PROGRAMMING TIPS

1) for absolute beginners: start programming with Octave M-scripts. Useful links for smartphone/PineA64+/OrangePi C++ users:

-original C++ official site, and online compiler

http://www.cplusplus.com/

http://www.cpp.sh/

-tutorials with online compiler on multicore platform session:

https://www.tutorialspoint.com/cplusplus/index.htm

http://www.tutorialspoint.com/compile_cpp11_online.php

2) for data processing beginners (GB of scalars and vectors processing experience) in sequential C++, parallel CUDA C:

-use **vector** container (read&process chunks of data); for psuedorandom accesses use **map** container (unordered_map – faster, map – self sorting during creation via iterator); quite intuitive and good results are for vector< vector< float > > - vecD (vector in D-dimensions),

-**FRAGMENT** and organise data:

-only small fragments of data with low programming instructions are fast

-commonly only small fragments are trivially attainable

-fragmented data are simpler to compute in parallel:

-**NORMALIZE** data:

-one can use fastest datatypes which are FLOAT (INT are commonly for counters – use auto, or iterators), and store only a median value

-FLOAT considerations:

-data storage precision: +-1000 variable value (better are +-100)

-accumulative computation accuracy max iteration = 100

-variable comparisions (err: 1.1==1.1!!!):

global float floatErr = 10e-5;

(1.1 – floatErr < 1.1)&&(1.1 + floatErr > 1.1)

pro: use CPU register described in 06_registerGlobal

PARALLELIZE computations:

-quite simple sequential parallelism on multicore CPU: GNU parallel scripts,

-quite simple parallelism on GPU: accelerated libraries like Thrust,

-computations of vectors with size bigger than 8000 elements are faster only on GPU

-**UNCOMPRESS** to RAM:

-tip from prof. Starzynski from Warsaw University: „faster for bigger data”

-applicable from few MB’s data size, can reduce used PCIe bandwidth in CPU-GPU transfers,