GPU PROGRAMMING WITH OPENACC: A LBM CASE STUDY

Jiri Kraus, JURECA Porting and Tuning Workshop, June 7. 2016
ACCELERATED COMPUTING

**CPU**
Optimized for Serial Tasks

**GPU Accelerator**
Optimized for Parallel Tasks
ACCELERATED COMPUTING

CPU Strengths

• Very large main memory
• Very fast clock speeds
• Latency optimized via large caches
• Small number of threads can run very quickly

CPU Weaknesses

• Relatively low memory bandwidth
• Cache misses very costly
• Low performance per watt
ACCELERATED COMPUTING

GPU Strengths

- High bandwidth main memory
- Latency tolerant via parallelism
- Significantly more compute resources
- High throughput
- High performance per watt

GPU Weaknesses

- Relatively low memory capacity
- Low per-thread performance
SPEED V. THROUGHPUT

Speed

Throughput

Which is better depends on your needs...
HOW GPU ACCELERATION WORKS

Application Code

CPU

GPU

Compute-Intensive Functions

Rest of Sequential CPU Code
#pragma acc data copyin(a,b) copyout(c)
{
    #pragma acc parallel
    {
        #pragma acc loop gang vector
        for (i = 0; i < n; ++i) {
            z[i] = x[i] + y[i];
            ...
        }
    }
...
Identify Available Parallelism

Optimize Loop Performance

Express Data Movement

Express Parallelism
LBM D2Q37
Lattice Boltzmann Method (LBM)

D2Q37 model

Application developed at U Rome Tore Vergata/INFN, U Ferrara/INFN, TU Eindhoven

Reproduce dynamics of fluid by simulating virtual particles which collide and propagate

Simulation of large systems requires double precision computation and many GPUs
LBM D2Q37

Versions

MPI + OpenMP + vector intrinsics using AoS data layout

MPI + OpenACC using SoA data layout (this version, starting without OpenACC directives, was used for the following)

MPI + CUDA C using SoA data layout

OpenCL

LBM D2Q37 - INITIAL VERSION

CPU Profile (480x512) - 1 MPI rank

<table>
<thead>
<tr>
<th></th>
<th>Method</th>
<th>Time (s)</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>collide</td>
<td>17.56</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>propagate</td>
<td>13.11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>bc</td>
<td>0.23</td>
<td></td>
</tr>
</tbody>
</table>

Application Reported Solve time: 30.93 s
LBM D2Q37
Change build environment

Enable OpenACC

-acc -ta=tesla

Enable Accelerator Information

-Minfo=accel
LBM D2Q37 - ACCELERATING COLLIDE

71: #pragma acc kernels pcopyin(prv[0:N]) pcopyout(nxt[0:N]) pcopyin(param[0:1])
72: #pragma acc loop independent
73: for ( ix = HX; ix < (HX+SIZEX); ix++) {
74:   #pragma acc loop independent device_type(NVIDIA) vector(LOCAL_WORK_SIZEX)
75:   for ( iy = HY; iy < (HY+SIZEY); iy++) {
76:     site_i = ix*NY + iy;
77:     rho = VZERO;
78:     v  = VZERO;
79:     u  = VZERO;
80:   #pragma acc loop independent device_type(NVIDIA) seq
82:     for( i = 0; i < NPOP; i++ ) {
83:         ...

collide:

40, include "lbm.c"
6, include "collide.h"

71, Generating copyin(prv[:9782208])
Generating copyout(nxt[:9782208])
Generating copyin(param[:1])

73, Loop is parallelizable
75, Loop is parallelizable
Accelerator kernel generated
Generating Tesla code

73, #pragma acc loop gang /* blockIdx.y */
75, #pragma acc loop gang, vector(256) /* blockIdx.x threadIdx.x */
LBM D2Q37 - COLLIDE ACCELERATED
CPU Profile (480x512) - 1 MPI rank

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (s) collide</th>
<th>Time (s) Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 collide</td>
<td>3.67</td>
<td>17.56</td>
</tr>
<tr>
<td>1 propagate</td>
<td>13.08</td>
<td>13.11</td>
</tr>
<tr>
<td>3 bc</td>
<td>0.08</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Application Reported Solvetime: 17.03 s (Initial: 30.93 s)
inline void propagate(const data_t* restrict prv, data_t* restrict nxt) {
    int ix, iy, site_i;
    #pragma acc kernels pcopyin(prv[0:NX*NY*NPOP]) pcopyout(nxt[0:NX*NY*NPOP])
    #pragma acc loop independent device_type(NVIDIA) gang
    for ( ix=HX; ix < (HX+SIZEX); ix++) {
        #pragma acc loop independent device_type(NVIDIA) vector(LOCAL_WORK_SIZEX)
        for ( iy=HY; iy < (HY+SIZEY); iy++) {
            site_i = (ix*NY) + iy;
            nxt[ site_i] = prv[ site_i - 3*NY + 1];
            nxt[ NX*NY + site_i] = prv[ NX*NY + site_i - 3*NY ];
            //...
            nxt[35*NX*NY + site_i] = prv[35*NX*NY + site_i + 3*NY ];
            nxt[36*NX*NY + site_i] = prv[36*NX*NY + site_i + 3*NY - 1];
        }
    }
}
# LBM D2Q37 - PROPAGATE ACCELERATED

**CPU Profile (480x512) - 1 MPI rank**

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (s)</th>
<th>Time (s)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+propagate</td>
<td>collide</td>
<td>Initial</td>
</tr>
<tr>
<td>1</td>
<td>collide</td>
<td>3.65</td>
<td>3.67</td>
</tr>
<tr>
<td>2</td>
<td>propagate</td>
<td>3.10</td>
<td>13.08</td>
</tr>
<tr>
<td>3</td>
<td>bc</td>
<td>0.19</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Application Reported Solvetime: 7.02 s (collide: 17.03 s)
# LBM D2Q37 - BC ACCELERATED

CPU Profile (480x512) - 1 MPI rank

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (s) +bc</th>
<th>Time (s) +propagate</th>
<th>Time (s) collide</th>
<th>Time (s) Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.65</td>
<td>3.65</td>
<td>3.67</td>
<td>17.56</td>
</tr>
<tr>
<td>3</td>
<td>3.10</td>
<td>3.10</td>
<td>13.08</td>
<td>13.11</td>
</tr>
<tr>
<td>1</td>
<td>8.18</td>
<td>0.19</td>
<td>0.08</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Application Reported Solvetime: 14.96 s (propagate: 7.02 s)
LBM D2Q37 - BC ACCELERATED

NVVP Timeline (480x512) - 1 MPI rank

$ srut -n 1 nvprof -o lbmD2Q37.bc.nvprof ./lbmD2Q37
LBM D2Q37 - BC ACCELERATED

NVVP Timeline (zoom) (480x512) - 1 MPI rank

Most of the time is spent with copying data between CPU and GPU

$ srun -n 1 nvprof -o lbmD2Q37.bc.nvprof ./lbmD2Q37
#pragma acc data copyin(f1[0:NX*NY*NPOP],param[0:1]) \
copy(f2[0:NX*NY*NPOP])

for ( i = 1; i <= NITER; i++ ) {
    ...
    propagate( f2, f1 );
    bc( f2, f1, param );
    collide( f1, f2, param );
    ...
} // For cycle over NITER
LBM D2Q37 - BC ACCELERATED

NVVP Timeline (480x512) - 1 MPI rank

No copies between CPU and GPU during simulation loop.

```bash
$ srun -n 1 nvprof -o lbmD2Q37.final.nvprof ./lbmD2Q37
```
LBM D2Q37 - FINAL
CPU Profile (480x512) - 1 MPI rank

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (s) Final</th>
<th>Time (s) Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 collide</td>
<td>0.69</td>
<td>17.56</td>
</tr>
<tr>
<td>2 propagate</td>
<td>0.11</td>
<td>13.11</td>
</tr>
<tr>
<td>3 bc</td>
<td>0.11</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Application Reported Solvetime: 1.09 s (Initial: 30.93 s)

<table>
<thead>
<tr>
<th>compiler model</th>
<th>E5-2630 v3</th>
<th>GK210</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagate perf. [GB/s]</td>
<td>ICC 15 Intrinsics</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>ICC 15 OMP</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>PGI 15.10 OACC</td>
<td>32</td>
</tr>
<tr>
<td>$\varepsilon_p$</td>
<td>65%</td>
<td>54%</td>
</tr>
<tr>
<td>collide perf. [MLUPS]</td>
<td>ICC 15 Intrinsics</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>ICC 15 OMP</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>PGI 15.10 OACC</td>
<td>12</td>
</tr>
<tr>
<td>collide perf. [GFLOPs]</td>
<td>ICC 15 Intrinsics</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>ICC 15 OMP</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>PGI 15.10 OACC</td>
<td>78</td>
</tr>
<tr>
<td>$\varepsilon_c$</td>
<td>28%</td>
<td>22%</td>
</tr>
<tr>
<td>Tot perf. [MLUPS]</td>
<td>ICC 15 Intrinsics</td>
<td>11.5</td>
</tr>
<tr>
<td></td>
<td>ICC 15 OMP</td>
<td>9.2</td>
</tr>
<tr>
<td></td>
<td>PGI 15.10 OACC</td>
<td>9.8</td>
</tr>
</tbody>
</table>
#pragma acc host_data use_device(f2)
{
    for (pp = 0; pp < NPOP; pp++) {
        MPI_Irecv(f2+(offRH+(pp*NX*NY)),3*NY,MPI_DOUBLE,rankR,0,
                   MPI_COMM_WORLD,req+pp);
        MPI_Irecv(f2+(offLH+(pp*NX*NY)),3*NY, MPI_DOUBLE,rankL,0,
                   MPI_COMM_WORLD,req+NPOP+pp);
    }
    for (pp = 0; pp < NPOP; pp++) {
        MPI_Send(f2+(offLB+(pp*NX*NY)),3*NY,MPI_DOUBLE,rankL,0,MPI_COMM_WORLD);
        MPI_Send(f2+(offRB+(pp*NX*NY)),3*NY,MPI_DOUBLE,rankR,0,MPI_COMM_WORLD);
    }
} MPI_Waitall(2*NPOP,req,MPI_STATUS_IGNORE);
int rank = 0; int size = 1;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &rank);

MPI_Comm_size(MPI_COMM_WORLD, &size);

#if _OPENACC

int ngpus = acc_get_num_devices(acc_device_nvidia);

int devicenum = rank % ngpus;

acc_set_device_num(devicenum, acc_device_nvidia);

acc_init(acc_device_nvidia);

#endif /* _OPENACC */

Alternative:

int devicenum = atoi(getenv("MPI_LOCALRANKID"));
LBM D2Q37 - MULTI GPU

Strong Scaling
LBM D2Q37 - MULTI GPU

Overlapping Communication and Computation

Grid size: 1920x2048
LBM D2Q37 - MULTI GPU
Overlapping Communication and Computation

Grid size: 1920x2048
OPENACC QUICKSTART ON JURECA

Live Demo

```bash
git clone https://github.com/NVIDIA-OpenACC-Course/nvidia-advanced-openacc-course-sources.git

module load PGI/16.3-GCC-4.9.3-2.25

module load MVAPICH2/2.2b-GDR

cd nvidia-advanced-openacc-course-sources/labs/

cd Advanced_Multi_GPU_Programming_with_MPI_and_OpenACC/C/task1/

make poisson2d

salloc -N 1 -n 4 -p develgpus --gres=gpu:4

srun ./poisson2d
```
NVIDIA APPLICATION LAB AT JÜLICH
Collaboration between JSC and NVIDIA since July 2012

Enable scientific application for GPU-based architectures
Provide support for their optimization
Investigate performance and scaling

Dirk Pleiter (JSC)  Andreas Herten (JSC)  Jiri Kraus (NVIDIA)  Andrew V. Adinetz (JSC)*

* until April 2015
OPENACC TRAINING COURSE
Introduction to GPU programming using OpenACC

Begin: 24.Oct.2016 09:00
Venue: Jülich Supercomputing Centre, Ausbildungsraum 1, building 16.3, room 213a
Target audience: Scientists who want to use GPU systems with OpenACC
Contact: Andreas Herten a.herten@fz-juelich.de
