Technische Universität München
Lehrstuhl Informatik V
M. Bader, O.Meister

WS 2013/14
Worksheet 3
November, 27th 2013

Tutorial: HPC - Algorithms and Applications WS 13/14

Complete the following assignments (alone or in a group), and send your source code via e-mail to `meistero@in.tum.de` until Sunday, December, 8th 2013.

Worksheet 3: Coalesced Access, Sparse Linear Algebra

Assignment 1: Recap on Coalesced Access

Let `n = 64; tx = threadIdx.x; ty = threadIdx.y; tz = threadIdx.z;` Are accesses to the array `float *A` in the following calls uncoalesced, partially coalesced or coalesced (chipset: NVidia Fermi, CUDA cc $\geq$ 2.0)? Answer shortly.

a) `float f = A[tx];`

b) `float f = A[tx + 1];`

c) `float f = A[2 * tx];`

d) `float f = A[tx / 2];`

e) `float f = A[n * tx];`

f) `float f = A[ty];`

g) `float f = A[(tx * n + ty) * n + tz];`

h) `float f = A[(tz * n + ty) * n + tx];`

Assignment 2: CSR kernel

Write a CSR matrix-vector multiplication kernel for the PageRank example code.

a) Open `kernels.cu`, define grid and block size

b) Implement the `k_csr_mat_vec_mm` kernel:

   i) Assign one thread to a matrix row

ii) Compute one row × vector product in a loop.

iii) Add the result to the output vector.

c) Compile the code using `make`. You will have to choose a suitable C compiler (gcc is the default in `Makefile`).

d) Try the kernel on a small matrix (*mtx/my.mtx*). The program output should be:

```
x_1 = 3.602992e-01
x_2 = 2.700142e-02
x_3 = 6.238353e-02
x_4 = 2.431707e-02
x_5 = 4.042290e-01
x_6 = 2.700142e-02
x_7 = 2.431707e-02
x_8 = 5.378454e-02
x_9 = 1.666666e-02
```

e) Next, test the kernel on a bigger matrix. Download for example *flickr.mtx* from `http://www.cise.ufl.edu/research/sparse/MM/Gleich` and run the PageRank algorith on the matrix (if it's too big choose a different matrix). Which page is the most relevant according to the algorithm?

Assignment 3: Vectorized CSR kernel

Write a vectorized CSR matrix-vector multiplication kernel for the PageRank example code.

a) Implement the `k_csr2_mat_vec_mm` kernel in `kernels.cu`:

i) Assign one warp to a matrix row

ii) Allocate a shared array `vals[]` for the partial results of a block

iii) Compute one row × vector product in a loop. This time, parallelize the loop over all 32 threads in the warp. Take care that access to the arrays `indices` and `data` is coalesced.

iv) Use a reduction of some kind (ideally: binary fan-in) to add up the partial sums in `vals[]` and add the output to the result vector.

b) Try the kernel on *mtx/my.mtx* again and check if the output is consistent with Assignment 2d.

c) Test both CSR kernels on the big matrix and measure execution times (`time ./sparse mtx/flicker.mtx`). How does performance compare?