10.01.2018

GNU Octave-Cli + GNU Parallel

1) Mainly C++-libraries based high level, script-programming language GNU Octave is freely provided with many low computational complexity algorithms. Overall Octave's scripts execution time efficiency is not comparable to optimized C++ code, but with usage of already implemented proper processing functions makes it trivially applicable for prototyping and tuning tests of algorithms. On the other hand, median many core CPU's provides additional capabilities of computations on modern Personal Computers local networks. GNU Parallel makes parallel computations of real data (at easiest example: on different data files) easy deployable. If one will use GNU Parallel with publishing purposes, please cite authors article. On next tip dashes there will be provided simplified usage of both packages on local network (it is applicable to Virtual Box machines for educational purposes as well).

2) install GNU Octave:

#sudo aptitude install parallel octave

3) write Octave test kernel function:

#cd ~/Documents

#mkdir octave_Parallel && cd octave_Parallel

#vi kernelFunction.m

```
function [ result, functionReturnVal ] = kernelFunction( argIn1, argIn2 )
functionReturnVal = -1; argIn1 = single( argIn1 ); argIn2 = single( argIn2 );
if ( argIn1 > 1 )
	result = argIn1 + argIn2;
else
	result = NaN;
	functionReturnVal = -2;
	disp( [ 'ERROR: wrong value of input argument (', num2str( argIn1 ) ,')!' ] );
	break;
endif;
functionReturnVal = 0;
endfunction;
```

4) check if kernel function is correctly adding numbers bigger that 1.0f

#octave-cli -p ~/Documents/octave_Parallel/ --eval "[res, retVal] = kernelFunction(1.0, 2.0)"

#octave-cli -p ~/Documents/octave_Parallel/ --eval "[res, retVal] = kernelFunction(2.0, 2.0)"

5) write Octave - Command Line Interface commands for deploying work to GNU Parallel (it will be executed on separate, each CPU core provided in GNU Parallel configuration file clusterList.txt):

#vi parallelCommands.txt

```
octave-cli -p ~/Documents/octave_Parallel/ --eval "kernelFunction( 2.0, 2.0 )"
octave-cli -p ~/Documents/octave_Parallel/ --eval "kernelFunction( 1.0, 2.0 )"
octave-cli -p ~/Documents/octave_Parallel/ --eval "kernelFunction( 3.0, 2.0 )"
octave-cli -p ~/Documents/octave_Parallel/ --eval "kernelFunction( 4.0, 2.0 )"
octave-cli -p ~/Documents/octave_Parallel/ --eval "kernelFunction( 5.0, 2.0 )"
octave-cli -p ~/Documents/octave_Parallel/ --eval "kernelFunction( 6.0, 2.0 )"
octave-cli -p ~/Documents/octave_Parallel/ --eval "kernelFunction( 7.0, 2.0 )"
```

such file can be easily generated in Octave or Bash script.

6) on each computer in local cluster create the same userName account,

7) write numbers of cores (one can use #nproc function within bash script to get CPU number of cores) to clusterList.txt. This file includes configs for GNU Parallel to make usage of local and remote computers cores. Lets suppose that we have two computers with 2 cores on local PC, and 4 cores on one remote PC. We will write GNU Parallel config file to use all local cores and 4 cores on remote PC with IPv4 address.

#vi clusterList.txt

```
1/:
4:/IPv4
```

8) generate local sh public-private key pairs (credentials) - one can use defaults:

#ssh-keygen

9) upload credentials to each worker (in our case, there is single remote computer) for automatic, authorized and encrypted work distribution:

#ssh-copy-id IPv4

10) check if there is no need for additional authorization via ssh (after this command user should be logged in automatically, without ask for password), create script instruction folders on remote computers (workers), and logout:

#ssh IPv4

#mkdir ~/Documents/octave_Parallel; exit

11) rsync local and remote folders (copy only file differences between folders). For details please refer to RSYNC tutorial.

#rsync -uv --progress -e ssh ~/Documents/octave_Parallel/* userName@IPv4:~/Documents/octave_Parallel/

12) run GNU Parallel for deploying parallel work on 6 cores in local cluster (2 local cores and 4 remote cores):

#touch log.txt && rm log.txt && parallel --slf clusterList.txt --progress < parallelCommands.txt &>> log.txt; cat log.txt

13) please note that data, instructions network deployment, and run of octave-cli provide overheads. As a result processing should be big enough for obtaining any benefits from such model. For small data and instructions communication please use Message Passing Interface, which is better for such communication,

14) few Octave tricks:

-try to preallocate pseudostatic variables used in Octave with zero function, for avoiding each loop pass dynamic variable copying,

-Octave does not fully manage dynamic memory deallocation, so use system thread trick inside Octave script for function Resource Acquisition in Initialization RAII. Described below function frees kernelFunction global variables resulting in memory freeing:

...some Octave instructions...

[threadStatus, functionReturnValue] = system(‘octave-cli -p ~/Documents/octave_Parallel/ --eval "kernelFunction(2.0, 2.0, \”someFunctionStringArgument\”)"’)

...some Octave instructions with usage of functionReturnValue script global variable, after checking threadStatus value...

-use single datatype (32 bit floating point) instead default double (64 bit floating point) variable, for getting about 1-2 orders of magnitude faster program execution.

15) some High Performance Computing tricks:

-create local high throughput RAMDISK for data deployment to workers (best efficiency will be provided with usage of PCIe multiple 1GE / 10GE network Host Bus Adapters HBA’s, and getting benefits from fastest possible star network topology) . For details please refer to RAMDISK tutorial. Single session mount of volatile 4GB RAMDISK:

#mkdir ~/RAMDISK && sudo mount -t tmpfs -o size=4096M,mode=777 tmpfs ~/RAMDISK/

check if RAMDISK is mounted, and running:

#mount;

#echo;echo RAM sequential writes:;for ((i=0;i<3;i++)); do dd if=/dev/zero of=~/RAMDISK/a.txt conv=fdatasync bs=1512M count=1;done; rm ~/RAMDISK/a.txt;echo;echo HDD sequential writes;for ((i=0;i<3;i++)); do dd if=/dev/zero of=~/a.txt conv=fdatasync bs=1512M count=1;done; rm ~/a.txt;

or within graphical environment with Network Shares icon.

-tmpfs RAMDISK has sequential read and write efficiency of about 6-10 SSD’s RAID0, but pseudorandom read (of small data) latency is about order of magnitude smaller. Please note that overall storage size is about two orders of magnitude smaller than Solid State Drive disk array,

-make Samba file sharing of RAMDISK on local computer with HBA’s. Mount file share with project folders on workers – there will be no need for rsync instructions, and data will be reasonably deployed to workers. For details please refer to Samba tutorial.

#sudo apt-get install samba cifs-utils; sudo smbpasswd -a userName

#sudo vim /etc/samba/smb.conf

```
[parallelProjectFileShare]
path = ~/RAMDISK/
force user = userName
force group = root
create mask = 0777
directory mask = 0777
hosts allow = IPv4 IPv4_1
read only = no
public = yes
guest ok = yes
writable = yes
```

#sudo service smbd restart

workers file share mount:

#ssh IPv4

#mkdir ~/remoteRAMDISK && sudo mount -t cifs //IPv4/parallelProjectFileShare /home/userName/remoteRAMDISK -o username=userName

-consider buying past generations of high-end servers as long as they provide big amounts of RAM and CPU cores in very low prices (for example: HP dl160g6; Dell R610; Dell R805, et cetera...). Most of servers are as loud as hairdryer. Using bigger amount of traditional servers, used for calculations / data processing provides to trivial conclusion about electricity bill and GPGPU technology utilization (on base of cheapest possible CPU with full support of PCIe3.0x16),

-consider switching to usage of LINUX on the basis of LINUX Puppy (author do recommend it for advanced users for working on synchronized liveCD sessions – please refer to LIVECD basic tutorial as introduction), resulting in deletion of majority of operating system hardware bottlenecks (mainly motherboard buses to drives, and hard-drives themselves),

Post Scriptum: it is simplified tutorial providing very basis of algorithms prototyping with GNU Octave-Cli and GNU Parallel packages in many core parallelism scheme. End program must be developed efficiently after completion of vast set of algorithm tests. Please do note, that programming language is less important (for example Bash scripts efficiency), in opposition to popular stereotypes. Program efficiency is mainly correlated with computations complexity, and efficiency of hardware use (General Purpose Graphical Processing Units provides best results for majority of current problems in 2018y. - efficient execution time, but with price of longer development time). Please do check your hardware candidate with Unit Tests before you will make bigger purchase order. Some pre-purchase research could bring you revealing ideas, like for example: the same efficiency of ordinary laptop with GPU and some high-end traditional server at higher price.
Post Post Scriptum: such programming model is applicable to many other techniques. Author personally used it for trivial and efficient many core (many CPU’s with more than one core each) work deployment. The kernel program was a C++11 source file, shared via rsync, compiled optimally on workers (...-mtune=native -march=native -O2…; #make -j`nproc`;), working on Samba file sharing data folder, via 10GE network star topology (some tip here: if one want to aggregate two 10GE’s one should not use two 10GE interfaces at single HBA and use two 10GE single interface within single HBA’s instead. Please refer to LINK_AGGREGATION tutorial).