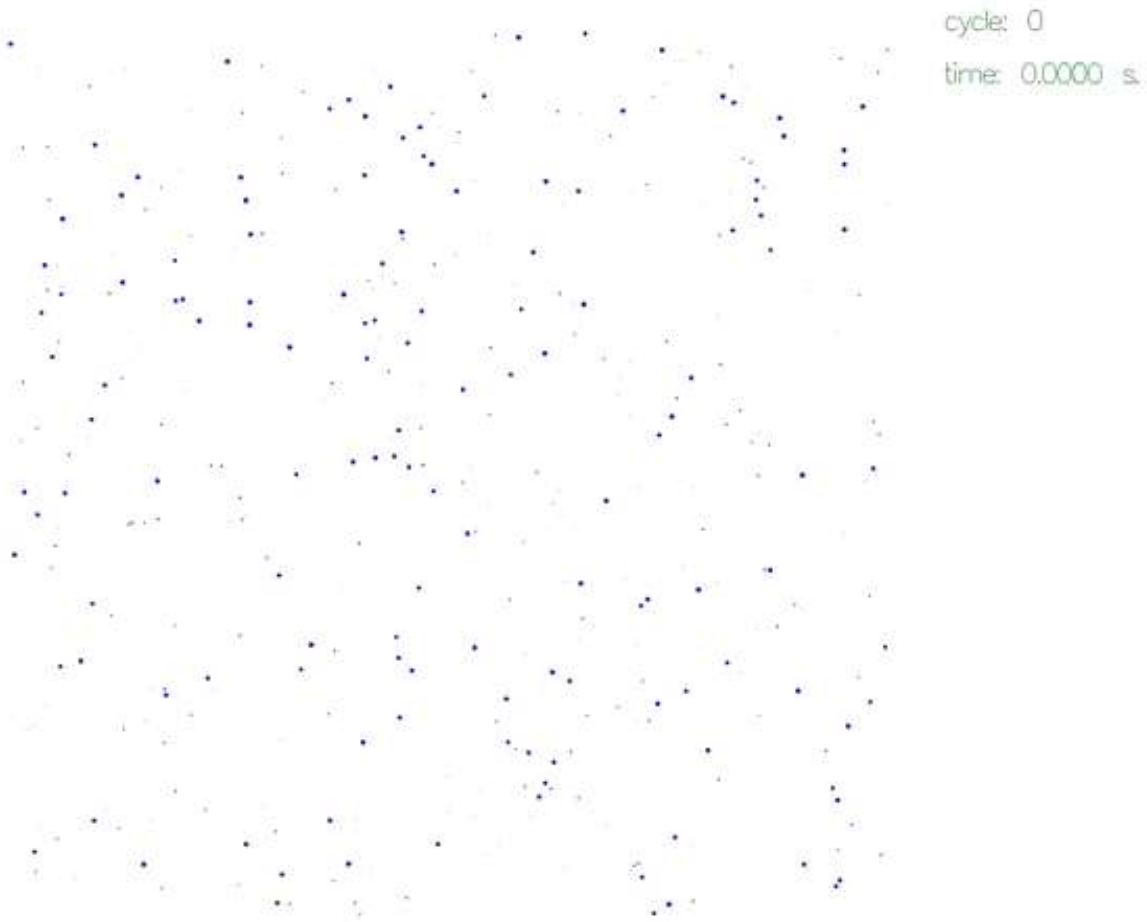


https://en.wikipedia.org/wiki/N-body_simulation

N-Body Problem

- Given N objects
  - Mass
  - Velocity
- Compute the status of each object
- Universal Gravitation
  - O(N^2) forces

- We need to scale!
- (Or have a better algorithm)



https://en.wikipedia.org/wiki/N-body_simulation

Cluster Computing

- Putting many (cheap) computers in a cluster
  - The computers in the same cluster do not have to be the same
  - Communication topology
    - All nodes are fully connected
    - Hubs
- Eventually, the communication will be the bottleneck

- For most cases, a network filesystem is employed

Message Passing Interface (MPI)

- Introduced in early 90’s
- Each process may have multiple threads
- Each process has its own address space
- Inter-process communication

MPI Example

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);
    printf(“hello world!\n");
    MPI_Finalize();
    return 0;
}
```

MPI Example

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf(“hello world from %d of %d!\n", rank, size);
    MPI_Finalize();
    return 0;
}
```

MPI Communications

```
int MPI_Send(
    void* data,
    int count,
    MPI_Datatype datatype,
    int destination,
    int tag,
    MPI_Comm communicator)
int MPI_Recv(
    void* data,
    int count,
    MPI_Datatype datatype,
    int source,
    int tag,
    MPI_Comm communicator,
    MPI_Status* status)
```

MPI Communications

```
int MPI_Probe(
    int source,
    int tag,
    MPI_Comm comm,
    MPI_Status* status)
int MPI_Get_count(
    MPI_Status* status,
    MPI_Datatype datatype,
    int* count)
```

MPI Communications

```
int MPI_Isend(
        const void *buf,
        int count,
        MPI_Datatype datatype,
        int dest,
        int tag,
        MPI_Comm comm,
        MPI_Request *request)
```

MPI Communications

```
int MPI_Wait(
        MPI_Request *request,
        MPI_Status *status)

int MPI_Test(
        MPI_Request *request,
        int *flag,
        MPI_Status *status)
```

Communicator

```
int MPI_Comm_split(
        MPI_Comm comm,
        int color,
        int key,
        MPI_Comm * newcomm)

int MPI_Comm_free(MPI_Comm *comm)
```

Compilation and Execution

- MPICH, OpenMPI
- mpicc, mpiCC, mpic++
- mpiexec, mpirun
- mpiexec -np 4 ./a.out
- mpiexec --showme

- SLURM
  - sbatch
  - srun