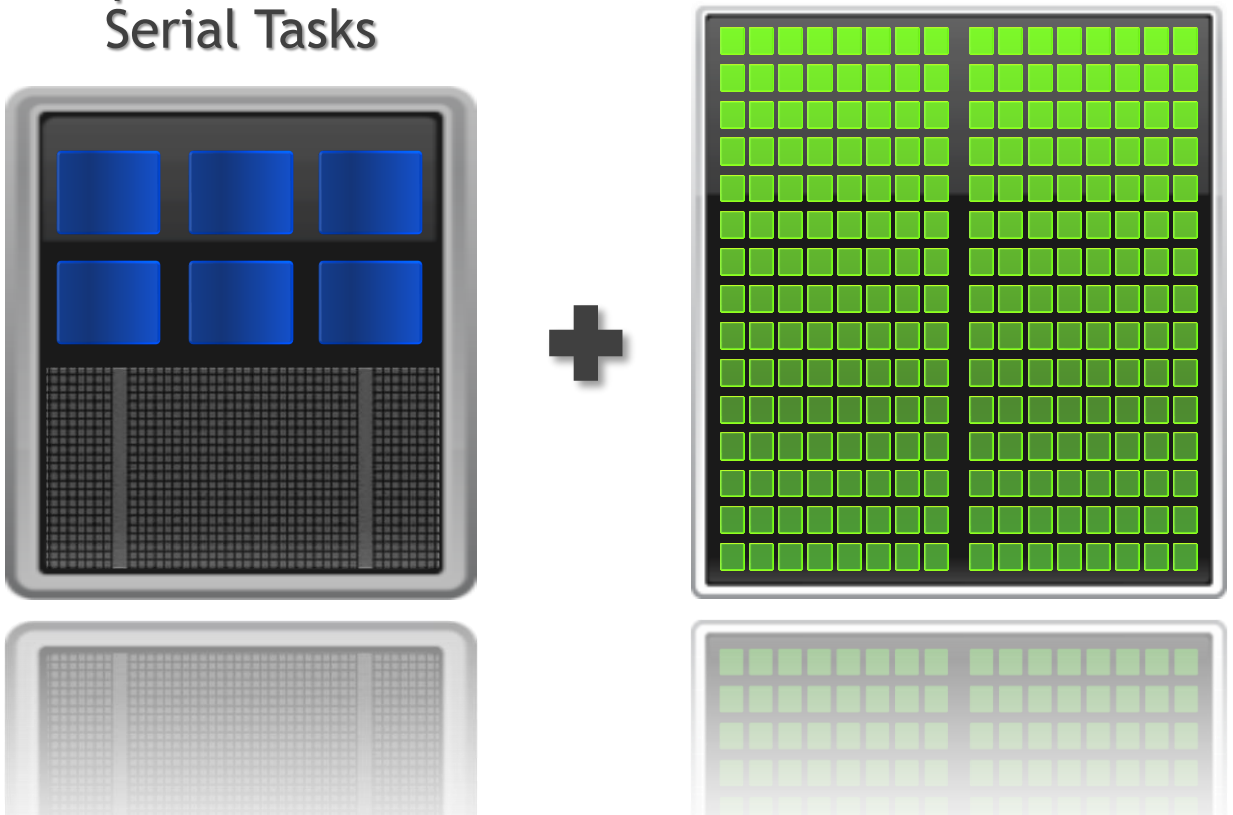
GPU PROGRAMMING WITH OPENACC: A LBM CASE STUDY

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

ACCELERATED COMPUTING

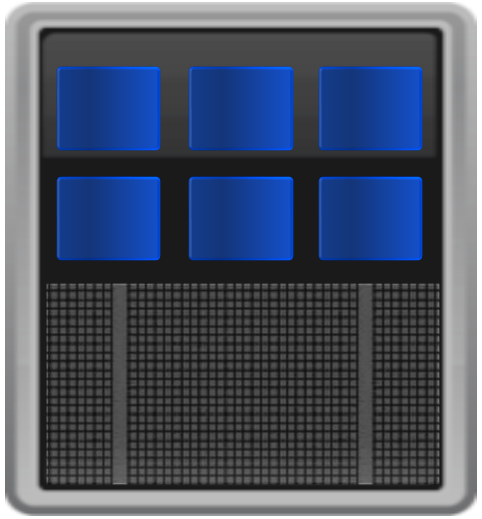
CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks



ACCELERATED COMPUTING

CPU
Optimized for
Serial Tasks



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

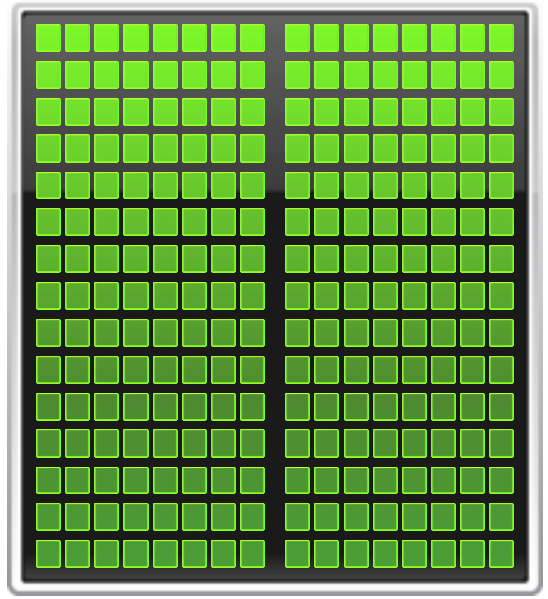
GPU Strengths

- 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

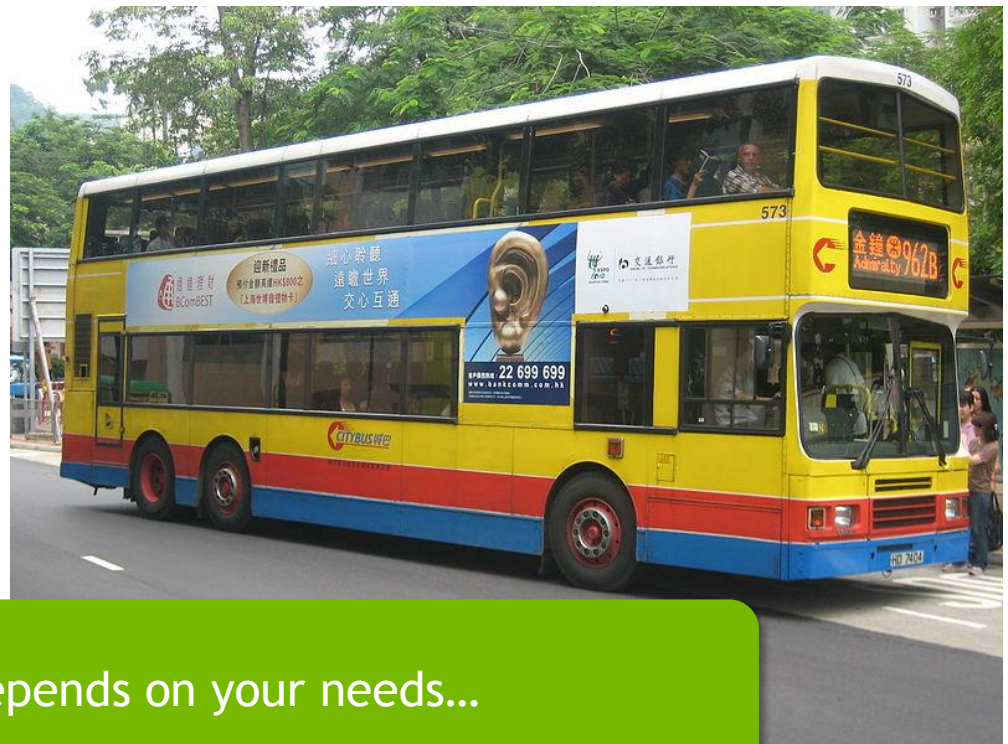
GPU Accelerator
Optimized for
Parallel Tasks



SPEED V. THROUGHPUT

Speed

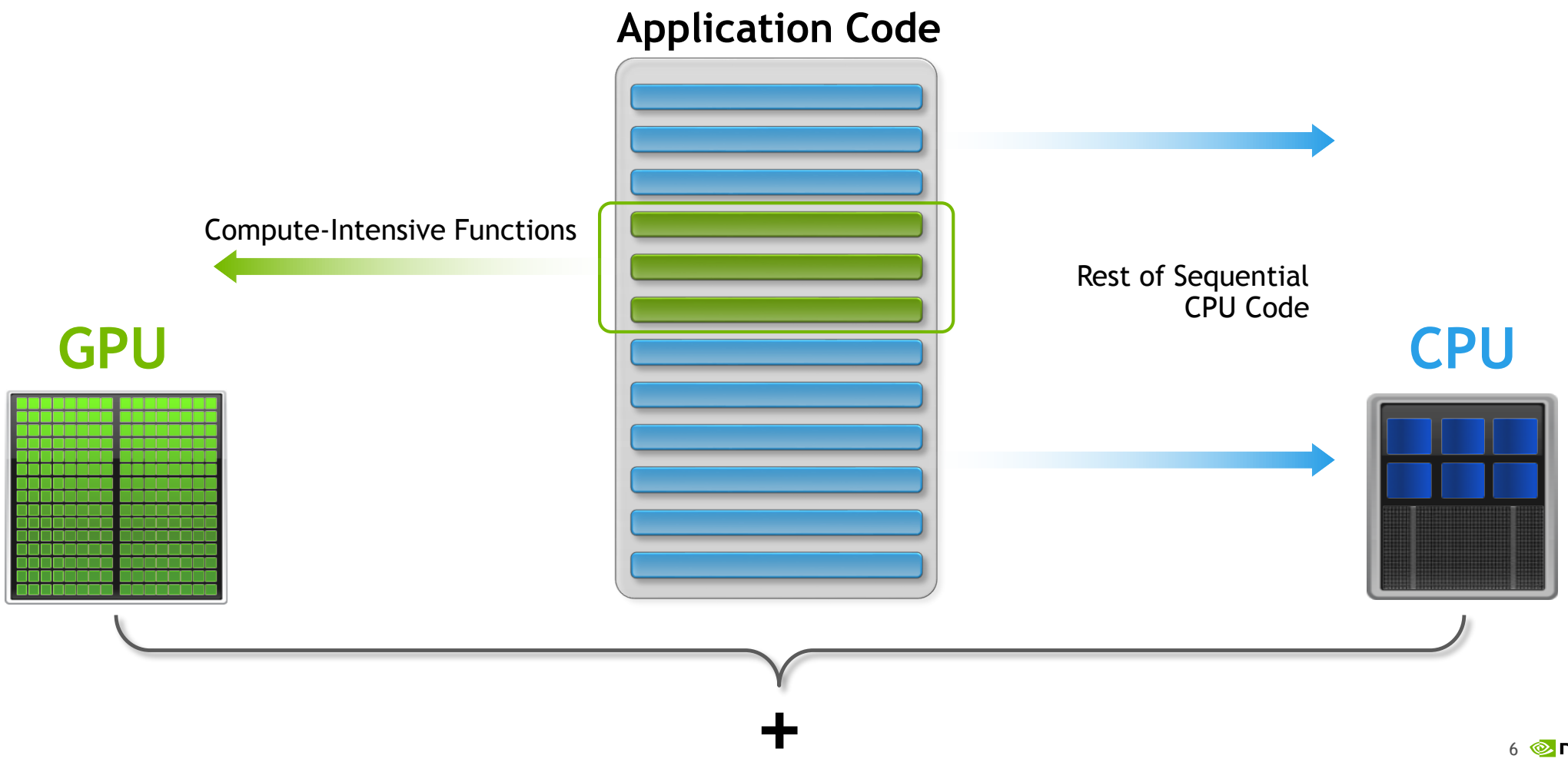
Throughput



Which is better depends on your needs…

*Images from Wikimedia Commons via Creative Commons

HOW GPU ACCELERATION WORKS



OPENACC DIRECTIVES

Manage Data Movement

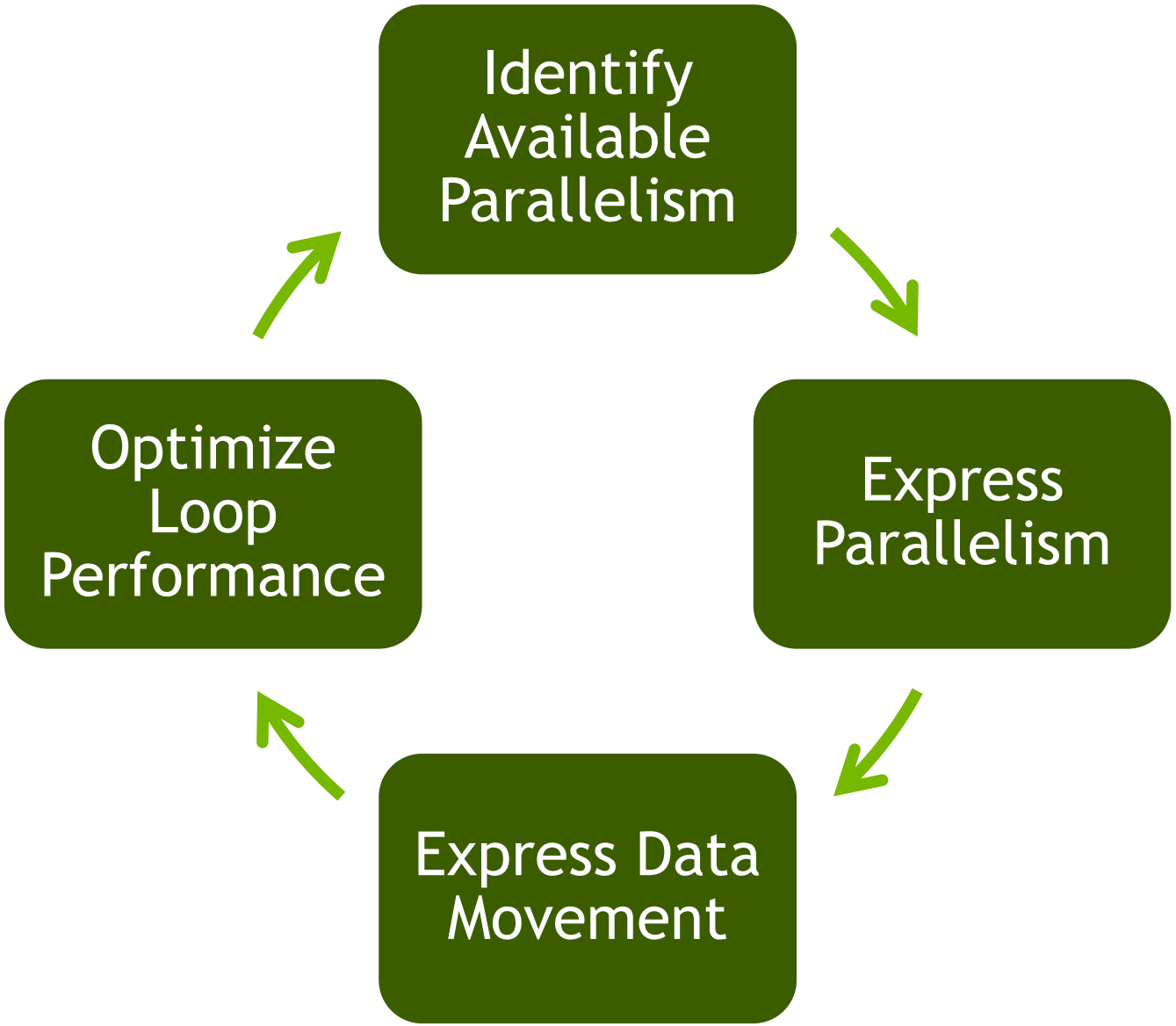
Initiate Parallel Execution

Optimize Loop Mappings

```
#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];
      ...
    }
  }
  ...
}
```

OpenACC
Directives for Accelerators

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, MIC


Identify Available Parallelism
Express Parallelism
Express Data Movement
Optimize Loop Performance

LBM D2Q37

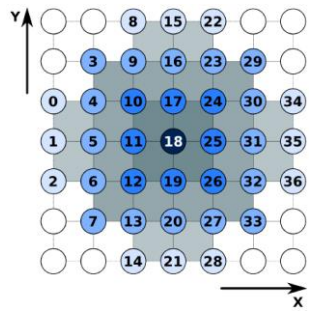
Lattice Boltzmann Method (LBM)

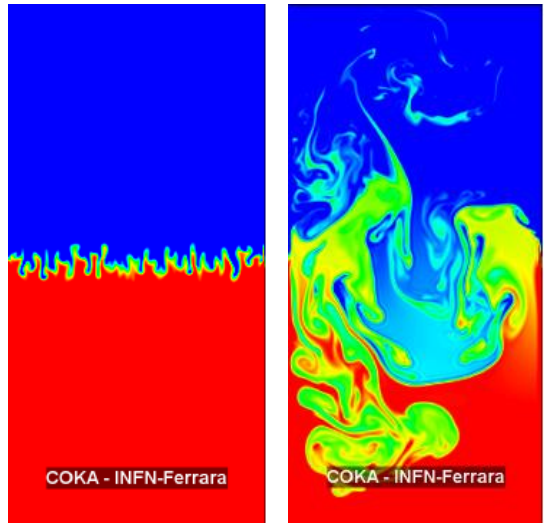
D2Q37 model

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

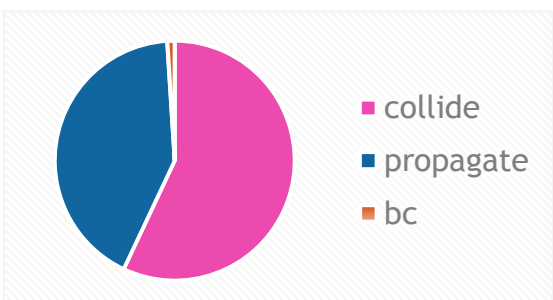
OpenCL

Paper comparing these variants: Calore, E., Gabbana, A., Kraus, J., Schifano, S. F., & Tripiccione, R. (2016). Performance and portability of accelerated lattice Boltzmann applications with OpenACC. *Concurrency and Computation: Practice and Experience*.

LBM D2Q37 – INITIAL VERSION

CPU Profile (480x512) – 1 MPI rank

	Method	Time (s) Initial
1	collide	17.56
2	propagate	13.11
3	bc	0.23



Application Reported Solvertime: 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++ ) {
         ...
```

LBM D2Q37 - ACCELERATING COLLIDE

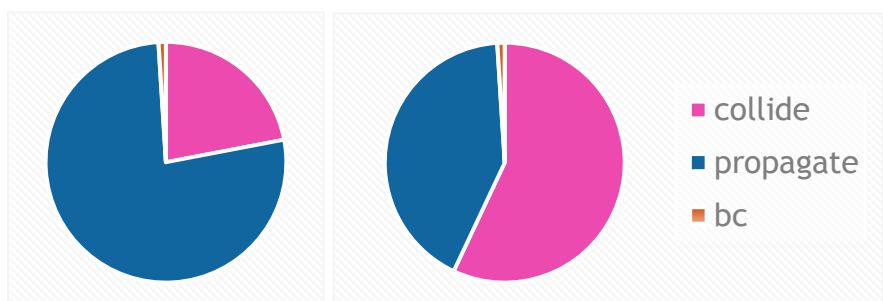
Compiler Feedback (-Minfo=accel)

```
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

	Method	Time (s) collide	Time (s) Initial
2	`collide`	3.67	17.56
1	`propagate`	13.08	13.11
3	`bc`	0.08	0.23



Application Reported Solvertime: 17.03 s (Initial: 30.93 s)

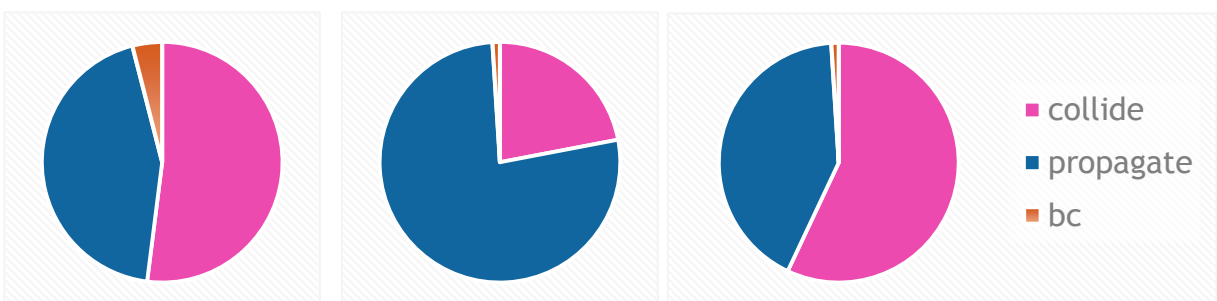
LBM D2Q37 - ACCELERATING PROPAGATE

```
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

	Method	Time (s) +propagate	Time (s) collide	Time (s) Initial
1	`collide`	3.65	3.67	17.56
2	`propagate`	3.10	13.08	13.11
3	`bc`	0.19	0.08	0.23

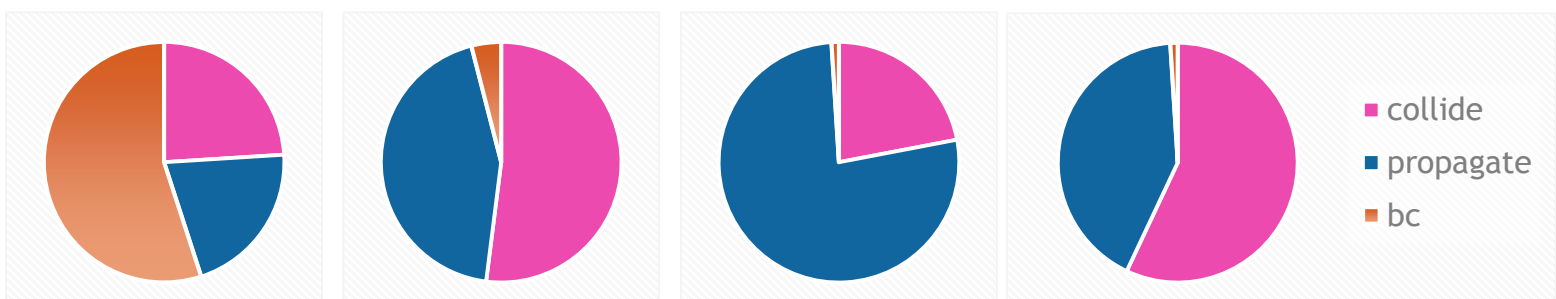


Application Reported Solvertime: 7.02 s (collide: 17.03 s)

LBM D2Q37 – BC ACCELERATED

CPU Profile (480x512) – 1 MPI rank

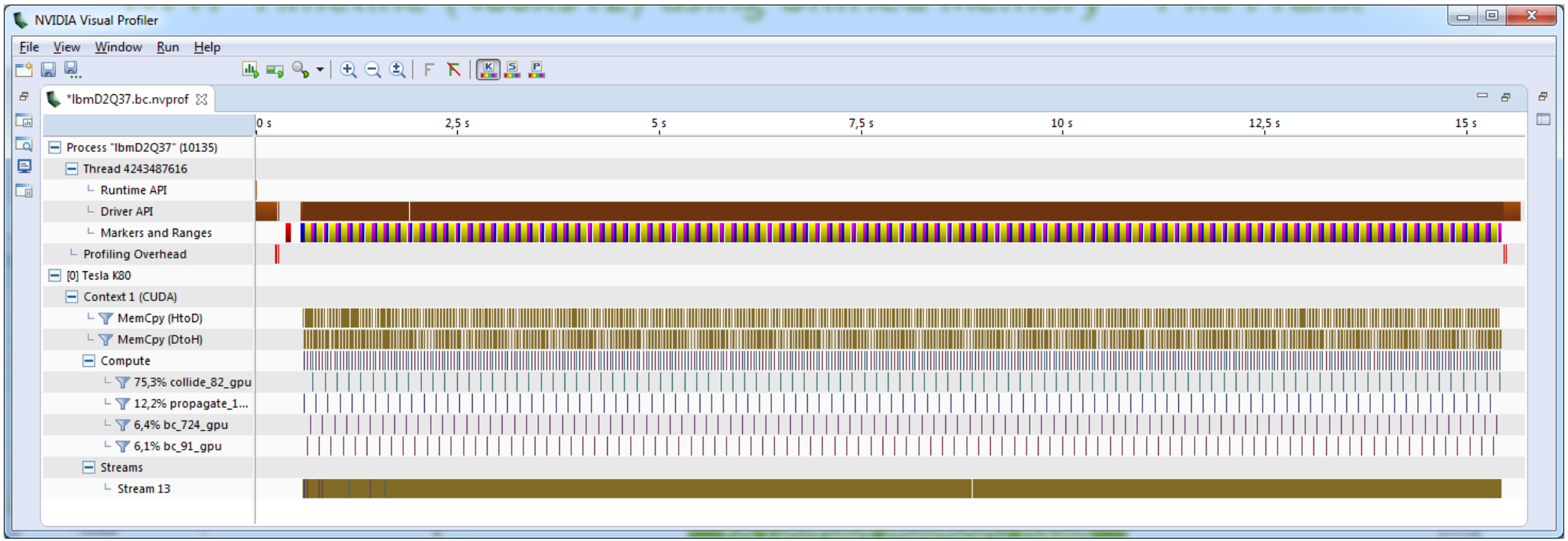
	Method	Time (s) +bc	Time (s) +propagate	Time (s) collide	Time (s) Initial
2	collide	3.65	3.65	3.67	17.56
3	propagate	3.10	3.10	13.08	13.11
1	bc	8.18	0.19	0.08	0.23



Application Reported Solvertime: 14.96 s (propagate: 7.02 s)

LBM D2Q37 - BC ACCELERATED

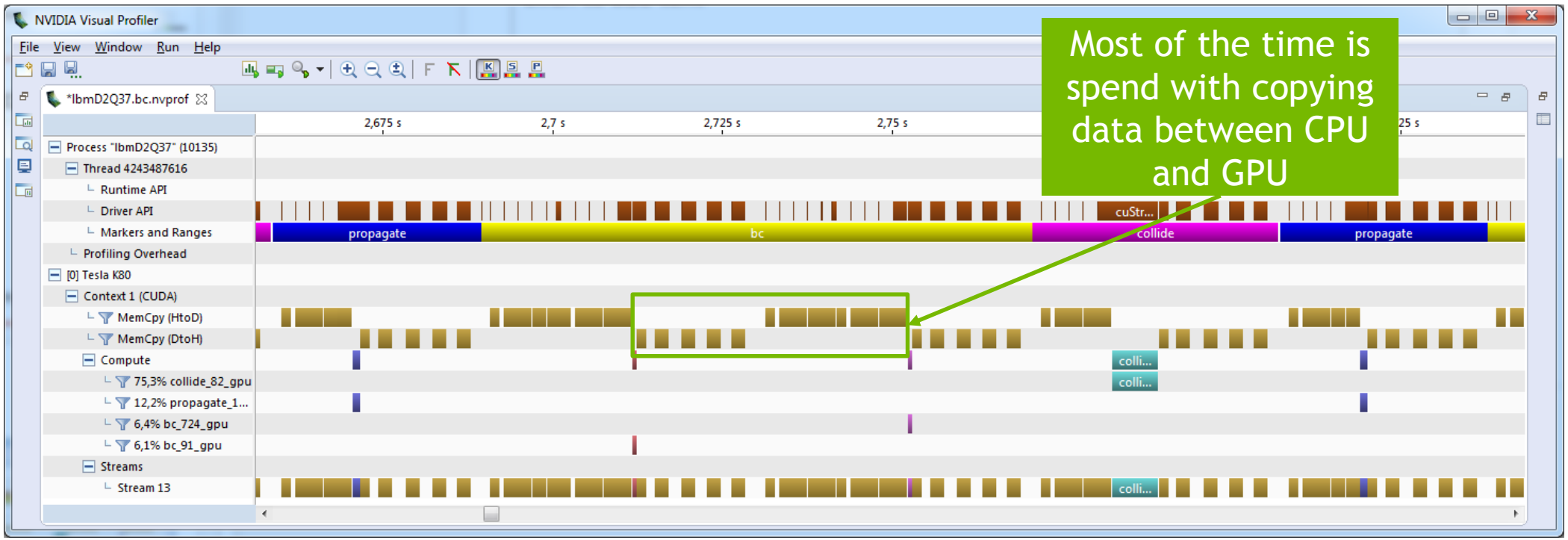
NVVP Timeline (480x512) - 1 MPI rank




```
$ srun -n 1 nvprof -o lbmD2Q37.bc.nvprof ./lbmD2Q37
```

LBM D2Q37 - BC ACCELERATED

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



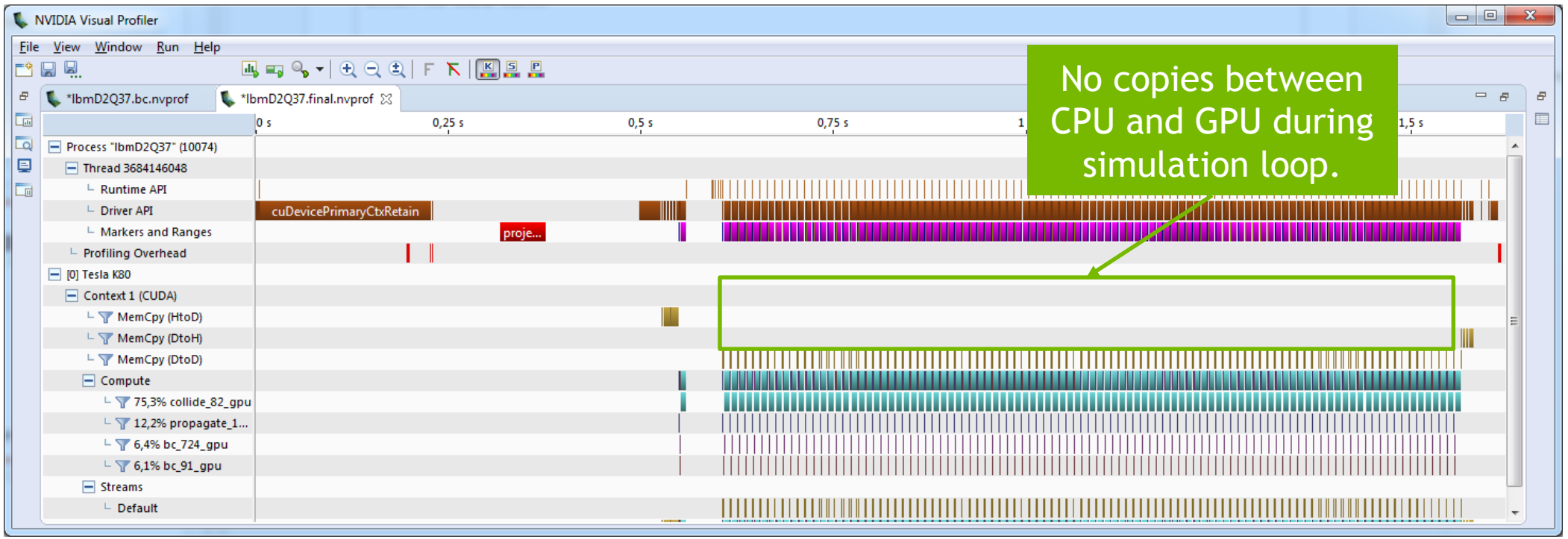

```
$ srun -n 1 nvprof -o lbmD2Q37.bc.nvprof ./lbmD2Q37
```

LBM D2Q37 - OUTER DATA REGION

```
#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

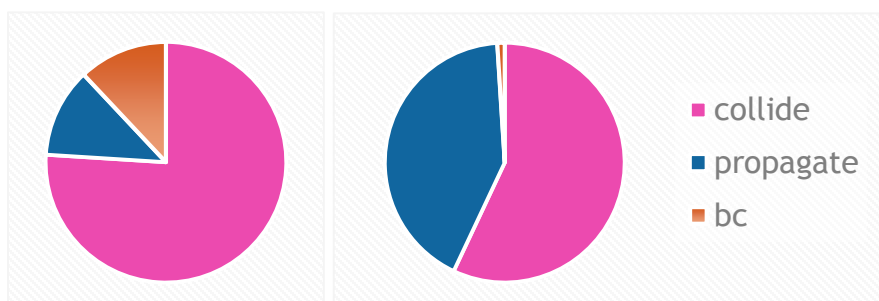



```
$ srun -n 1 nvprof -o lbmD2Q37.final.nvprof ./lbmD2Q37
```

LBM D2Q37 - FINAL

CPU Profile (480x512) – 1 MPI rank

	Method	Time (s) Final	Time (s) Initial
1	collide	0.69	17.56
2	propagate	0.11	13.11
3	bc	0.11	0.23



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

LBM D2Q37 – FINAL

1920x2048 - 1 multi-threaded MPI rank

	E5-2630 v3			GK210	
compiler model	ICC 15 Intrinsics	ICC 15 OMP	PGI 15.10 OACC	NVCC 7.5 CUDA	PGI 15.10 OACC
`propagate` perf. [GB/s]	38	32	32	154	155
$\mathcal{E}_p$	65%	54%	54%	64%	65%
`collide` perf. [MLUPS]	14	11	12	117	55
`collide` perf. [GFLOPs]	92	71	78	760	356
$\mathcal{E}_c$	28%	22%	24%	52%	24%
Tot perf. [MLUPS]	11.5	9.2	9.8	80.7	45.6

Calore, E., Gabbana, A., Kraus, J., Schifano, S. F., & Tripiccione, R. (2016). Performance and portability of accelerated lattice Boltzmann applications with OpenACC. *Concurrency and Computation: Practice and Experience*.

LBM D2Q37 - MULTI GPU

Inter GPU communication with CUDA-aware MPI

```
#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);
```

LBM D2Q37 - MULTI GPU

Handling GPU AFFINITY

```
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:

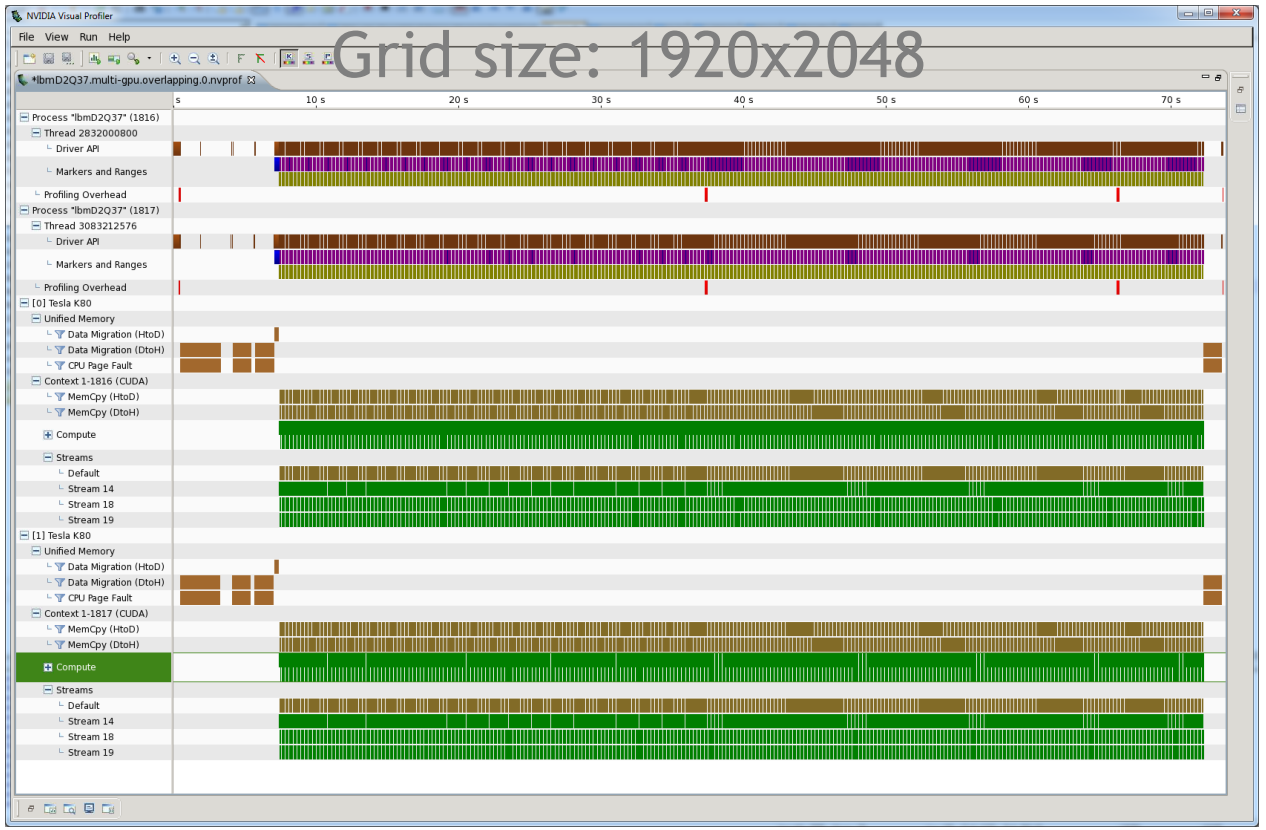
```
int devicenum = atoi(getenv("MPI_LOCALRANKID"));
```

LBM D2Q37 – MULTI GPU

Strong Scaling

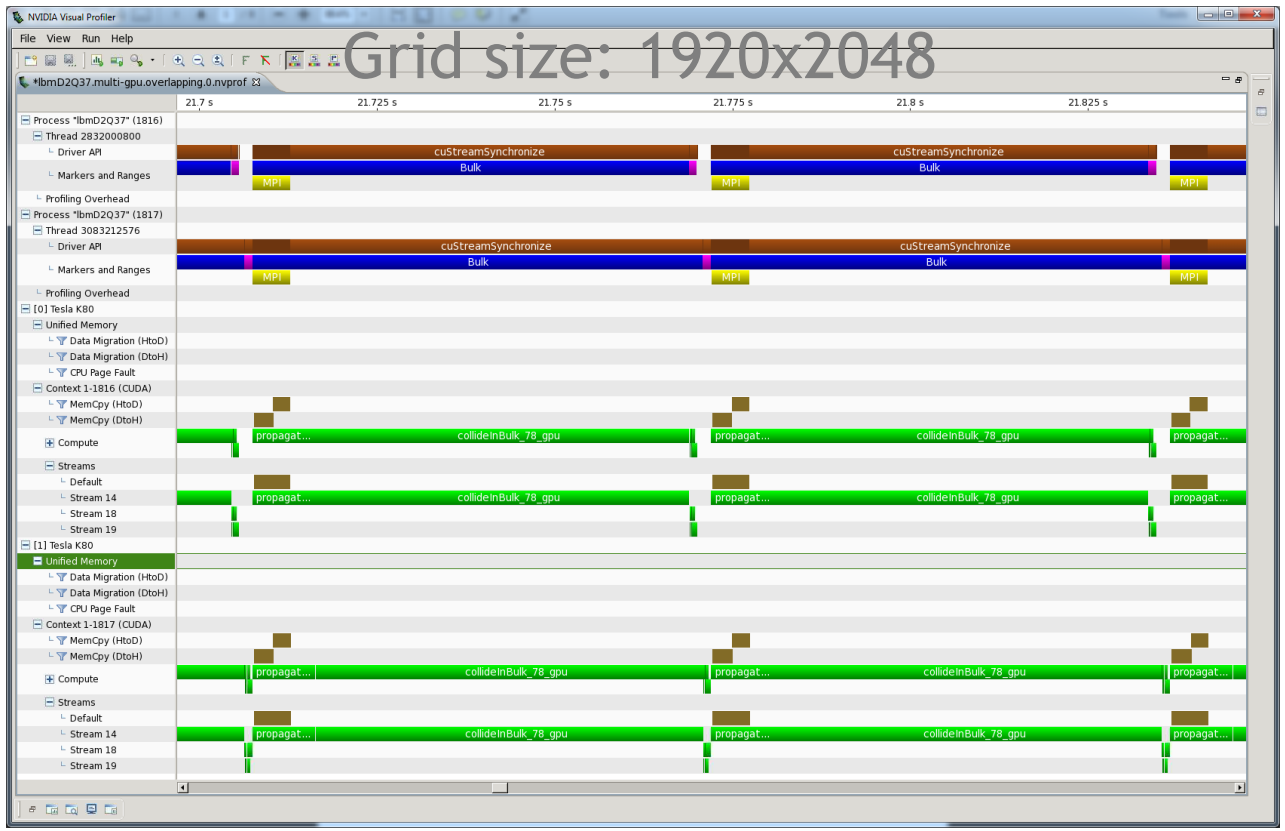
LBM D2Q37 - MULTI GPU

Overlapping Communication and Computation



LBM D2Q37 – MULTI GPU

Overlapping Communication and Computation



OPENACC QUICKSTART ON JURECA

Live Demo

```
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

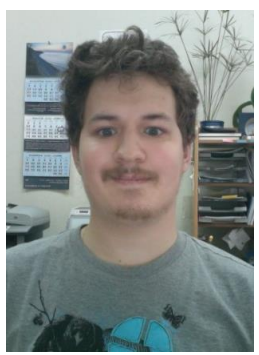
Investigate performance and scaling

JÜLICH
APPLICATION LAB
NVIDIA

Dirk Pleiter (JSC)

Andreas Herten (JSC)

Jiri Kraus (NVIDIA)



Andrew V. Adinetz (JSC)*

* untill April 2015

OPENACC TRAINING COURSE

Introduction to GPU programming using OpenACC

Begin: 24.Oct.2016 09:00

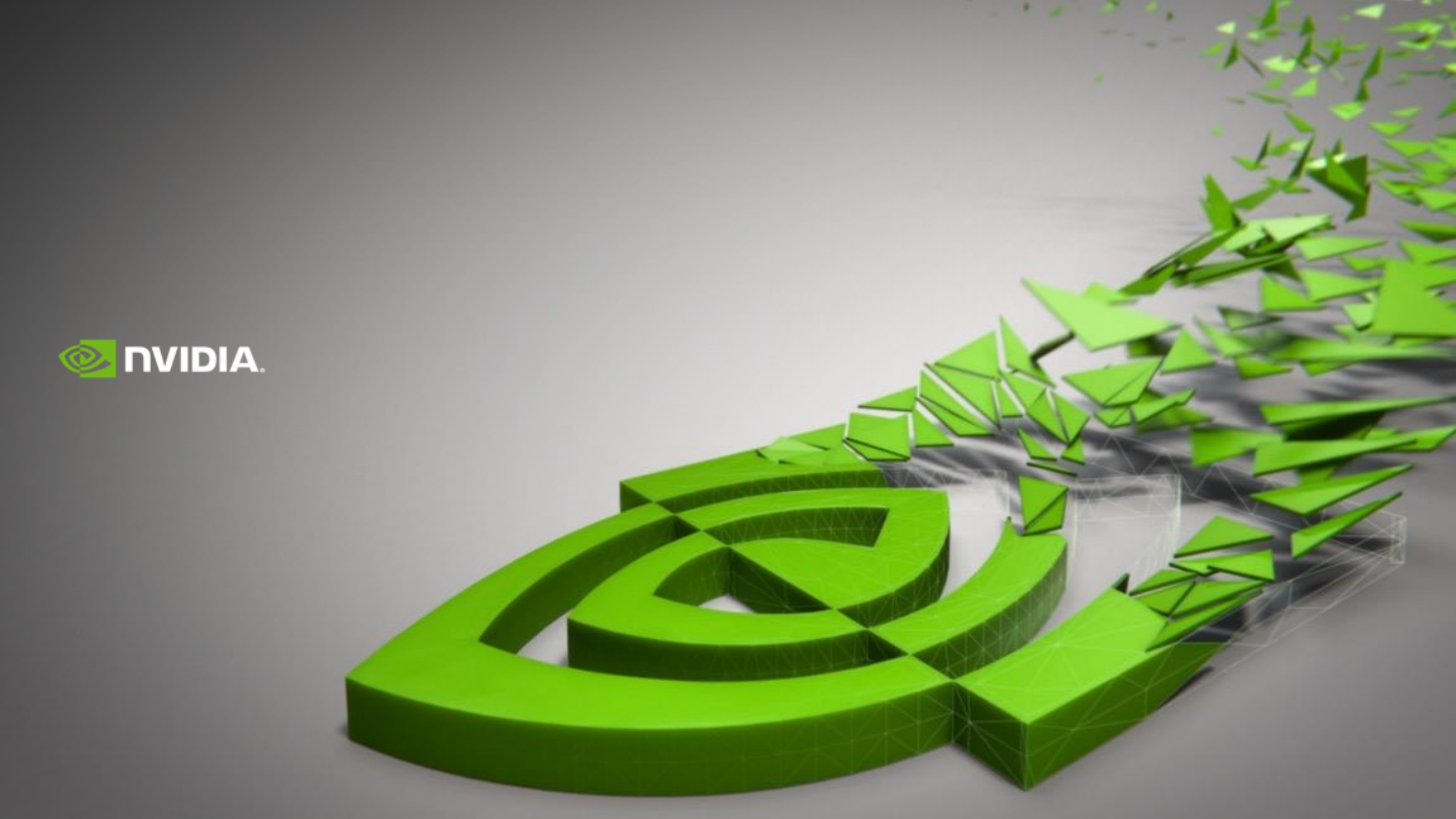
End: 25.Oct.2016 16:30

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

http://www.fz-juelich.de/SharedDocs/Termine/IAS/JSC/EN/courses/2016/gpu-openacc-2016.html


NVIDIA.