

ARM Beowolf Cluster

Using big.LITTLE MP Scheduling Architecture, MPICH3, and NFS

Introduction

Through the continued use of cloud computing to establish external computing for everyday users of technology, there is an increasing demand for large computing power. Currently with typical architectures, the computing cluster has been established on typical x86, and x64 architectures. Using these architectures are quite powerful, however they demand a large amount of power at peak performance. Another (possibly not quite economical yet, due to manufacturing costs) methodology is through the use of ARM architecture, could be used. The current configuration of an ARM cluster would demand more nodes than that of a standard x64/x86 architecture, ARM processors meet the needs of less power use, and cost/unit. ARM has become increasingly relevant in today's computing through the use in smartphones and tablets.

Cost/Unit, Cost/Watt, Cost/Performance Breakdown

The choice for a business to choose viable HPC hardware should consider the factors of cost, power, and performance (in general, however there are many other factors such as scaling...). By examining these traits we are able to make an economic and potentially environmentally appealing decision.

Beowolf Cluster

By connecting lots of computers together (Each called a 'node') through a 'backbone', such as a LAN switch, you are able to harness the power of all machines computing in parallel: the catch being, the latency provided by the network speed (typically 10 Gigabit Ethernet or fiber-optic).

A Three Node Experiment

As a hobby I have purchased three ARM nodes to test out a computing cluster, I will also be providing benchmarks at the end of the paper. I will be examining the architecture behind how these low power and low cost (60\$) computers, as well as how they operating on the operating system level through the examination of different kernels/scheduling algorithms.

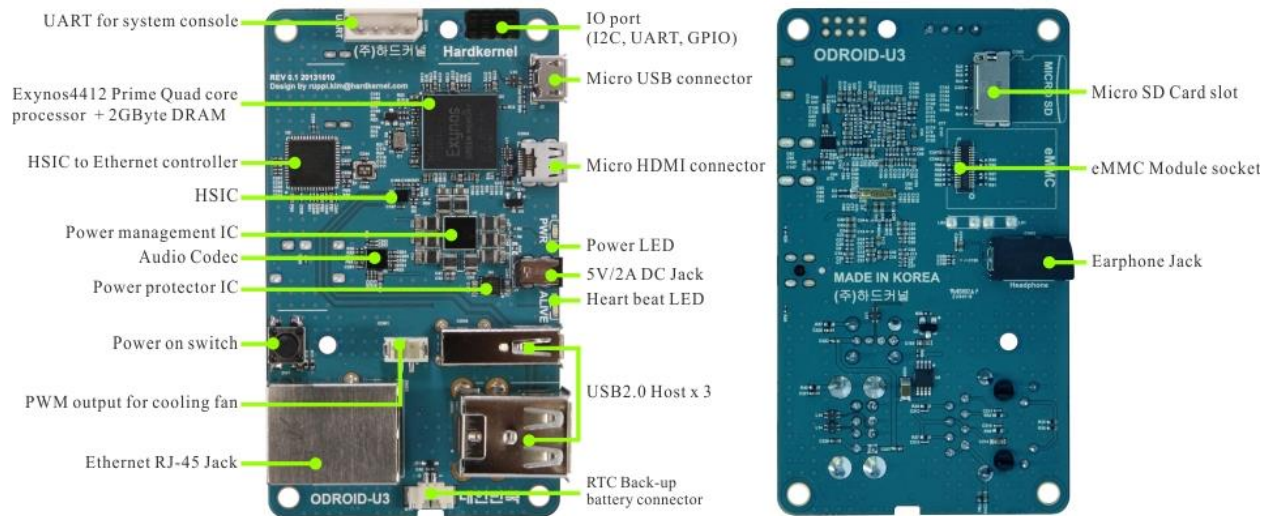
Node Hardware

Hardkernel's Odroid U3

- **Network:** 10/100 Ethernet
- **CPU:** Quad Core 1.7Ghz
- **Memory:** 2GB DRAM
- **Power:** 5 watts / 2 amps peak



Specification Diagram



Node Software & Architecture

XUbuntu 13.10

XUbuntu is a light weight Linux variant of the Ubuntu OS. I have selected a version with a Linux Kernel version above 3.9 which uses a scheduling algorithm that is more productive for use in high performance environments (allows all cores to be active simultaneously). See *Alternate big.LITTLE Scheduling Methodologies* section.

ARM's Big.LITTLE Architecture

The CPU uses a set of ARM A9's and a set of ARM A7 CPUs, and runs seamlessly switching from one to another depending on the task given to them. This architecture was first introduced in 2011. This method of computation will draw much less power than previous architectures with single onboard processors with default process scheduling. The software driving the big.LITTLE architecture is driven by three main software usage models: The Cluster Migration, CPU Migration, and the Global Task Scheduling CPU Migrations. There are three main profiles which are triggered depending on the current

state of events in the software. These profiles are characterized by: High-Intensity workloads (Throw all resources available at it for short period of time), Sustained workloads (throttled by power and thermal limitations), and lastly the Long-use category for low intensity workloads, low power, and long periods of time. These behaviours can be modeled by the graph presented; numbers represented below are in the order of introduction.

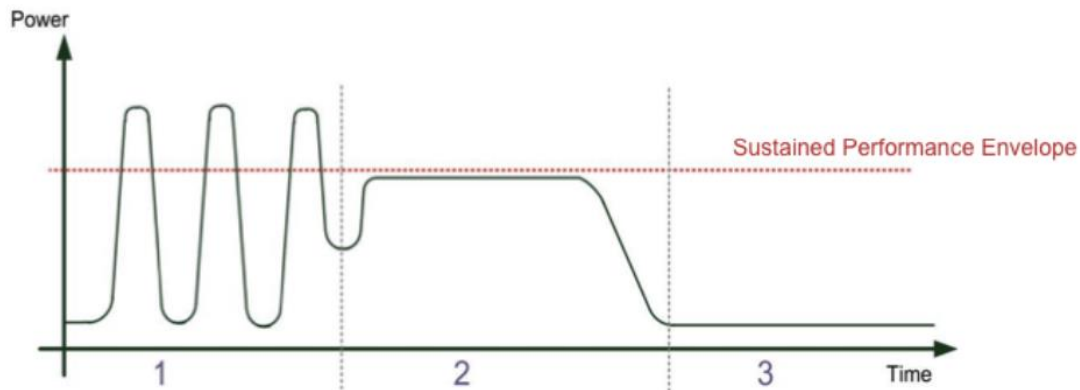


Figure 1: Taken from ARM big.LITTLE GTS Paper [7]

High Intensity Profile: Allocated to Cortex-A9 cores, switching back to A7 during low periods.

Sustained workload Profile: While keeping within the performance envelope above (80% power budget), enables GPU workload to take power precedence over CPU: Mostly allocated Cortex-A9.

Long-use Profile: run entirely on Cortex-A7's, and completely turn off Cortex-15's.

big.LITTLE Architecture and Cluster Computing Concept

During the processing of a clustered task (using MPI), the software must wait for tasks to be delivered on the backbone of the cluster, this allows for the processors to scale down in periods of idle time. However, once a task is delivered through the backbone, the processors scale back to the Cortex-A9's for peak performance, or if the task demands less processing power, processing is handled on the Cortex-A7's. As with any processing, the system must save and fetch to a storage location, whether that be RAM, or any other attached storage device, invariably all processing tasks must pause or halt a computing thread in order to communicate at regular intervals. Through the use of ARM processors, paired together with big.LITTLE architecture, a cluster of nodes would interact at a fraction of the power of standard x86 and x64 architectures, no matter the difference of nm scale of manufacturing.

Multi-Threading and Global Task Scheduling

Another advantage of having access to multiple SoC's at your disposal is the ability to use all of them simultaneously for peak performance. A single task may be able to run in series on the Cortex-A9, but any side tasks can be run in parallel using the slower Cortex-A7 chips simultaneously. This in and of itself can be thought of as a micro-cluster within each node, without the slower latency backbone. The backbone that ARM uses have been named the *coherent interconnect* which provides a bridge between each of the processors via level two (L2) cache (As well as the systems integrated GPU, in the case of our nodes, the Mali 400). In order for the system to operate in parallel, it must be able to wait/poll for results from outside of the main thread.

Comparison to Mainline Linux Scheduler (CFS)

The current equation used in the CFS scheduler for a specific core's CPU load does not take into account the time that is consumed by each task, whereas both scheduling algorithms undertaken by ARM have access to this CPU attribute. Another aspect which the big.LITTLE architecture must implement is the *cpuidle*, and *cpufreq* attributes of a core. These attributes keep track of how long the core will take to wake up from idle state from its current state, and if not idle, the current frequency it is running at to make full use of the task that is required of it. With all of this in mind, the big.LITTLE architecture aims to load balance based on power, rather than spreading load across all CPU's. This scheduler mindset will be counter-productive towards cluster utilizations, where we would like to make full use of the performance at hand for extended periods of time. So really, the only real advantage comes only from the decreased power draw of ARM processors, and budget permitting; the ability to scale the cluster over more nodes for the cost of a standard x86/x64 architectures.

Alternate big.LITTLE Scheduling Methodologies

Linaro is a small non-for-profit company (owned by ARM) which provides open source ARM Linux Kernel and scheduling software. It has produced two scheduling methodologies for the big.LITTLE architecture: the Heterogeneous Multi Processing (**HMP**), and In Kernel Switching (**IKS**). Here are an at-a-glance feature set provided by the two scheduling methodologies:

HMP (Linaro Kernel \geq v3.9)	(Default) IKS
<ul style="list-style-type: none">• All cores can be active simultaneously• Scheduler needs CPU power values• Very complex scheduling for Kernel• Addresses individual cores• Claims to be ~10% performance/watt faster switching decisions	<ul style="list-style-type: none">• CPU's conglomerate into a single virtual one• CPU's are abstracted from kernel, acts as it would for a generic multi-core CPU• Only one core active at a time.

The standard big.LITTLE implementation is to use the IKS switching scheduler, however for cluster environments it would be beneficial to use the HMP scheduler to make use of all active cores, for a small power hit (relative to non-ARM architectures with greater power draws). For the benchmarks used in this paper, I will be using the HMP methodology packaged in XUbuntu 13.10 which uses v.3.11.0.4 and above Linux Kernels.

Message Passing Interface over Chameleon (MPICH)

An open source, high performance, portable implementation of Message Passing Interface (MPI) for desktop systems, shared-memory systems, and multi-core architectures. Also provides support for high-speed networks such as 10 GB/s Ethernet, Infiniband, Myrinet, Quadrics. MPICH is able to compile C and Fortran programming languages to run in cluster environments. It is paramount that a method of communication between node's is established, and for the afore mentioned cluster I will be using MPICH.

For this cluster, I have installed the latest MPICH3 stable release v3.0.4. I will be using C/C++ for the purpose of this paper for benchmarking the 3 node ARM cluster's performance.

Network File System (NFS)

Provides multiple systems with a shared file system, which is essential in Beowolf compute clusters. By using this storage methodology, nodes are able to access files and programs as if they were local to the node. For the purpose of this paper, I will configure NFS to be hosted by node 1: *Alpha* making it a NFS server, and the remaining nodes 2 and 3 (Beta, Gamma) will have the client NFS software installed to access Alpha's */nfs* directory: As configured in the *Node Setup* portion of the document.

It is also important to note that each node is using an 8GB microSD card for internal storage. Benchmark results at the end of the paper are restricted by the class 10 (10MB/s) capable speed of the card.

Node Setup

As this cluster is a compute cluster, there is no point in having a user interface. Each node is headless (command line through SSH)

XUbuntu Node Setup

SSH Commands Issued (Remove XFCE, and other X/Ubuntu software overhead)

```
>> sudo apt-get remove a2ps abiword abiword-common abiword-plugin-grammar abiword-plugin-mathview aumix catfish exo-utils fortune-mod fortunes-min gigolo gnumeric gnumeric-common gnumeric-doc gpicview gtk2-engines-xfce imagemagick imagemagick-doc libaiksaurus-1.2-0c2a libaiksaurus-1.2-data libaiksaurusgtk-1.2-0c2a libdiscid0 libfftw3-3 libgdome2-0 libgdome2-cpp-smart0c2a libgtkmathview0c2a liblink-grammar4 libloudmouth1-0 libmad0 libnotify-bin libofa0 libots0 librcode0 libt1-5 libtagc0 libxfcegui4-4 libxfconf-0-2 link-grammar-dictionaries-en mousepad orage oss-compat psutils python-musicbrainz2 python-mutagen python-ogg python-pymad python-pysqlite2 python-pyvorbis scim-modules-table scim-tables-additional tango-icon-theme tango-icon-theme-common tcl8.4 thunar thunar-archive-plugin thunar-data thunar-media-tags-plugin thunar-volman thunderbird vim-runtime wdiff xchat xchat-common xfce4-appfinder xfce4-battery-plugin xfce4-clipman-plugin xfce4-cpugraph-plugin xfce4-dict xfce4-fsguard-plugin xfce4-mailwatch-plugin xfce4-mixer xfce4-mount-plugin xfce4-netload-plugin xfce4-notes-plugin xfce4-panel xfce4-places-plugin xfce4-quicklauncher-plugin xfce4-screenshooter xfce4-session xfce4-settings xfce4-smartbookmark-plugin xfce4-systemload-plugin xfce4-terminal xfce4-verve-plugin xfce4-weather-plugin xfce4-xkb-plugin xfconf xfdesktop4 xfdesktop4-data xfswitch-plugin xfwm4 xfwm4-themes xubuntu-artwork xubuntu-default-settings xubuntu-desktop xubuntu-docs ca-certificates-java default-jre-headless icedtea-7-jre-jamvm openjdk-7-jre-headless java-common tzdata-java
>> sudo apt-get autoremove
>> sudo apt-get dist-upgrade
```

Networking

The backbone of this cluster will be using a Linksys WRT160N, for a 4 port GbE switch. It is important to note that the ODroid U3 nodes are hardware restricted to a maximum Ethernet speed of 100MB/s, which will be a major constraint to the cluster's overall performance.

IP & Host Name Table

Node	Host Name	Node Local IP
1	Alpha	192.168.1.2
2	Beta	192.168.1.3
3	Gamma	192.168.1.4

```
root@gamma: ~
GNU nanoFile: ...hosts Modified
127.0.0.1 localhost gamma
127.0.0.1 gamma

192.168.1.1 admin-pc
192.168.1.2 alpha
192.168.1.3 beta
192.168.1.4 gamma
192.168.1.77 pi

root@gamma: ~
GNU nano 2.2.6 File: /etc/network/interfaces Modified
# Network Interface - ARM Cluster Node #3 - "Gamma"
auto eth0
iface eth0 inet static
address 192.168.1.4
netmask 255.255.255.0
gateway 192.168.1.254

# set DNS to Google
dns-nameservers 8.8.8.8
```

Install and configure the Network File Server (NFS) on Alpha

```
>> sudo apt-get install nfs-server
>> sudo nano /etc/exports
```

Install and configure NFS on work nodes (Beta, Gamma)

```
>> sudo apt-get install nfs-client
>> sudo mount alpha:/nfs /nfs
>> sudo nano /etc/fstab
```

Define a User for running MPI programs, and configure automatic ssh login,

```
>> sudo useradd mpiu -d /nfs
>> sudo passwd mpiu
>> sudo chown mpiu /nfs
>> sudo ssh-keygen -t rsa
>> ssh-copy-id -i ~/.ssh/id_rsa.pub beta
>> ssh-copy-id -i ~/.ssh/id_rsa.pub gamma
```

NOTE: When I was trying to ping from alpha and beta nodes (and vice versa) they were unable to see

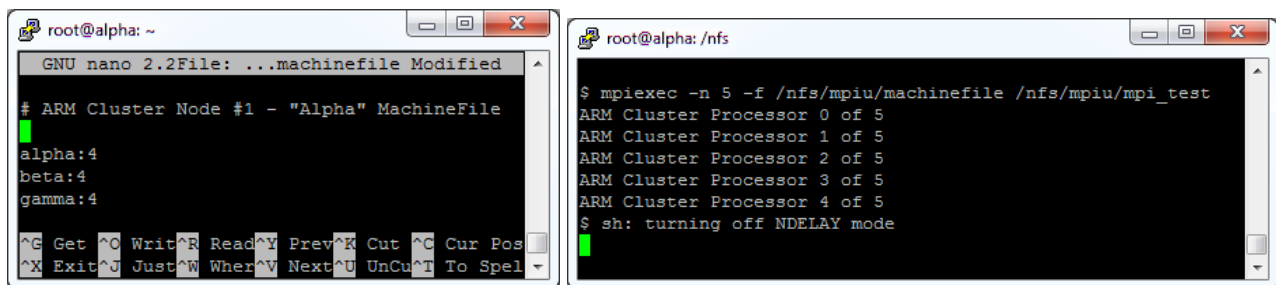
each other. I cloned alpha's image to the others, it must have used the same mac address for both. I reset the mac address of beta with a new key using

```
>> openssl rand -hex 6 | sed 's/(.)/\1:/g; s/$//'  
>> ifconfig eth0 down && ifconfig eth0 hw ether ##:##:##:##:##:## ; ifconfig eth0 up
```

Setting up MPICH Machine File

The Odroid U3 node has 4 processors, so 4 processes on each in the configuration.

```
>> sudo nano /nfs/mpiu/machinefile
```



The image shows two terminal windows. The left window shows the contents of the machinefile: `# ARM Cluster Node #1 - "Alpha" MachineFile
alpha:4
beta:4
gamma:4`. The right window shows the execution of `mpirun -n 5 -f /nfs/mpiu/machinefile /nfs/mpiu/mpiu_test`, which outputs: `ARM Cluster Processor 0 of 5
ARM Cluster Processor 1 of 5
ARM Cluster Processor 2 of 5
ARM Cluster Processor 3 of 5
ARM Cluster Processor 4 of 5
$ sh: turning off NDELAY mode`

Testing MPICH

```
>> nano /nfs/mpiu_test.c  
>> mpicc /nfs/mpiu_test.c -o /nfs/mpiu/mpiu_test  
>> mpirun -n 5 -f /nfs/mpiu/machinefile /nfs/mpiu/mpiu_test
```

```
#include <stdio.h>  
#include <mpi.h>  
  
int main(int argc, char** argv) {  
    int myrank, nprocs;  
  
    MPI_Init(&argc, &argv);  
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);  
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);  
  
    printf("ARM Cluster processor %d of %d\n", myrank, nprocs);  
  
    MPI_Finalize();  
    return 0;  
}
```

Results, Comparison & Benchmarks

Using Phoronix Test Suite with benchmark results posted to openbenchmarking.org for hardware performance comparison.

Upper Bound - Desktop – x64: For direct comparison with moderate-high computing speeds, I will be benchmarking against Intel’s 2600K, providing a quad, eight-thread CPU clocked in at a stock 3.4Ghz with 16GB of memory – CPU Runs for ~\$300-320

Lower Bound – ARMv6: For direct comparison with low end ARM computing speeds, I will be benchmarking against a Raspberry Pi Model B, clocked in at 700 MHz single core, with 512MB of memory. Full board costs ~\$30

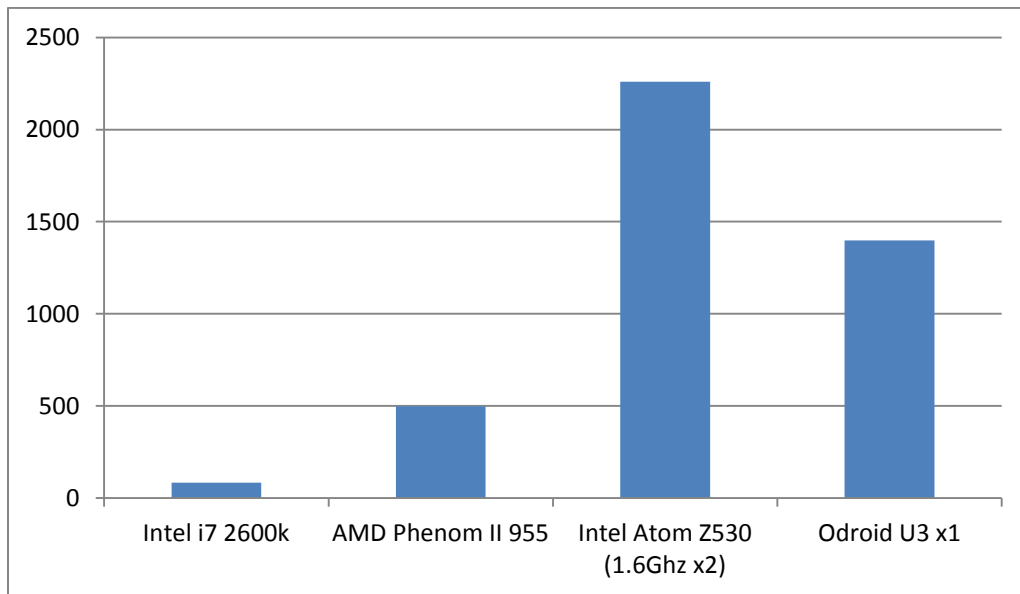
Cluster Scaling – ARMv7: For scaling comparison of the ARM cluster, I will be benchmarking a single, double, and triple node configurations, each with stock 1.7 GHz clock, with 2GB of memory. Each node costs ~60\$

CPU’s Mentioned otherwise have been pulled from <http://openbenchmarking.org/> which is tied into the benchmark posting process of Phoronix Test Suite.

Processor Benchmarks – Primesieve Benchmark

pts/primesieve

Only benchmark I was able to run on *Phoronix Test Suite* that was cross platform ARM and x64 for testing processor performance. Note: In terms of seconds to run test, lower is better.

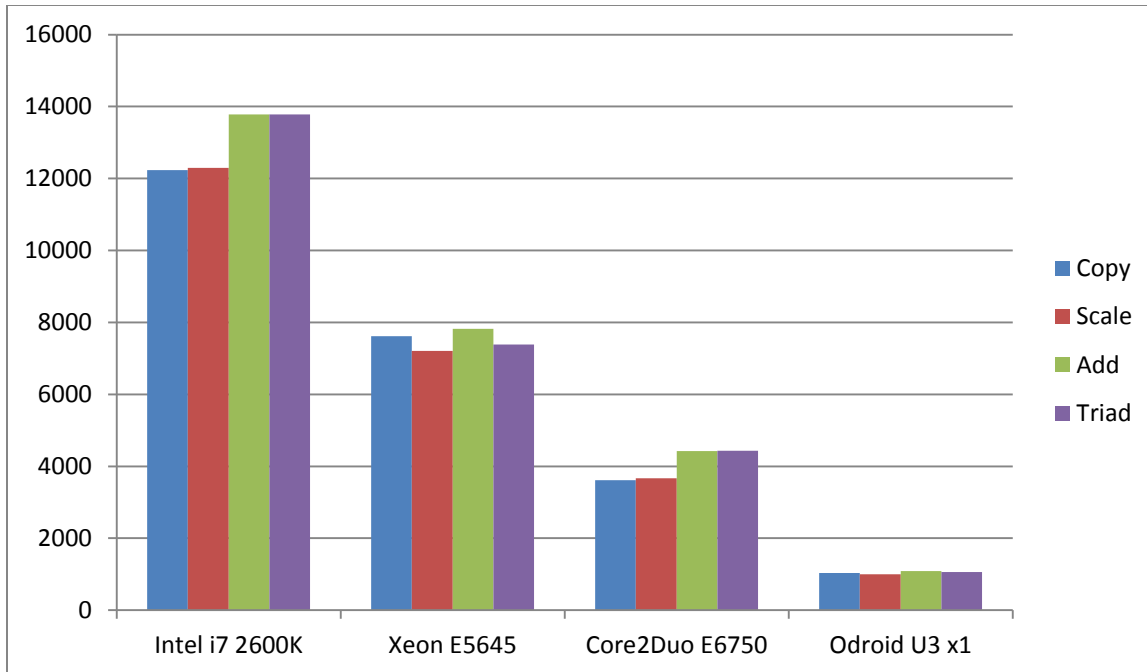


Using High Performance Linpack (HPL) to determine a nodes GFLOP/s:

Memory & IO

pts/stream

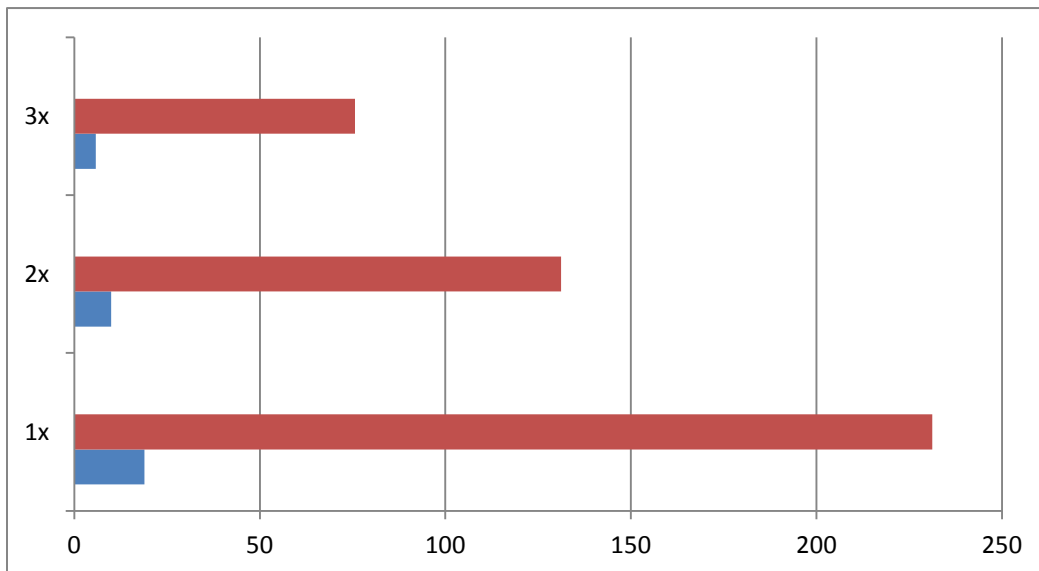
Tests copying, scale, add and triad functions of memory, Higher is better.



Cluster Scaling - NAS Parallel Benchmarks Test (NPB)

pts/npb

Provided by NASA, quite renowned for testing the limitations of clusters through five main Kernels and three pseudo-applications. I will be testing two: *Embarrassingly Parallel (EP)* class A problem size, and a pseudo application: LU class A problem size – *Lower-Upper Gauss-Seidel Solver (LU)*.



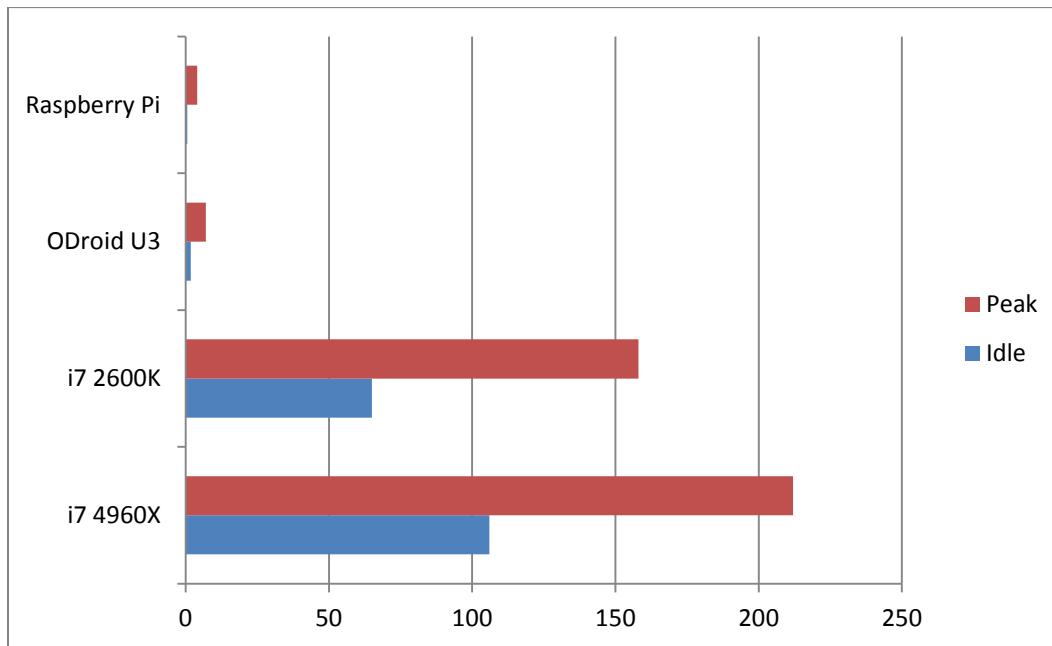
Scaling Ratio: LU (1-2): **~1.7633**, LU (2-3): **~1.733**, EP (1-2) **~1.9011**, EP (2-3) **~1.7251**

Average Scaling Ratio: ~1.781

Power Consumption

pts/battery-power-usage

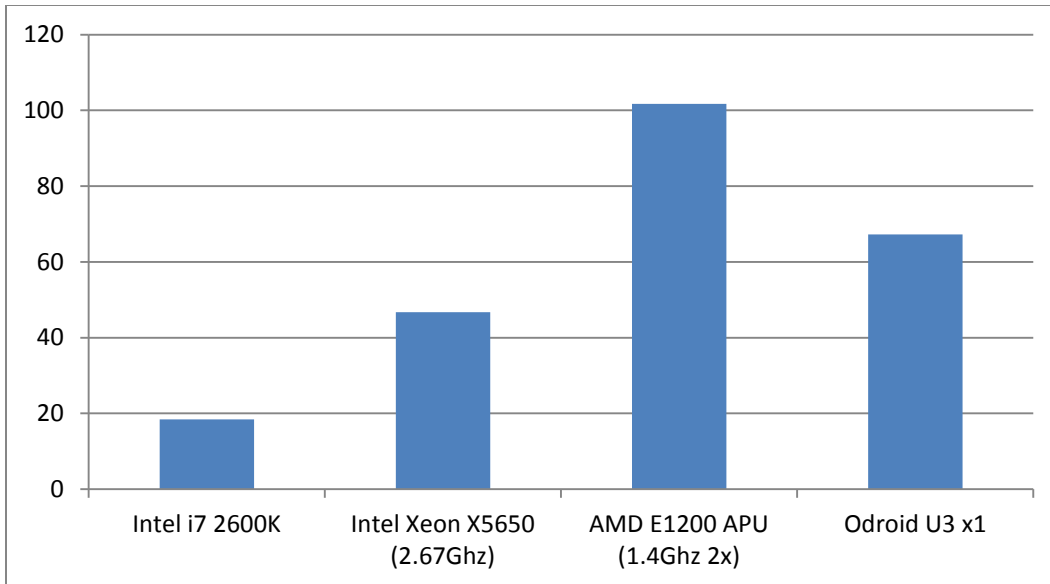
Power consumption of the Raspberry Pi's ARMv6 clocked at 700Mhz, ODroid U3's ARMv7 Quad Cortex-A9's at 1.7Ghz, Intel i7 2600k Quad, eight-thread, clocked at 3.4Ghz, and lastly the current top-end processor on the market the Intel i7 4960X clocked at 3.6Ghz, with six-core, twelve thread combination [13]



Cluster Networking

pts/network-loopback

TCP Network Performance Loopback Benchmark (Local capable network speed), results in seconds, lower is better.



Conclusion

Product	Watt (Peak)	Performance	Performance/Watt
1x Raspberry Pi	4w	0.175 GFLOP/s	0.04375
1x ODroid U3	5w	3.79 GFLOP/s	0.758
8x ODroid U3	32w	14.79 GFLOP/s	0.462
32x ODroid U3	128w	94.67 GFLOP/s	0.73960
40x ODroid U3	160w	118.34 GFLOP/s	0.73964
1x Intel i7 2600K	158w	49.459 GFLOP/s	0.313

Scaling compensation of 1.780625 as established in *Cluster Scaling Benchmark* above.

Power Scaling compensation: $-1w/node$ added (not all peaking because of network idling)

Note: At around 40 node U3 cluster, we reach the same power consumption of the i7 2600K, however the cost difference is a rather whopping **\$2400** at 60\$ / unit. However, for that cost you will have a computing power of 118 GFLOP/s which is almost **2.4x** more powerful than a single Intel i7 2600K assuming scaling and other network overhead hold true.

Should technology continue to move in the way it is currently moving; towards the betterment of mobile/integrated technology, and the demand to meet lower power but extremely powerful computing clusters, I would expect ARM to fit that demand. We are not too far off from the development and lower cost of more powerful mobile technology, which was simply not available in previous years. Through the use of architectures like big.LITTLE, to efficiently manage clusters of processors in a single SoC, as well as the continued efforts towards improving the Linux Kernel to support such technologies in an open-source basic, I can see a bright future for HPC taking a more efficient path towards the use of less power.

References

- [0] <http://yclept.ucdavis.edu/Beowulf/aboutbeowulf.html>
- [1] <http://www.mpich.org/>
- [2] <http://xubuntu.org/tour/>
- [3] <http://xubuntu.org/about/>
- [4] <https://help.ubuntu.com/community/MpichCluster>
- [5] <http://www.phoronix-test-suite.com/?k=home>
- [6] <http://www.arm.com/products/processors/technologies/biglittleprocessing.php>
- [7] http://www.arm.com/files/pdf/big_LITTLE_technology_moves_towards_fully_heterogeneous_Global_Task_Scheduling.pdf
- [8] <http://www.linuxplumbersconf.org/2012/wp-content/uploads/2012/09/2012-lpc-scheduler-task-placement-rasmussen.pdf>
- [9] https://events.linuxfoundation.org/images/stories/slides/elc2013_poirier.pdf
- [10] <http://linuxgizmos.com/linaro-optimizes-linux-support-for-arm-big-little/>
- [11] <https://help.ubuntu.com/12.04/serverguide/network-file-system.html>
- [12] <http://www.cs.washington.edu/research/projects/zpl/comm/mpi/mpich/examples/perftest/README>
- [13] <http://www.anandtech.com/show/7255/intel-core-i7-4960x-ivy-bridge-e-review>
- [14] http://www.nvidia.com/content/PDF/GDC2011/Alex_Ramirez_SC11.pdf
- [15] <http://www.montblanc-project.eu/sites/default/files/publications/armhpc-sc.pdf>