PROGRAMMING TIPS

1) for absolute beginners: start programming with Octave M-scripts (high-level, ultra-easy-to-learn, scientific language) or Python,

2) 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

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

-use **vector** container (read&process chunks of data – CPU memory bandwidth usage); for psuedorandom accesses use **map** container (unordered_set – faster, map – self sorting during creation via iterator); quite intuitive and good results are for vector< vector< float > > - vecD (vector in D-dimensions, first dimension is allocated in continous virtual memory address – physicall memory continuity is platform dependend),

-acceptable results will be for bidirectional list<vector<float>> if different size vectors are often added/deleted,

-**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,

-**SORT** = organise data:

-easier and faster to processing

-quite fast to obtain statistics (min, max, median, etc.)

-**NORMALIZE** data:

-one way to normalize is get median value from mean of sorted data and store it separately and as second step subtract it from data. If it does fit in float storage acceptable range get min element as 0 and divide each element by max element normalize to 1, (note that norm[-1, 1] step is lossy if max element is much greater than median - use mean instead). Consider making variables as struct similar to (during parallel calculations it should make better job than double calculations – just check variable range in code execution, it is typical effort in Digital Signal Processor fixed point computations):

```
struct floatMedian
{
        float professionalComputations [ const unsigned bigEnoughForParallel];
        long medianExtracted += calculatedMedian; //initalize with 0
};
```

-note median drawbacks – there are some other tricks like trimmed mean

-one can use fastest datatypes which are commonly FLOAT (uint16_t are basicly for counters – use auto, or iterators), and store only an additional median value to on-air normalized data

-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;

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

pro: use CPU register described in 06_registerGlobal

-note that float pointer size is doubled than float datatype size on 64bit architecture:

sizeof(float*) = 8
sizeof(float) = 4
sizeof(uint16_t) = 2

-instead of float, the size of variable double representation size is doubled, but computation capabilities of double variables are not halfed (smaller),
-keep vectors normalized and store its median value as separate variable
-greatest range for floats is [-1.0f, +1.0f] (23bit - minimal storage error)

-**FRAGMENT** data:

-only not too small fragments of data with low no. of programming instructions are fast
-commonly only small fragments are trivially attainable
-fragmented data are simpler to compute in parallel:

PARALLELIZE computations:

-quite simple sequential parallelism on multicore CPU: GNU parallel scripts,
-quite simple parallelism on GPU: accelerated libraries like Thrust (thrust::device_vector - omit expensive transfers for small data size),
-computations of vectors with size bigger than 8000 elements are faster only on GPU (for AMD CPU fans – please check single thread CPU and GPU efficiency for big enough data – you will be surprised with results)
-keep computations only on GPU, use CPU for communication,

-**Time To Market cosiderations** – cost of computing Octave easto-to-write script few times during night or writing an OpenCL optimized application for week and obtain results in seconds (quite rare situation of often used programs)

-**Look up Tables –** usage of static vector containing prec-computed values highly reduces overall amount of computations. Check your program for multiple calculations of **2+2=4!**

4) C++ offline documentation could be accessible via cppman package:

```
#sudo apt-get install cppman
#cppman -c                  #offline download / update
#cppman -m true             #set cppman instead of #man in #vim
```

a) #cppman vector
b) #cppman std::vector
c) #man std::vector

5) shell alias is convenient way to make an autofilling in terminal, simply edit bash configuration file (for example „ls -als” shortcut with „ll”):

```
#sudo vim ~/.bashrc
#alias ll="ls -als"
```