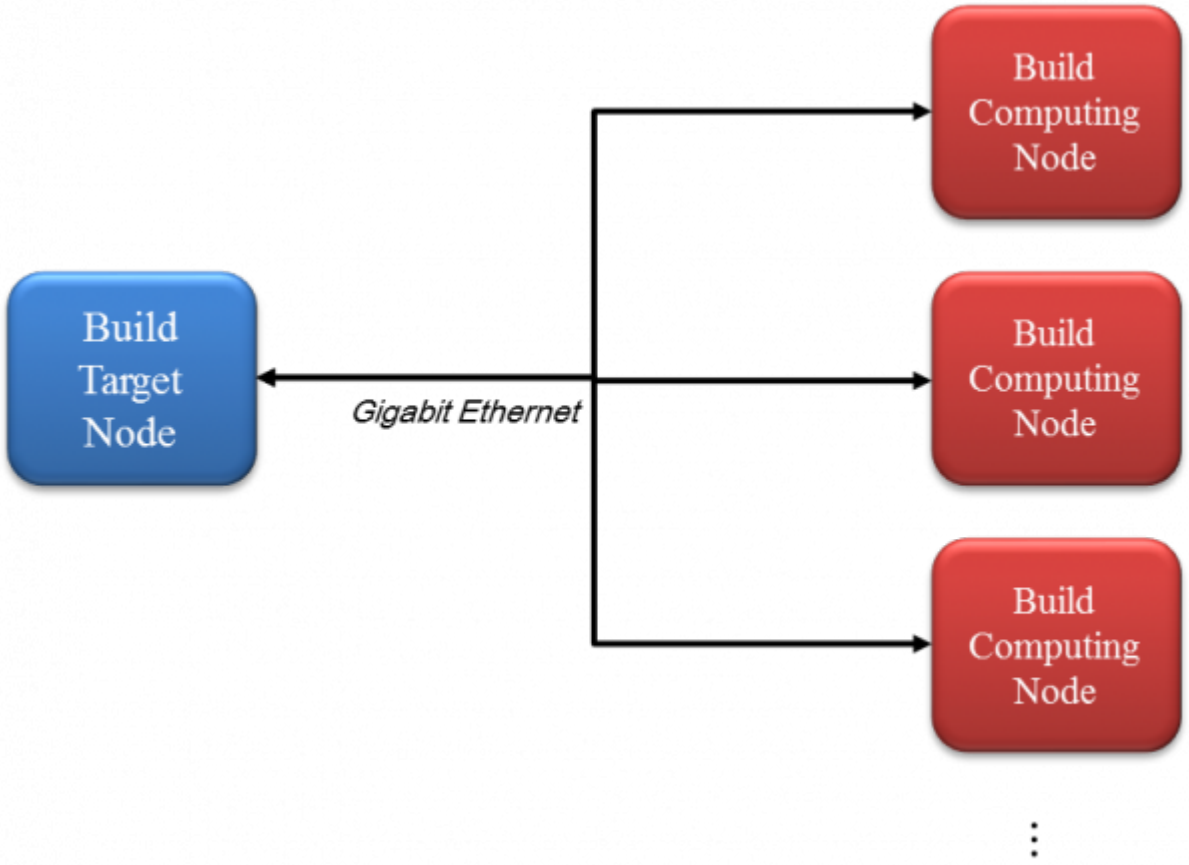
Creating build farm using ODROID-MC1/HC1

Compile farm page of Wikipedia said:

> **A compile farm** is a server farm, a collection of one or more servers, which has been set up to compile computer programs remotely for various reasons. Uses of a compile farm include:
>
> * Cross-platform development: When writing software that runs on multiple processor architectures and operating systems, it can be infeasible for each developer to have their own machine for each architecture — for example, one platform might have an expensive or obscure type of CPU. In this scenario, a compile farm is useful as a tool for developers to build and test their software on a shared server running the target OS and CPU. Compile farms may be preferable to cross-compilation as cross compilers are often complicated to configure, and in some cases compilation is only possible on the target, making cross-compilation impossible.
>
> * Cross-platform continuous integration testing: under this scenario, each server has a different processor architecture or runs a different operating system; scripts automatically build the latest version of a source tree from a version control repository. One of the difficulties of cross-platform development is that a programmer may unintentionally introduce an error that causes the software to stop functioning on a different CPU/OS platform from the one they are using. By using a cross-platform compile farm, such errors can be identified and fixed.
>
> * Distributed compilation: Building software packages typically requires operations that can be run in parallel (for example, compiling individual source code files). By using a compile farm, these operations can be run in parallel on separate machines. An example of a program which can be used to do this is distcc.

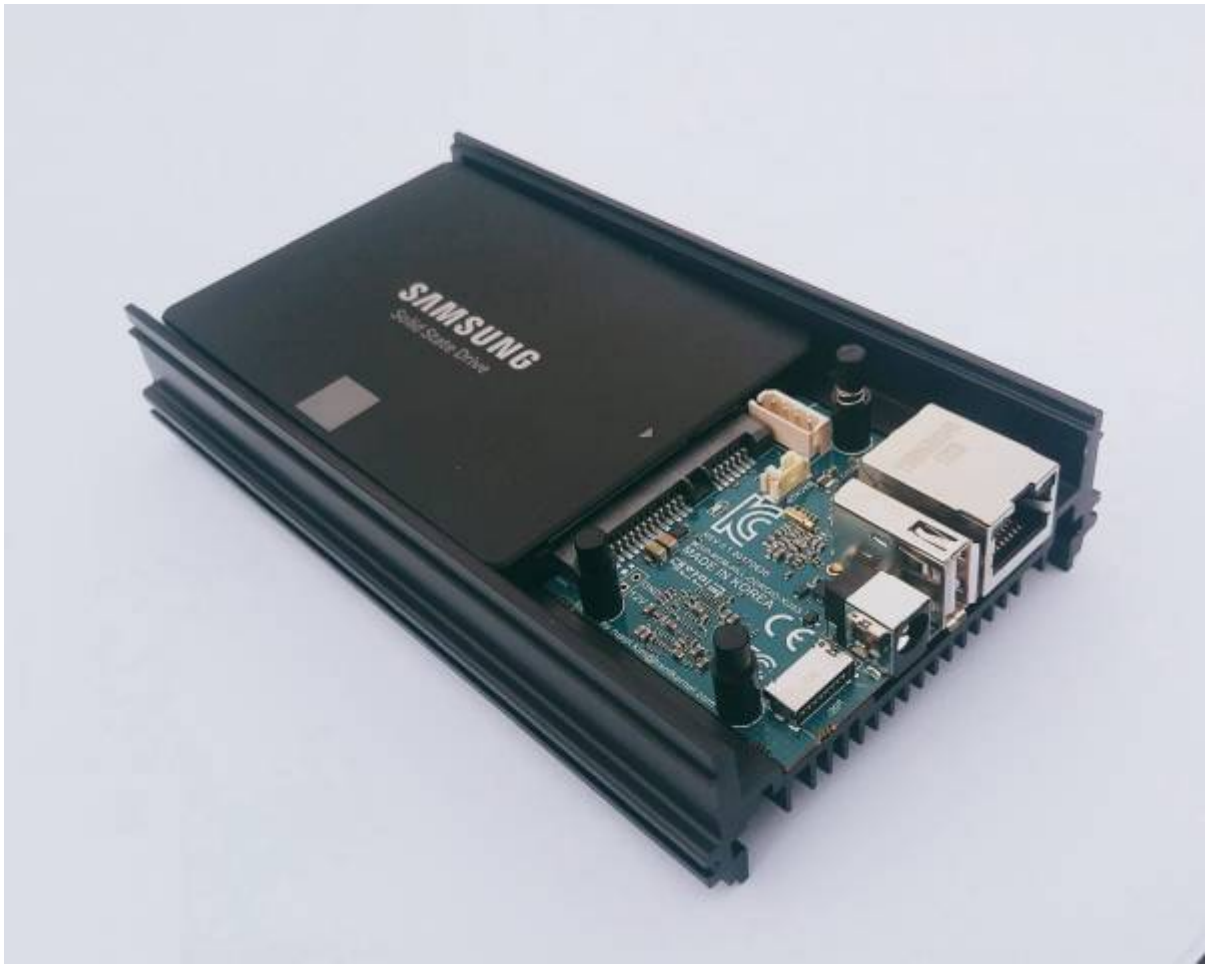
This page will give you the guide about how to create build farm using ODROID-MC1(or several ODROID-XU4), ODROID-HC1 and distcc. This guide is based on Hardkernel's Ubuntu 16.04.2 or later.

Build Farm System Diagram

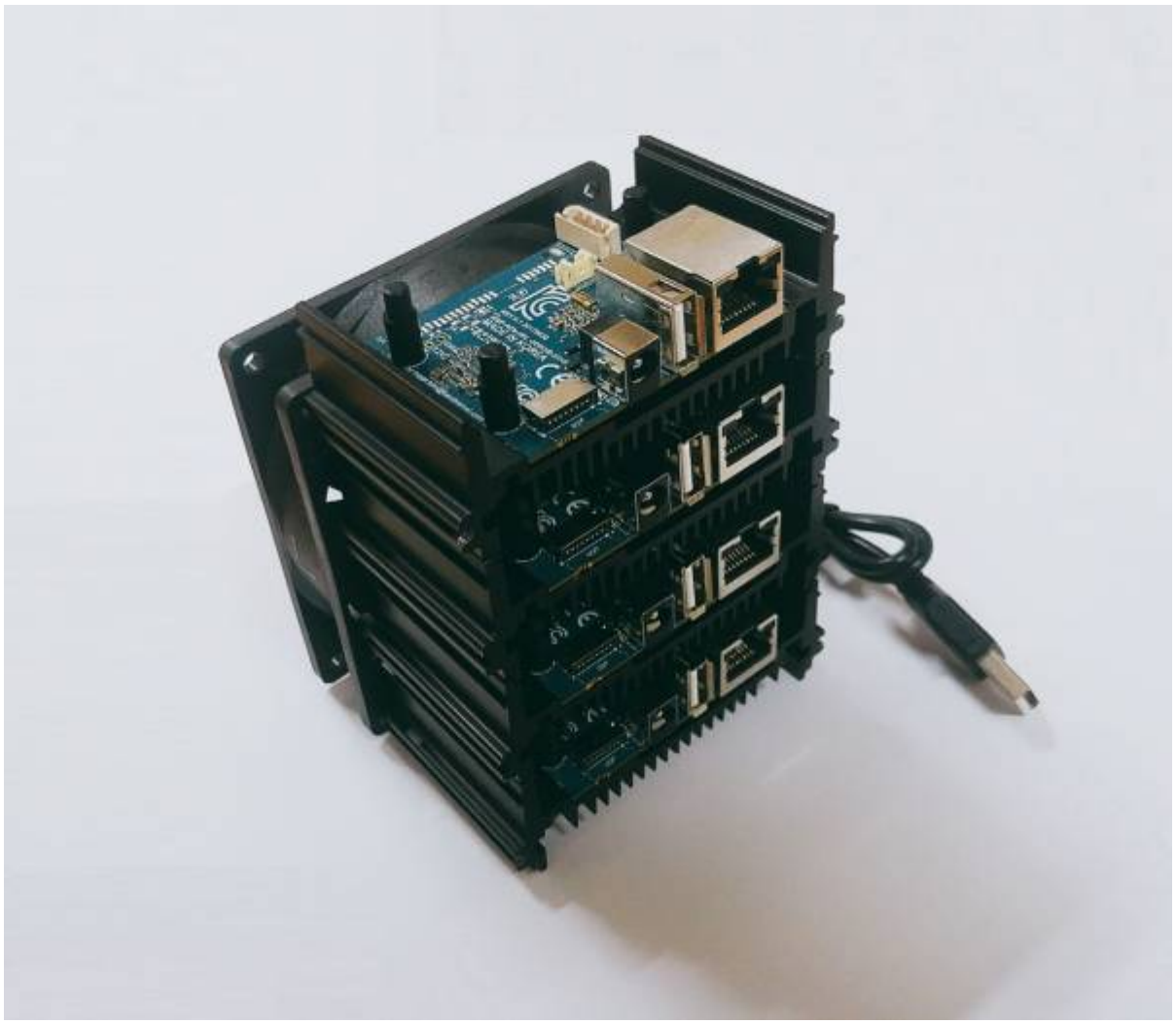


There are two type nodes for the build farm system – ***Build target node*** and ***build computing node***. We use the ODROID-HC1 for build target node. Build target node contains the source codes which will be compiled in this build farm system. ODROID-HC1 is good for build target node because it can have large storage space by attached HDD. It is helpful to improve build performance that the SATA HDD(or SSD) has good I/O performance. Build computing node receives the source file and compiles it. ODROID-MC1 is the best choice for build computing node. MC1 is optimized for distributed computing like build computing node.

Build Target Node - ODROID-HC1 contained the source codes



Build Computing Node - ODROID-MC1



Video

Build Computing Node Configuration

To access ODROID-MC1 and ODROID-HC1 console, you have to get the ip address of the node. Please refer to Boot the ODROID and find IP address section in Headless setup wiki page.

Install *distcc* and configure it. That's all!

```
$ sudo apt update
$ sudo apt install distcc
$ nano /etc/default/distcc
```

- **/etc/default/distcc**

```
STARTDISTCC="true"
ALLOWEDNETS="192.168.100.0/24"
JOBS="8"
ZEROCONF="false"
```

“192.168.100.0/24” ip network range is my local network ip address. It must be modified to suit your network environment.
Restart distcc service.

```
$ sudo /etc/init.d/distcc restart
```

Build Target Node Configuration

Install *distcc*, *distcc-pump* and *distccmon-gnome*. *distcc-pump* is to run distcc pump mode. Distcc's pump mode accelerates remote compilation with distcc by also distributing preprocessing to the servers. *distccmon-gnome* is the distcc monitoring application.

```
$ sudo apt update
$ sudo apt install distcc distcc-pump distccmon-gnome
```

Set the ip addresses of the build computing nodes. In this case, there are 8 build computing nodes (ODROID-MC1s). It is different for each network environment. Write the distcc hosts ip addresses for your environments.

```
$ nano ~/.distcc/hosts
```

- **~/.distcc/hosts**

```
192.168.100.17,lzo,cpp
192.168.100.19,lzo,cpp
192.168.100.23,lzo,cpp
```

```
192.168.100.24,lzo,cpp
192.168.100.26,lzo,cpp
192.168.100.27,lzo,cpp
192.168.100.37,lzo,cpp
```

The examples add the following options to the address:

- lzo: Enables LZO compression for this TCP or SSH host (slave).
- cpp: Enables distcc-pump mode for this host (slave). Note: the build command must be wrapped in the pump script in order to start the include server.

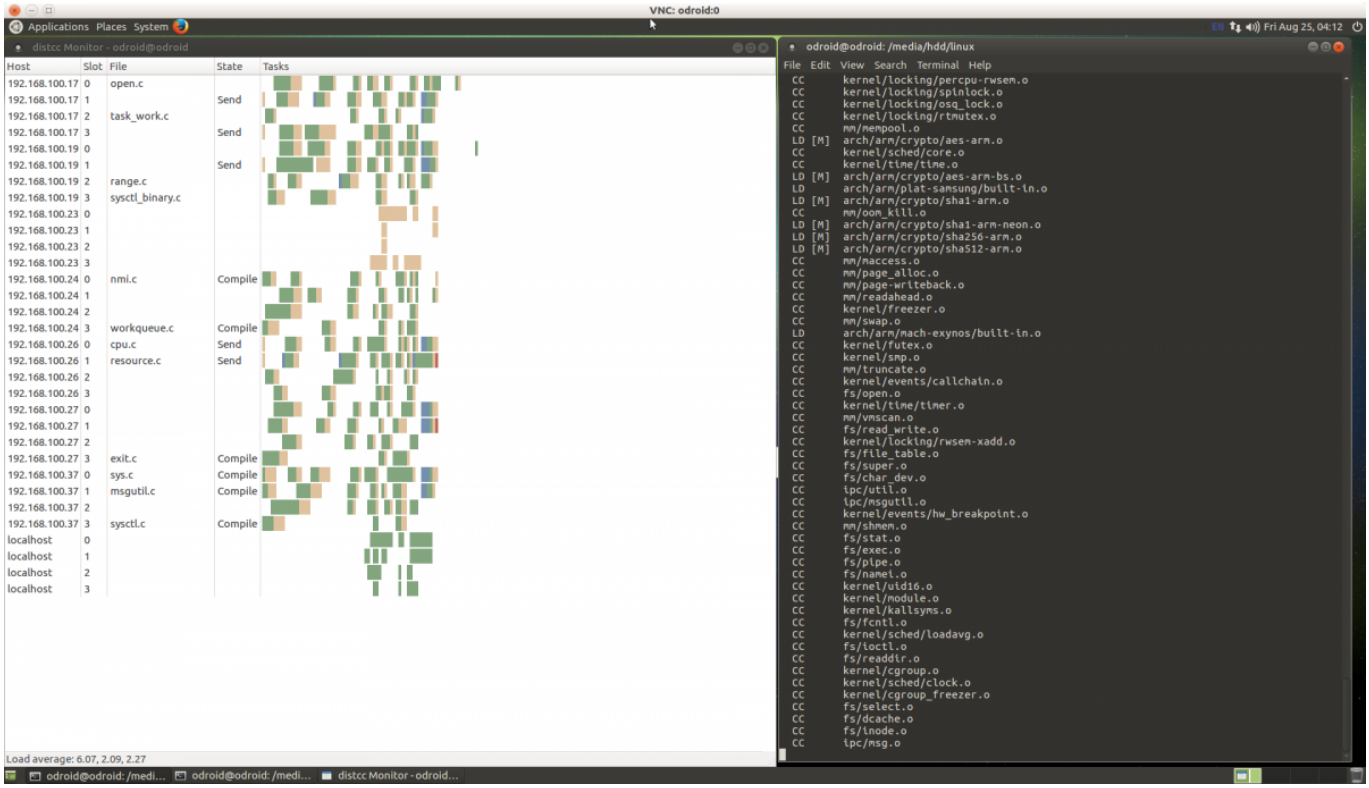
A description for the distcc pump mode can be found here: distcc's pump mode: A New Design for Distributed C/C++ Compilation

Run (Linux kernel compile example)

```
$ distcc-pump make -j64 CC=distcc
```

We can see the distcc compiling status on *distccmon-gnome*.

```
$ distccmon-gnome
```



Build Performance Measurement Experiment

We measured distributed compiling performance according to number of build computing nodes. This is the build time of linux kernel sources for ODROID-XU4. It has better performance than local build

when it is more than 4 build computing nodes because distributed compiling has additional overhead as network. Build commands are as below (On Build Target Node):

```
$ sudo apt update
$ sudo apt install git
$ git clone --depth 1 https://github.com/hardkernel/linux.git -b
odroidxu4-4.9.y
$ cd linux
$ make odroidxu4_defconfig
$ time make -j8
(Local build...)
$ make clean
$ time distcc-pump make -j64 CC=distcc
(Distributed build / 7 nodes...)
```