OPTIMIZING A LBM CODE FOR COMPUTE CLUSTERS WITH KEPLER GPUS

Jiri Kraus, Developer Technology NVIDIA
LBM D2Q37

- Lattice Boltzmann Method
  - Reproduce dynamics of fluid by simulating virtual particles which collide and propagate

- Application developed at University of Rome Tor Vergata/INFN, University Ferrara/INFN, TU Eindhoven

Implementation:
F. Schifano (U Ferrara)
LBM D2Q37

- Very accurate D2Q37 model
  - correctly reproduces the thermo-hydrodynamical equations of motion of a fluid
  - automatically enforces the equation of state of a perfect gas
- Used e.g. to study Rayleigh Taylor instability
LBM D2Q37

- Basically two massively parallel steps:
  - Collide: very compute intensive for D2Q37 Model more than 6000 double precision flops per site
  - Propagate: Memory bandwidth bound

- Initial version was optimized for Fermi

<table>
<thead>
<tr>
<th></th>
<th>M2090</th>
<th>K20X</th>
</tr>
</thead>
<tbody>
<tr>
<td>collide</td>
<td>134 ms</td>
<td>86 ms</td>
</tr>
<tr>
<td>propagate</td>
<td>21 ms</td>
<td>16 ms</td>
</tr>
</tbody>
</table>

Why does the collide performance only improve by 1.5x although the compute throughput increased by about 2x?
INITIAL SINGLE GPU PERFORMANCE

- `nvprof -analysis-metrics -o <filename>`
- Import `<filename>` into nvvp
- Run guided Analysis
INITIAL SINGLE GPU PERFORMANCE

Kepler:

ptxas info : Compiling entry function '_Z7collidePdS_S_' for 'sm_35'
ptxas info : Function properties for _Z7collidePdS_S_

296 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info : Used 88 registers, 344 bytes cmem[0], 68 bytes cmem[2]

Fermi:

ptxas info : Compiling entry function '_Z7collidePdS_S_' for 'sm_20'
ptxas info : Function properties for _Z7collidePdS_S_

800 bytes stack frame, 652 bytes spill stores, 1328 bytes spill loads
ptxas info : Used 63 registers, 56 bytes cmem[0], 44 bytes cmem[16]

Less local memory usage on Kepler than on Fermi.
Why is Kepler affected by local memory overhead anyway?
THE KEPLER SMX

Kepler has less L1(Shared Memory) per CUDA core than Fermi, but a much larger register file

<table>
<thead>
<tr>
<th>Resource</th>
<th>Kepler GK110 vs. Fermi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point throughput</td>
<td>2-3x</td>
</tr>
<tr>
<td>Max blocks per SMX</td>
<td>2x</td>
</tr>
<tr>
<td>Max threads per SMX</td>
<td>1.3x</td>
</tr>
<tr>
<td>Register file bandwidth</td>
<td>2x</td>
</tr>
<tr>
<td>Register file capacity</td>
<td>2x</td>
</tr>
<tr>
<td>Shared memory bandwidth</td>
<td>2x</td>
</tr>
<tr>
<td>Shared memory capacity</td>
<td>1x</td>
</tr>
</tbody>
</table>
ANALYZING THE COLLIDE KERNEL (INITIAL)

#K20X, CUDA 5.5 Driver 319.72, ECC on
$ nvprof --metrics l1_cache_local_hit_rate ./lbmD2Q37
==12419== Profiling application: ./lbmD2Q37
==12419== Profiling result:
==12419== Metric result:
Invocations   Metric Name              Metric Description         Min         Max         Avg
Device "Tesla K20Xm (0)"
Kernel: propagate (double*, double*)
  100         l1_cache_local_hit_rate               L1 Local Hit Rate       0.00%       0.00%       0.00%
Kernel: collide(double*, double*, double*)
  100         l1_cache_local_hit_rate               L1 Local Hit Rate      19.52%      19.54%      19.53%

<table>
<thead>
<tr>
<th></th>
<th>M2090</th>
<th>K20X</th>
</tr>
</thead>
<tbody>
<tr>
<td>collide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1$ Hit rate</td>
<td>53.43%</td>
<td>19.53%</td>
</tr>
</tbody>
</table>

More threads are competing for the same amount of L1$
OPTIMIZE LBM CODE FOR THE KEPLER SMX

- Each thread has a thread private array localPop to save roundtrips to DRAM
  - This local memory array is cached in L1 for Fermi and Kepler
- Try to utilize larger register file of the Kepler SMX
  - Cache in registers instead of L1
  - Unrolling all loops accessing localPop allows the compiler to use registers for localPop

```c
for(int i = 0; i < NPOP; i++) {
    popTemp = localPop[i];
    rho = rho + popTemp;
    u = u * popTemp + param_cx[i];
    v = v * popTemp + param_cy[i];
}
```

```c
#pragma unroll
for(int i = 0; i < NPOP; i++) {
    popTemp = localPop[i];
    rho = rho + popTemp;
    u = u * popTemp + param_cx[i];
    v = v * popTemp + param_cy[i];
}
```
OPTIMIZE LBM CODE FOR THE KEPLER SMX

for (int i = 0; i < NPOP; i++) {
    popTemp = localPop[i];
    rho = rho + popTemp;
    u = u * popTemp + param_cx[i];
    v = v * popTemp + param_cy[i];
}

#pragma unroll
for (int i = 0; i < NPOP; i++) {
    popTemp = localPop[i];
    rho = rho + popTemp;
    u = u * popTemp + param_cx[i];
    v = v * popTemp + param_cy[i];
}

<table>
<thead>
<tr>
<th>collide runtime K20X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial version</td>
</tr>
<tr>
<td>Unrolled version</td>
</tr>
</tbody>
</table>

What went wrong?
ANALYZING THE COLLIDE KERNEL (UNROLLED)

#K20X, CUDA 5.5 Driver 319.72, ECC on, With unrolled loops
$ nvprof --metrics l1_cache_local_hit_rate,achieved_occupancy ./lbmD2Q37

Profiling application: ./lbmD2Q37

Profiling result:

Metric result:

<table>
<thead>
<tr>
<th>Invocations</th>
<th>Metric Name</th>
<th>Metric Description</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device &quot;Tesla K20Xm (0)&quot;</td>
<td>Kernel: propagate(double*, double*)</td>
<td>L1 Local Hit Rate</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>100</td>
<td>l1_cache_local_hit_rate</td>
<td>Achieved Occupancy</td>
<td>0.920924</td>
<td>0.927788</td>
<td>0.924370</td>
</tr>
<tr>
<td></td>
<td>achieved_occupancy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel: collide(double*, double*, double*)</td>
<td>100</td>
<td>l1_cache_local_hit_rate</td>
<td>L1 Local Hit Rate</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>achieved_occupancy</td>
<td>Achieved Occupancy</td>
<td>0.124922</td>
<td>0.124924</td>
<td>0.124923</td>
</tr>
</tbody>
</table>

Heavy register usage limits occupancy.

<table>
<thead>
<tr>
<th></th>
<th>L1$ Hit rate</th>
<th>Achieved Occupancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial version</td>
<td>19.53%</td>
<td>24.5%</td>
</tr>
<tr>
<td>Unrolled version</td>
<td>0.0%</td>
<td>12.5%</td>
</tr>
</tbody>
</table>
OPTIMIZE LBM CODE FOR THE KEPLER SMX

- How can we make use of the larger register file and maintain the achieved occupancy?
  - Use __launch_bounds__

```c
__launch_bounds__(256, 2)
__global__ void collide(
    data_t * __restrict__ const p_nxt,
    const data_t * __restrict__ const p_prv
) {
```
**OPTIMIZE LBM CODE FOR THE KEPLER SMX**

<table>
<thead>
<tr>
<th></th>
<th>L1$ Hit rate</th>
<th>Achieved Occupancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial version</td>
<td>19.53%</td>
<td>24.5%</td>
</tr>
<tr>
<td>Unrolled version</td>
<td>0.0%</td>
<td>12.5%</td>
</tr>
<tr>
<td>Unrolled + Launch bounds version</td>
<td>27.87%</td>
<td>24.9%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>collide runtime K20X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial version</td>
<td>86 ms</td>
</tr>
<tr>
<td>Unrolled version</td>
<td>91 ms</td>
</tr>
<tr>
<td>Unrolled + Launch bounds version</td>
<td>60 ms</td>
</tr>
</tbody>
</table>

Can we improve further?
ANALYZING THE COLLIDE KERNEL

- Use line-based profiling/SASS inspection to identify further performance improvements
  - Compile with "-lineinfo"
  - Use NVVP feature (new with CUDA 6) side-by-side source and disassembly view accessible via the “Kernel Profile” experiment
**ANALYZING THE COLLIDE KERNEL**

<table>
<thead>
<tr>
<th>Line</th>
<th>Exec Count</th>
<th>File: /home/jkrus/workspace/LBM/single_gpu/collide_kernel.cuh</th>
<th>Disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>175</td>
<td>51328000</td>
<td>modc2 = (param_cx[i][j][k]</td>
<td>DDMX R58, R56, R56, R60;</td>
</tr>
<tr>
<td>176</td>
<td></td>
<td>+ param_cy[i][j][k]) * param</td>
<td>LDD.LU R71, R41+0x218;</td>
</tr>
<tr>
<td>177</td>
<td>47232000</td>
<td>H2x[i][j][k]) + projection2xx +</td>
<td>DDMX R8, R56, R26, R84;</td>
</tr>
<tr>
<td>178</td>
<td></td>
<td>VTHRE * param_H2y[i][j][k] * projection2yy +</td>
<td>STL R41=0x1c0, R9;</td>
</tr>
<tr>
<td>179</td>
<td></td>
<td>param_H2y[i][j][k] * projection2yy;</td>
<td>DADD R22, R18, 8;</td>
</tr>
<tr>
<td>181</td>
<td>79232000</td>
<td>Hermite3 = param_H3xxx[i][j][k] * projection3xxx +</td>
<td>STL R41=0x1c8, R8;</td>
</tr>
<tr>
<td>182</td>
<td></td>
<td>VTHRE * param_H3xyy[i][j][k] * projection3xyy +</td>
<td>DMUL R18, R26, R18;</td>
</tr>
<tr>
<td>183</td>
<td></td>
<td>VTHRE * param_H3xyy[i][j][k] * projection3xyy +</td>
<td>DMMX R88, R56, R22, R84;</td>
</tr>
<tr>
<td>184</td>
<td></td>
<td>param_H3yyy[i][j][k] * projection3yyy;</td>
<td>NOV321 R120, 0x0;</td>
</tr>
<tr>
<td>186</td>
<td>117888000</td>
<td>Hermite4 = param_H4xxxx[i][j][k][k] * projection4xxxx +</td>
<td>NOV321 R85, 0x0;</td>
</tr>
<tr>
<td>187</td>
<td></td>
<td>VFOUR * param_H4xxx[i][j][k][k] * projection4xxx +</td>
<td>NOV321 R84, R56;</td>
</tr>
<tr>
<td>188</td>
<td></td>
<td>VSIXS * param_H4xxxy[i][j][k][k] * projection4xxxy +</td>
<td>NOV321 c[0x3][0x1w0];</td>
</tr>
<tr>
<td>189</td>
<td></td>
<td>VFOUR * param_H4xxxy[i][j][k][k] * projection4xxxy +</td>
<td>NOV321 c[0x3][0x1e4];</td>
</tr>
<tr>
<td>190</td>
<td></td>
<td>param_H4yyxy[i][j][k][k] * projection4yyxy;</td>
<td>STL R41=0x1c0, R9;</td>
</tr>
<tr>
<td>191</td>
<td>24832000</td>
<td>eqHermite2 = rho *</td>
<td>DADD R22, R18, R8, R36;</td>
</tr>
<tr>
<td>194</td>
<td></td>
<td>(scalar * scalar - modc2) +</td>
<td>DMMX R8, c[0x3][0x9e8], R22;</td>
</tr>
<tr>
<td>195</td>
<td></td>
<td>(temp - VONES) * (modc2 - VTWOS)</td>
<td>STL R41=0x1c0, R36;</td>
</tr>
<tr>
<td>197</td>
<td>34304000</td>
<td>eqHermite3 = rho *</td>
<td>DMMX R88, R8, 0;</td>
</tr>
<tr>
<td>199</td>
<td></td>
<td>(scalar * scalar -</td>
<td>DMUL R18, R120, c[0x3][0x40c0];</td>
</tr>
<tr>
<td>200</td>
<td></td>
<td>VTHRE * modc2 +</td>
<td>DMUL R60, R120, c[0x3][0x40c0];</td>
</tr>
<tr>
<td>201</td>
<td></td>
<td>VTHRE * (theta - VONES) * (modc2 - VFOUR)</td>
<td>STL R41=0x1c0, R37;</td>
</tr>
<tr>
<td>203</td>
<td></td>
<td>)</td>
<td>DMMX R8, c[0x3][0x9e8], R22;</td>
</tr>
<tr>
<td>204</td>
<td></td>
<td>);</td>
<td>STL R41=0x1c0, R36;</td>
</tr>
<tr>
<td>205</td>
<td></td>
<td></td>
<td>DMMX R88, R8, 0;</td>
</tr>
</tbody>
</table>

---

**GPU TECHNOLOGY CONFERENCE**
ANALYZING THE COLLIDE KERNEL

- Reorder expressions to mathematically equivalent formulations
  - Compiler can recognize constants and extract them from the loop
ANALYZING THE COLLIDE KERNEL

was 117888000  
(4x improvement)
### Optimize LBM Code for the Kepler SMX

<table>
<thead>
<tr>
<th></th>
<th>L1$ Hit rate</th>
<th>Achieved Occupancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial version</td>
<td>19.53%</td>
<td>24.5%</td>
</tr>
<tr>
<td>Unrolled version</td>
<td>0.0%</td>
<td>12.5%</td>
</tr>
<tr>
<td>Unrolled + Launch bounds version</td>
<td>27.87%</td>
<td>24.9%</td>
</tr>
<tr>
<td>Final version</td>
<td>33.93%</td>
<td>24.4%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>collide runtime K20X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial version</td>
<td>86 ms</td>
</tr>
<tr>
<td>Unrolled version</td>
<td>91 ms</td>
</tr>
<tr>
<td>Unrolled + Launch bounds version</td>
<td>60 ms</td>
</tr>
<tr>
<td>Final</td>
<td>51 ms</td>
</tr>
</tbody>
</table>
OPTIMIZE LBM CODE FOR THE KEPLER SMX

Kernel Performance Is Bound By Compute

For device “Tesla K20Xm” the kernel's memory utilization is significantly lower than its compute utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by computation on the SMs.
MULTI GPU VERSION OF D2Q37

- The code works on a 2D domain with periodic boundaries on the left and right boundary
- Multi GPU version applies a 1D domain decomposition with 1 GPU per MPI rank.
- MPI ranks/GPUs are arranged in a ring
**MULTI GPU VERSION OF D2Q37**

- MPI communication runs concurrently with the propagate step
  - Communication time is hidden if \( t(\text{propagate}) \geq t(\text{MPI}) \)
MULTI GPU PERFORMANCE

- 16384x256 and 256x16384 Grid
  - 256x16384 Chosen to be dominated by communication in the propagate step
- 4 Node Cluster
- FDR Infiniband
- 2 K20X per Node
- 2 Intel Xeon E5-2650@2.00GHz per node
- MVAPICH2 2.0b
- CUDA Driver 331.20
- CUDA 5.5
MULTI GPU SCALABILITY

16384x256 - strong scaling

Speedup vs. 1 GPU

LBM D2Q37  Linear
MULTI GPU SCALABILITY

256x16384 - strong scaling
MULTI GPU VERSION OF D2Q37

- MPI communication runs concurrently with the propagate step
  - Communication time is only partially hidden if $t(\text{propagate}) < t(\text{MPI})$

![Diagram showing parallel streams for propagation and MPI communication](image-url)
OPTIMIZE THE MULTI GPU VERSION

cudAmemcpy(sendbuf_h, sendbuf_d, N*sizeof(double),
cudAmemcpyDeviceToHost);

MPI_Sendrecv(
    sendbuf_h, N, MPI_DOUBLE, mpi_nxt, 0,
    recvbuf_h, N, MPI_DOUBLE, mpi_prv, 0,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE
);
cudAmemcpy(recvbuf_d, recvbuf_h, N*sizeof(double),
cudAmemcpyHostToDevice);

#pragma omp parallel for
    for (int i = 0; i < N; i++)
    
        cudaMemcpy(sendbuf_h[i], sendbuf_d[i],
            cudaMemcpyHostToDevice);

// No Host staging needed

MPI_Sendrecv(
    sendbuf_d, N, MPI_DOUBLE, mpi_nxt, 0,
    recvbuf_d, N, MPI_DOUBLE, mpi_prv, 0,
    MPI_COMM_WORLD, MPI_STATUS_IGNORE
);

// No Host staging needed

More Details:

- Recording of GTC 2014 Session S4236 Multi GPU Programming with MPI (Part I+II+III) was on Monday 03/24
MULTI GPU SCALABILITY

256x16384 - strong scaling

Speedup vs. 1 GPU

1 GPU  2 GPUs  4 GPUs  8 GPUs

Regular MPI  CUDA-aware MPI

1.35x
CONCLUSIONS

- Single GPU Performance Matters
- Communication time should be hidden
- CUDA-aware MPI can help to improve Multi-GPU scaling

This work was carried out in the scope of the NVIDIA Application Lab@Jülich

http://www.fz-juelich.de/ias/jsc/nvlab