MPI implementation over mobile handsets

These days mobile phones are quite powerful and can perform a lot of computation. So, the idea is to use the idle computation capabilities of mobile phones to solve some compute-intensive problem such as the inversion of a matrix of large dimensions using parallel computation.
MPI or message passing interface (open source library
used in high-performance computing for message passing between parallel computing nodes) can be used to create multiple processes under the control of a parent process.
MPI processes can coordinate together and concurrently execute in different devices.
Need to build an MPI-based mobile cluster to run distributed applications on mobile phones.
References -
https://thesai.org/Downloads/Volume7No9/Paper_13-High_Performance_Computing_Over_Parallel_Mobile_Systems.pdf

https://www.scientificbulletin.upb.ro/rev_docs_arhiva/fullffc_583765.pdf

Deliverables

- Performance and energy-efficiency gains. By splitting the processing among several devices. This will not only speed up the processing, but will also distribute the battery drain across all devices.

- Situations where access to High Performance Computing machines is nonexistent and the only means for accomplishing resource-intensive computations are mobile devices. Those situations turned out to be widely spread in many military as well as civilian applications.

Methodology to be adopted for the implementation of the project

A) Writing the program for the problem specified (here - computing inverse of matrix of large dimension) .

B) Parallelizing it using MPI.

C) Compiling MPICH2 for ARM™ architecture.

D) Building a cluster of Android devices.

E) Running the distributed application.

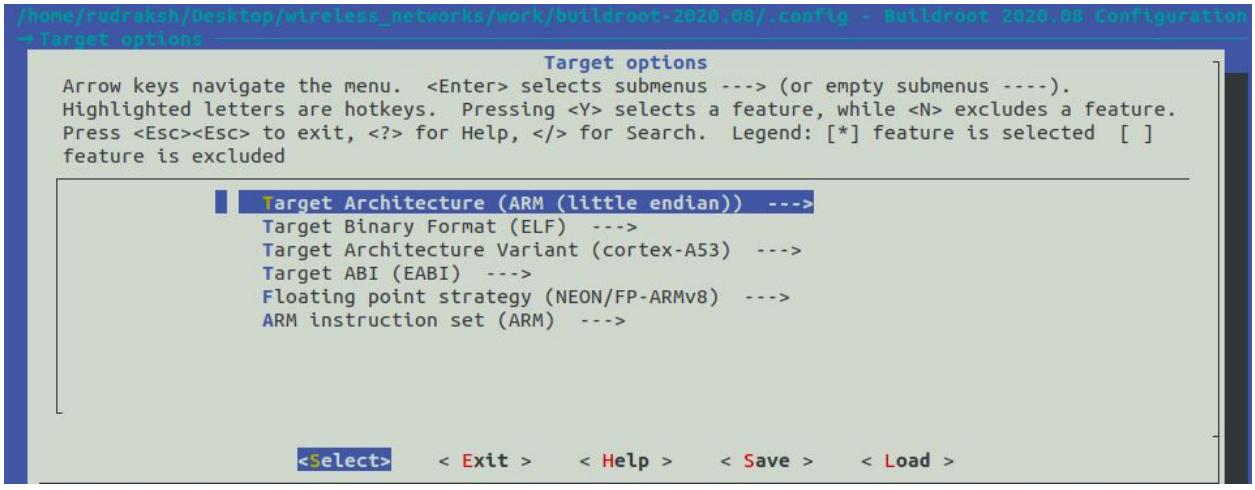
Compiling MPICH2 for ARM architecture

Current Android OS current phones run on ARM™ CPUs but are not shipped with a preinstalled native compiler. Tools and distributed application executable code for ARM™ need to be generated on a platform where a C compiler is available. Using a compiler to generate executable code for a different platform different than the one where compiling takes place is a process named cross-compiling.

In order to build an MPI-based mobile cluster, we have to statically link and cross compile MPICH2 with the Uclibc library(a library made especially for linux-embedded devices).

For cross compiling to ARM executable code on Intel X86 platforms we need to generate a Uclibc based GNU toolchain(the compiler tool chain contains three basic elements, first compiler itself, second Linker and third Loader) for the ARM architecture, the processor's architecture of the devices used in the current project.

1) To generate the toolchain. After downloading Buildroot software, go to the Buildroot directory and use *make menuconfig* command to open Buildroot configuration tool. After choosing the suitable configuration options, start the building process using *make* command (this could take upto an hour to finish so be patient!) . The Output will be found in '/PathToBuildroot/output/host/usr/bin' directory.

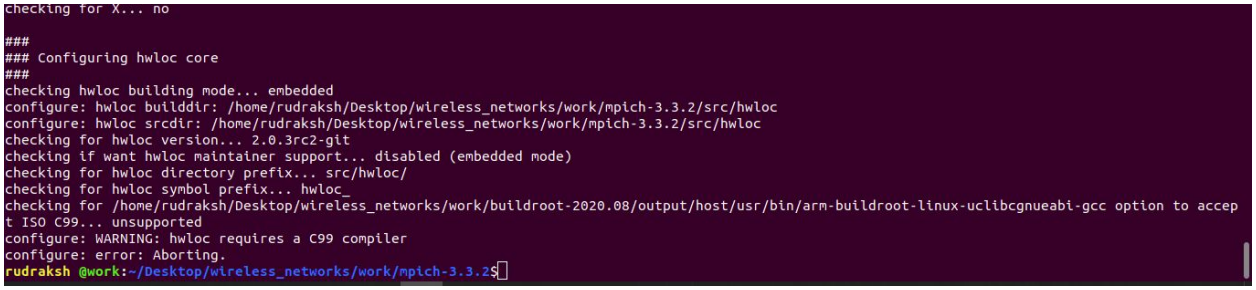


2) Compile MPICH for the ARM architecture. After downloading MPICH, create a new folder with the name “build”. Then use the following command to statically build MPICH (from the MPICH directory):
    - *CFLAGS="-I/PathToBuildroot/output/host/usr/include/ -static"*
    - *LDFLAGS=" - L/PathToBuildroot/output/host/usr/lib/ -static"*
    - *CC=/PathToBuildroot/output/host/usr/bin/arm-buildroot-linux-uclibcgnueabi-gcc*
    - *./configure –prefix=/mpich/build/ --enable-static --disable-shared --host=arm-linux --with-pm=smpd --with-device=ch3:sock --disable-f77 --disable-fc*
    - *make*
    - *make install*

The output should be static executable for ARM:
file ./bin/mpiexec should return

```
bin/mpiexec: ELF 32-bit LSB executable, ARM, version 1 (SYSV),
statically linked, not stripped
```

NOTE : Don’t download the latest version of MPICH because it will need C99 compiler to compile and the generated toolchain dont support C99 compiler, so instead download the older version mpich2-x.x.x which uses C89 compiler instead of C99.



3) Use */mpich/build/bin/mpicc* command to compile your C code. e.g.:
/mpich/build/bin/mpicc -o myCode mycode.c
Then copy the generated “myCode” file to each mobile device */data* directory.

(ref - https://hex.ro/wp/blog/compiling-mpich2-for-android-and-running-on-two-phones/)

Copying the files to Android device

Copy the files in the generated /mpich/build/bin directory to each device in /system/xbin directory and copy the generated “myCode” file to each device /data directory.

Requirements -

- **Android** Debug Bridge installed (**adb** is a versatile command-line tool that lets you communicate with a device).

- **Unlocked bootloader** -

  Open your phone's app drawer, tap the Settings icon, and select "About Phone". Scroll all the way down and tap the "Build Number" item seven times. You should get a message saying you are now a developer.

  Head back to the main Settings page, and you should see a new option near the bottom called "Developer Options". Open that, and enable "OEM Unlocking", if the option exists (if it doesn't, no worries–it's only necessary on some phones).

  Next, enable "USB Debugging".

- **Flashing software** depending on your phone model. Most common ones are fastboot (official flashing tool from google), Odin (exclusively for samsung devices).

There are 2 ways one can copy files from pc to /system folder on mobile devices

1. Using ADB push command -

   Connect your phone to the computer, open the terminal and run *adb push -p /mpich/build/bin /system/xbin.* You need to run adb as root in order to execute this command. Now if you try to run adb root you will see the error message `adbd cannot run as root in production builds.`

   read -
   https://stackoverflow.com/questions/25477424/adb-shell-su-works-but-adb-root-does-not

   https://android.stackexchange.com/questions/5884/is-there-a-way-for-me-to-run-adb-shell-as-root-without-typing-in-su

   After reading above references you should be able to understand that we need to flash a rom which has ro.debuggable flag set to 1 and ro.secure set to 0. The easiest way to do

that is to download stock rom for you device from the internet then extract it, you will see a boot.img file, unpack that file and modify the corresponding flags, repack it again and flash it to your device. Detailed process is given at the below link https://android.stackexchange.com/questions/69954/how-to-unpack-and-edit-boot-img-for-rom-porting

If the above steps went successfully you should now be able to use the adb push command. It didn't work? don't worry we have an easier plan B for you.

2. Root your device then use any "root file explorer" application and copy paste the files in the desired directory.

Rooting the Android device

There are tons of video tutorials out there on YouTube and there are many website articles available on how to root your phone but unfortunately most of them are fake and useless.
Some will say that you can root your phone by installing just an rooting app, actually there was a time when these apps were freely available on Google Playstore and can possibly root your phone(older ones) but later google remove all such apps from playstore, and i don't think that new smartphones can be rooted by just an app.
The most common method which i'll recommend for rooting your device is to first flash a custom recovery on your phone, then boot into recovery mode and flash SuperSu or Magisk Manager to your phone.

Here is an article https://odindownload.com/root-android/

Process of rooting is device specific, so please check if your device enforces dm-verity.

In my case i've **Samsung Galaxy J7prime(SM-G610F)**, so I'll discuss steps for rooting particular to my device.
Reference - [RECOVERY][ROOT] TWRP 3.1.1-1 Galaxy J7 Prim… | Samsung Galaxy J7

WHATEVER YOU DO BEFORE FLASHING ANYTHING MAKE A BACKUP OF YOUR DEVICE AND HAVE THE STOCK FIRMWARE TO HAND JUST IN CASE DOWNLOADED FROM SAMMOBILE OR UPDATO.

https://www.sammobile.com/firmwares/ is a trusted site where you can find stock firmware of your samsung device.

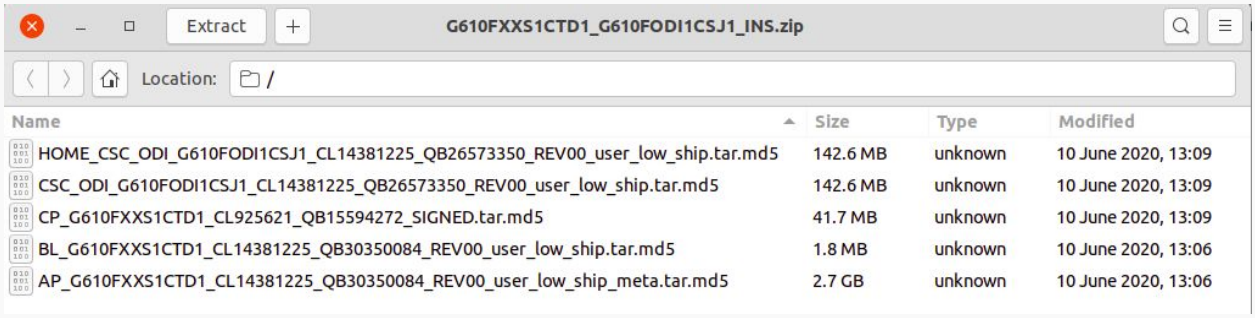
1. Download custom recovery. I recommend using TWRP(device specific, unfortunately not available for SM-G610F) but luckily i managed to find an unofficial version for my device here.

2. Download a flashing software(odin for samsung devices) and install usb drivers from here.

3. Check if your device enforces dm-verity and force encryption. If it does then you need to flash a patch file after flashing SuperSU to disable it. You can download it from here which is compatible with most android devices.
   IMPORTANT! My device enforces dm-verity. ANY modifications or even mounting system will put the device into a bootloop. To prevent this TWRP will ask at first boot if you want to keep system 'Read only' or 'Allow modifications to system'. If you choose to keep 'Read only' you will have to flash TWRP at every boot to recovery.
   If you choose to 'Allow' then SuperSU or the boot patch needs flashing below to disable dm-verity.
   Detailed info on dm-verity - https://source.android.com/security/verifiedboot/dm-verity

4. Flash TWRP, boot into recovery to flash SuperSU (you can do it with adb sideload for easy) then flash the patch file you downloaded. That's it you're now rooted.

5. If you mess up anything or are stuck in bootloop then flash the stock firmware you've downloaded before.
   You only need to flash the AP_G10Fxx file, see this for more info.



Building a cluster of Android devices

After copying the files to the devices change /system/etc/hosts file on each device to contain all IP addresses. Any file explorer application that requires root permission can be used. The hosts file should look like
this:
10.0.0.1 localhost
10.1.1.1 hostname1
10.2.2.2 hostname2

NOTE: The default hostname ”localhost” was changed on all devices and each device was given a distinct hostname.
MPICH should be running on the device to test it you can run the following command from the phone terminal:
`mpiexec -n 1 /data/myCode`
In case that “permission denied” message appears:
Run `chmod 777 /system/xbin/mpiexec` on the phone's terminal to give the program executable permission.

Setup ssh for communication between devices. First you have to download ssh server for your Android phones. In my case I used Android ports “Unix command line packages built for Android”. Use the mobile device's shell to run this command:
`opkg install dropbear openssh`. Then generate public and private key pairs in //.ssh/id_rsa (in system(root)) using this command `/data/local/bin/ssh-keygen -t rsa`. After copying the public key to the other phone //.ssh/authorized_keys (if the file does not exist, create it). Start dropbear on all devices by running: `/data/local/bin/dropbear`. Then Check that ssh is working by trying this command `/data/local/bin/sshroot@ipaddress`.

NOTE: The hotspot was enabled on one of the devices and the other devices connected to it without being connected to the Internet just to create a local network.