1. Overview

1.1 Overview

This document describes performance evaluation of R-Car Gen 3 Linux BSP.

In this document, the background color of the text-box means the following:

- White: Operation and display on Host PC
- Black: Operation and display on the evaluation board

1.2 Evaluation Item

Evaluation item is described below.

Table 1.1 Evaluation Item

Item	Content
CPU	CPU computing capability is evaluated by **SPEC2000**, **SPEC2006, Dhrystone**, and **Linpack**.
Memory	Memory perfromance is evaluated by **CacheBench**, **lmbench**, and **pmbw**.
System	Whole system performance is evaluated by **UnixBench**.
File System	File System performance is evaluated by **IOzone** and **bonnie++**.
Driver	Driver performance is evaluated by **dd**.
Boot Process	Boot Process performance is evaluated by **bootchart**.

1.3 Reference

Related material is listed below.

Table 1.2 Reference (R-Car H3/M3)

No	Issued by	Title	Revision	Date
1	Renesas Electronics	R-Car Series, 3rd Generation User's Manual: Hardware	Rev. 0.52 Draft	Mar. 2016
2	Renesas Electronics	R-CarH3-SiP System Evaluation Board RTP0RC7795SIPB0011S (Salvator-X) Hardware Manual	Rev.1.00	Mar. 2016
3	Renesas Electronics	Linux Interface Specification Yocto recipe Start-Up Guide	2.19.0	Apr. 2017

2. Working Environment

2.1 Hardware

The hardware is used to evaluate performance is listed below.

Table 2.1 Hardware (R-Car H3/M3)

Name	Version	Manufacturer
R-CarH3-SiP System Evaluation Board RTP0RC7795SIPB0011S (Salvator-X)	—	Renesas Electronics

2.2 Software

These evaluation results were obtained based on the following software.

Table 2.2 Software

Name	Version	Note
Yocto Package	2.19.0	Use BSP configuration only (except bootchart)
Linux BSP, U-boot	3.5.3	Built by Yocto
Linaro-5.2-2015.11-2	5.2	Exported by Yocto
Dhrystone	[T.B.D.]	http://www.netlib.org/benchmark/dhry-c
Linpack	[T.B.D.]	http://www.netlib.org/benchmark/linpackc.new
SPEC2000	1.3.1	cpu2000-1.3.1.iso
SPEC2006	1.2	cpu2006.tar.xz
UnixBench	5.1.3	https://github.com/kdlucas/byte-unixbench http://code.google.com/p/byte-unixbench/
Perl	5.22.0	Used to running UnixBench
LLCbench	[T.B.D.]	http://icl.cs.utk.edu/llcbench/index.htm
LMbench	3.0-a9	http://www.bitmover.com/lmbench/
pmbw	0.6.2	https://panthema.net/2013/pmbw/ https://github.com/bingmann/pmbw/
IOzone	3.434	http://www.iozone.org/
bonnie++	1.03e	http://www.coker.com.au/bonnie++/
bootchart	1.16	https://github.com/sofar/bootchart

NOTICE:

In this document, rootfs is assumed to be placed on SD card.

2.2.1 Path of toolchain

In the command line examples in this document, <toolchain_path> appears some times. This must be replaced with the path string where the toolchain is installed.

The path string depends on the version of Yocto package and the environment of Host PC. When executing the procedure in this document, please replace it with the path string corresponding to each environment.

The default for the path string in the Yocto package version targeted by this document is "**/opt/poky/2.1.2**".

2.3 Checking build date

In order to prevent IPL, U-boot, and kernel update omissions, we must check the build dates and make sure none of them are old before executing each evaluation. As an example, the check points of board boot logs are shown in **red bold** below.

```
NOTICE:  BL2: R-Car Gen3 Initial Program Loader(CA57) Rev.1.0.9.10
...
NOTICE:  BL2: Built : 02:20:59, Mar 25 2017
...
U-Boot 2015.04 (Mar 25 2017 - 02:20:46)
...
Starting kernel ...
...
[    0.000000] Linux version 4.6.0-yocto-standard (builduser@buildmachine) (gcc version 5.2.1 20151005
(Linaro GCC 5.2-2015.11-2) ) #1 SMP PREEMPT Sat Mar 25 01:33:28 JST 2017
...
```

For the timing of this check, follow the procedure of each evaluation.

3. CPU Performance Evaluation

This chapter describes procedures and results for evaluating CPU performance.

3.1 Dhrystone [T.B.D.]

[T.B.D]

3.2 Linpack [T.B.D.]

[T.B.D]

3.3 SPEC2000

Please refer to “SPEC CPU2000_Operation Procedure.docx”.

3.4 SPEC2006

Please refer to “SPEC CPU2006_Operation Procedure.docx”.

4. Memory Performance Evaluation

This chapter describes procedures and results for evaluating memory performance.

4.1 CacheBench [T.B.D.]

[T.B.D]

4.2 LMbench

LMbench is a suite of simple, portable, ANSI/C microbenchmarks for UNIX/POSIX. In general, it measures two key features: latency and bandwidth

- Bandwidth benchmarks
  - Cached file read
  - Memory copy (bcopy)
  - Memory read
  - Memory write
  - Pipe
  - TCP
- Latency benchmarks
  - Context switching.
  - Networking: connection establishment, pipe, TCP, UDP, and RPC hot potato
  - File system creates and deletes.
  - Process creation.
  - Signal handling
  - System call overhead

4.2.1 Evaluation Procedure

(1) Download and extract LMbench package

- Download the package at the below link:
  https://sourceforge.net/projects/lmbench/files/development/lmbench-3.0-a9/lmbench-3.0-a9.tgz
- Extract the package by Linux command
- Replace gnu-os script with aarch64 compatible version

```
$ tar xf lmbench-3.0-a9.tgz
$ ssh 172.29.143.231 # server gitlab
$ export http_proxy="http://user_name:Pass_window_log_in@172.29.137.2:8080"
$ wget "http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.guess;¥
hb=6a82322dd05cdc57b4cd9f7effdf1e2fd6f7482b" -O lmbench-3.0-a9/scripts/gnu-os
```

Note: user_name is your name of Linux account (ex: dangle), Pass_windown_log_in is your pass to login Linux account.

(2) Applying patches

- Save the contents of the following text-box as a patch file, and apply the patch to the source package.

```
diff -uprN a/scripts/config b/scripts/config
--- a/scripts/config 2000-02-01 09:29:31.000000000 +0900
+++ b/scripts/config 2017-03-17 10:14:26.465884472 +0900
@@ -1,6 +1,7 @@
 #!/bin/sh

-UNAME=`uname -n 2>/dev/null`
+#UNAME=`uname -n 2>/dev/null`
+UNAME="salvator-x"
 if [ X$UNAME = X ]
 then	echo CONFIG
 else	echo CONFIG.$UNAME
diff -uprN a/scripts/gnu-os b/scripts/gnu-os
--- a/scripts/gnu-os 2017-03-17 10:19:40.445891142 +0900
+++ b/scripts/gnu-os 2017-03-17 10:15:51.577886280 +0900
@@ -127,7 +127,8 @@ if (test -f /.attbin/uname) >/dev/null 2
 	PATH=$PATH:/.attbin ; export PATH
 fi

-UNAME_MACHINE=`(uname -m) 2>/dev/null` || UNAME_MACHINE=unknown
+#UNAME_MACHINE=`(uname -m) 2>/dev/null` || UNAME_MACHINE=unknown
+UNAME_MACHINE="aarch64"
 UNAME_RELEASE=`(uname -r) 2>/dev/null` || UNAME_RELEASE=unknown
 UNAME_SYSTEM=`(uname -s) 2>/dev/null`  || UNAME_SYSTEM=unknown
 UNAME_VERSION=`(uname -v) 2>/dev/null` || UNAME_VERSION=unknown
diff -uprN a/scripts/lmbench b/scripts/lmbench
--- a/scripts/lmbench		2006-06-28 01:27:19.000000000 +0900
+++ b/scripts/lmbench		2017-03-17 15:20:26.634274514 +0900
@@ -470,7 +470,7 @@ if [ X$BENCHMARK_HARDWARE = XYES -o X$BE
 	echo "Memory load latency" 1>&2
 	if [ X$FASTMEM = XYES ]
 	then	lat_mem_rd -P $SYNC_MAX $MB 128
-	else	lat_mem_rd -P $SYNC_MAX $MB 16 32 64 128 256 512 1024
+	else	lat_mem_rd -P $SYNC_MAX $MB 16 32 64 128 256 512 1024 1048576
 	fi
 	echo "" 1>&2
 	echo "Random load latency" 1>&2
diff -uprN a/src/Makefile b/src/Makefile
--- a/src/Makefile	2007-04-10 21:16:49.000000000 +0900
+++ b/src/Makefile	2017-03-22 20:11:38.191823049 +0900
@@ -34,9 +34,9 @@
 # I finally know why Larry Wall's Makefile says "Grrrr".
 SHELL=/bin/sh

-CC=`../scripts/compiler`
+#CC=`../scripts/compiler`
 MAKE=`../scripts/make`
-AR=ar
+#AR=ar
 ARCREATE=cr

 # base of installation location
```

- Here is an example of command execution when the patch file is saved as **cross-compile-lmbench-3.0-a9.patch**:

```
$ cd lmbench-3.0-a9
$ patch -p1 < cross-compile-lmbench-3.0-a9.patch
patching file scripts/config
patching file scripts/gnu-os
patching file scripts/lmbench
patching file src/Makefile
```

(3) Optimization Options

- lmbench-3.0-a9/src/Makefile

```
@env CFLAGS=-O MAKE="$(MAKE)" MAKEFLAGS="$(MAKEFLAGS)" CC="$(CC)" OS="$(OS)" ../scripts/build all
-@env CFLAGS=-O MAKE="$(MAKE)" MAKEFLAGS="k$(MAKEFLAGS)" CC="$(CC)" OS="$(OS)" ../scripts/build opt
```

(4) Compile the package

- Replace <toolchain_path> with the path string where the cross toolchain is installed. For details, refer to **2.2.1 Path of toolchain**.

```
$ source <toolchain_path>/environment-setup-aarch64-poky-linux
$ export LDFLAGS=""
$ make
```

(5) Copy the entire LMbench folder to target file system

```
$ cp -r lmbench-3.0-a9 <rootfs>/home/root
```

(6) Check build date

- Boot the system and check the build date of the environment from the serial log. Please refer to **2.3 Checking build date** for the check points.

(7) Execute the benchmark on target

1. Before executing the benchmark, modify the following files.
   - lmbench-3.0-a9/scripts/lmbench
     Add 1M to the stride setting of "Memory load latency"
     * This modification is included in **(2) Applying patches**

```
-        else    lat_mem_rd -P $SYNC_MAX $MB 16 32 64 128 256 512 1024
+        else    lat_mem_rd -P $SYNC_MAX $MB 16 32 64 128 256 512 1024 1048576
```

(8) Make file can be executed

```
$ cd /home/root/lmbench-3.0-a9/scripts/
$ chmod +x *
```

2. Create a config file.
   - When executing config-run, you need to enter parameters.

```
# cd lmbench-3.0-a9/src/
# ../scripts/config-run
```

- When you run make results, LMbench will let you do some choose to configure your test. Here are each items mean and some suggest.
  - **MULTIPLE COPIES [default 1]**

If you are running on a SMP machine, LMbench will allow to run multiple copies of lmbench in parallel, choose the default value(**1**) should be OK.

- **Job placement selection:**

You must choose a This item allow to choose LMbench how to schedule the test. If you don't know what the seven option mean, just choose **1)**.

- **MB [default 695]**

Several benchmarks operate on a range of memory. This memory should be sized such that it is at least **4 times** as big as the external cache[s] on your system. It should be no more than **80%** of your physical memory.

- **SUBSET (ALL|HARWARE|OS|DEVELOPMENT) [default all]**

LMbench measures a wide variety of system performance, and the full suite of benchmarks can take a long time on some platforms. Consequently, LMbench offer the capability to run only predefined subsets of benchmarks, one for operating system specific benchmarks and one for hardware specific benchmarks. LMbench also offer the option of running only selected benchmarks which is useful during operating system development. If you don't know which to choose, just choose the default value (**ALL**).

- **FASTMEM [default no]**

This benchmark measures, by default, memory latency for a number of different strides. That can take a long time and is most useful if you are trying to figure out your cache line size or if your cache line size is greater than 128 bytes. For xen performance testing, i suggest to choose the default value (**no**).

- **SLOWFS [default no]**

This benchmark measures, by default, file system latency. That can take a long time on systems with old style file systems (i.e., UFS, FFS, etc.). Linux' ext2fs and Sun's tmpfs are fast enough that this test is not painful. For xen performance testing, i suggest to choose the default value (**no**).

- **DISKS [default none]**

This option is used to determine whether to test disk performance. If you want to skip the disk tests, hit return below. If you want to include disk tests, then specify the path to the disk device, such as /dev/sda. For xen testing, we only use LMbench to test guest's memory performance, no disk testing include, so choose the default value (**none**).

- **REMOTE [default none]**

This option is used to determine whether to test network performance. If you want to skip the network tests, hit return below. If you want to include network tests, then specify the ip address of other host. For xen testing, we only use LMbench to test guest's memory performance, no network testing include, so choose the default value (**none**).

- **Processor mhz [default 1579 MHz, 0.6333 nanosec clock]**

The default value for 'Processor mhz' maybe wrong, you need to input the correct value according to your cpu frequency.

- **FSDIR [default /usr/tmp]**

The directory to store a test file as well as create and delete a large number of small files. For xen testing, just choose the default value (**/usr/tmp/**).

- **Status output file [default /dev/tty]**

LMbench outputs status information as it runs various benchmarks. For xen testing, just choose the default value (**/dev/tty**).

- **Mail results [default yes]**

  There is a database of benchmark results that is shipped with new releases of lmbench. Your results can be included in the database if you wish. For xen testing, there is no need to send the results, so set the value to **no**.

An example of parameter input is described below. Please enter the **red boldface** part.

```
MULTIPLE COPIES:<Enter>
Job placement selection:<Enter>
MB:64 <- Specify the amount of memory to allocate (used with file IO and memory)
SUBSET (ALL|HARWARE|OS|DEVELOPMENT):<Enter>
FASTMEM:<Enter>
SLOWFS:yes
DISKS:<Enter>
REMOTE:<Enter>
Processor mhz:<Enter>
FSDIR:/tmp <- Specify the directory on the target file system
Status output file:<Enter>
Mail results:no
```

3. Execute the benchmark.

```
# ../scripts/results
```

4.2.2 Evaluation Result

4.2.2.1 Overview

LMbench can acquire execution results in text format and graph form.
The output procedure of the result is described below.

All the following procedures are executed on the Host PC.
Before execution, it is necessary to copy the lmbench-3.0-a9 directory of the target file system to an arbitrary directory of Host PC.

(1) Procedure to output in text format

```
$ cd lmbench-3.0-a9/results
$ make summary
```

The result in text result is the latency such as:

- "Processor, Processes - times in microseconds - smaller is better" is lmbench latencies for selected processor/process activities. The values are all times in microseconds averaged over ten independent runs (with error estimates provided by an unbiased standard deviation), so ``smaller is better''.
- Basic integer operations - times in nanoseconds: performace calculate operation type interger.
- Basic uint64 operations - times in nanoseconds: performace calculate operation type unsigned 64-bit integer.
- Basic float operations - times in nanoseconds: performace calculate operation type float.
- Basic double operations - times in nanoseconds: performace calculate operation type double.
- Context switching - times in microseconds: ability to change from current process to other process.
- *Local* Communication latencies in microseconds: abiliy to connect to local network.
- *Remote* Communication latencies in microseconds: ability to connect to other machine.
- File & VM system latencies in microseconds: ability to create/delete files and virtual machine of that system.
- Memory latencies in nanoseconds: performance acess memory.

The result in text result is the bandwidth such as:

- *Local* Communication bandwidths in MB/s: ability to transfer data.

For more detai, refer site:

http://webhome.phy.duke.edu/~rgb/Beowulf/beowulf_book/beowulf_book/node40.html

(2) Procedure to output in graph form

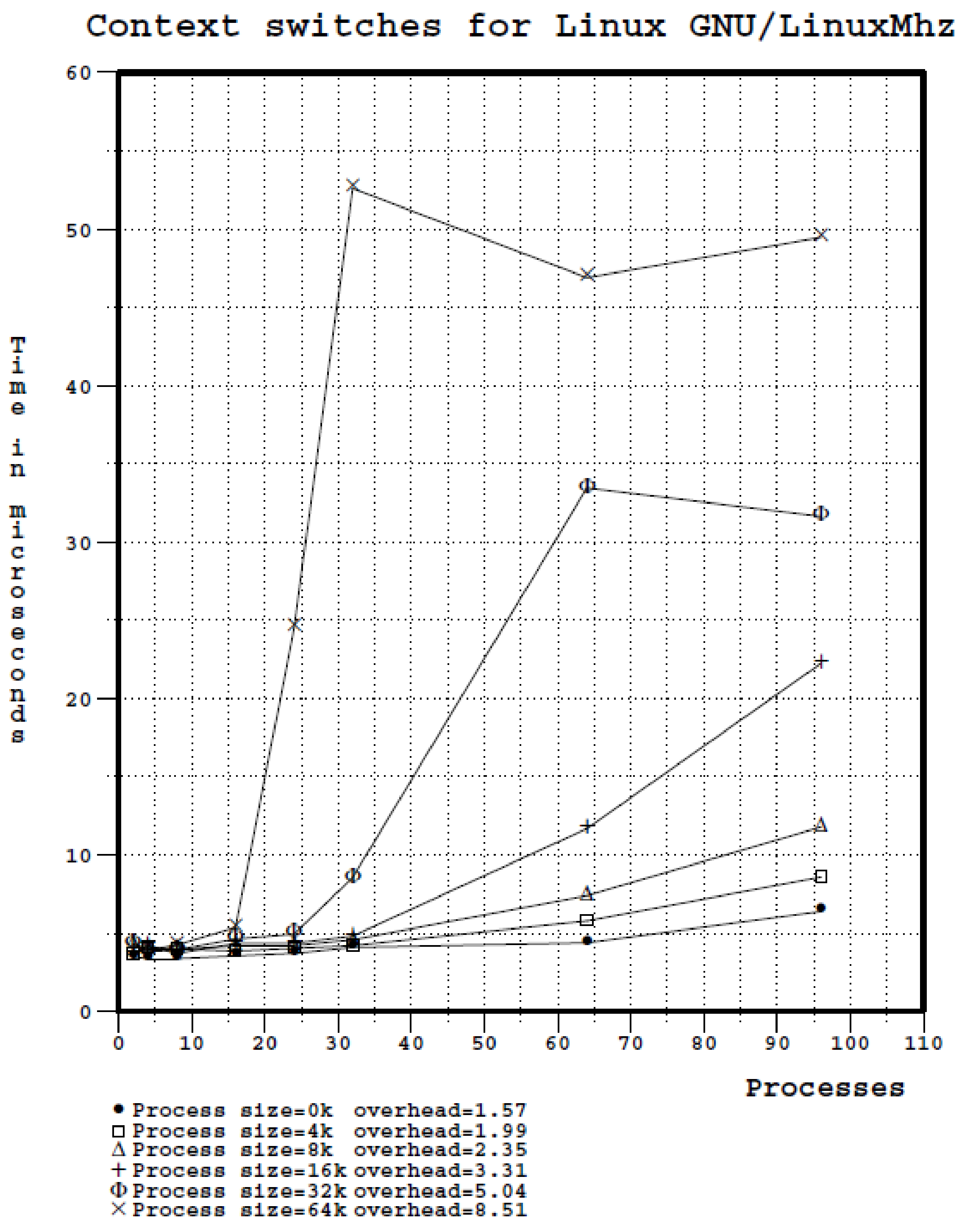
```
$ cd lmbench-3.0-a9/results
$ make ps
$ cd PS
$ ps2pdf14 PS PS.pdf
$ ps2pdf14 PS.1 PS1.pdf
$ ps2pdf14 PS.2 PS2.pdf
$ ps2pdf14 PS.3 PS3.pdf
```

4.2.2.2 Evaluation Result

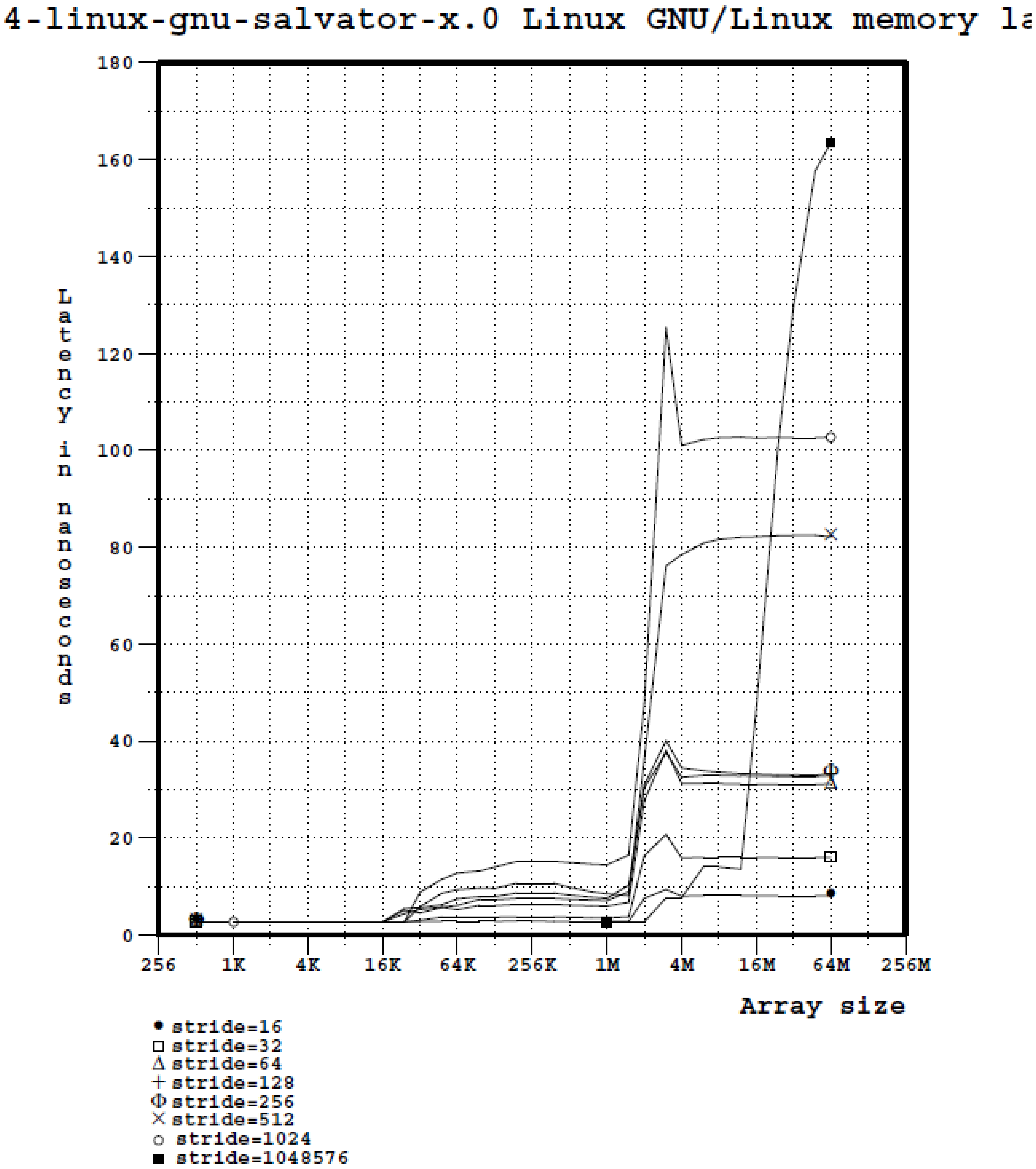
- H3(CA57) Text

```
$ make summary

                 L M B E N C H  3 . 0   S U M M A R Y
                 ------------------------------------
                 (Alpha software, do not distribute)

Basic system parameters
------------------------------------------------------------------------------
Host                 OS Description              Mhz  tlb  cache  mem   scal
                                                     pages line   par   load
                                                           bytes
--------- ------------ ---------------------- ---- ----- ----- ------ ----
salvator- Linux 4.6.0-y       aarch64-linux-gnu 1499    32    64 5.8600    1

Processor, Processes - times in microseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS  Mhz null null      open slct sig  sig  fork exec sh
                             call  I/O stat clos TCP  inst hndl proc proc proc
--------- ------------ ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
salvator- Linux 4.6.0-y 1499 0.36 0.45 1.18 2.95 7.09 0.54 2.26 264. 803. 2481

Basic integer operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  intgr intgr  intgr  intgr  intgr
                          bit   add    mul    div    mod
--------- ------------ ------ ------ ------ ------ ------
salvator- Linux 4.6.0-y 0.6700 0.6700        7.3500 6.0100

Basic uint64 operations - times in nanoseconds - smaller is better
------------------------------------------------------------------
Host                 OS int64  int64  int64  int64  int64
                         bit    add    mul    div    mod
--------- ------------ ------ ------ ------ ------ ------
salvator- Linux 4.6.0-y  0.670                 10.4 7.3500

Basic float operations - times in nanoseconds - smaller is better
-----------------------------------------------------------------
Host                 OS  float  float  float  float
                         add    mul    div    bogo
--------- ------------ ------ ------ ------ ------
salvator- Linux 4.6.0-y 3.3400 4.0100   12.0 5.3400

Basic double operations - times in nanoseconds - smaller is better
------------------------------------------------------------------
Host                 OS  double double double double
                         add    mul    div    bogo
--------- ------------ ------  ------ ------ ------
salvator- Linux 4.6.0-y 3.3400 4.0000   21.4   20.0

Context switching - times in microseconds - smaller is better
-------------------------------------------------------------------------
Host                 OS  2p/0K 2p/16K 2p/64K 8p/16K 8p/64K 16p/16K 16p/64K
                         ctxsw  ctxsw  ctxsw ctxsw  ctxsw   ctxsw   ctxsw
--------- ------------ ------ ------ ------ ------ ------ ------- -------
salvator- Linux 4.6.0-y 3.4600 3.6500 3.7600 3.6900 4.2400 4.34000 5.36000

*Local* Communication latencies in microseconds - smaller is better
---------------------------------------------------------------------
Host                 OS 2p/0K  Pipe AF     UDP  RPC/   TCP  RPC/ TCP
                        ctxsw       UNIX         UDP         TCP conn
--------- ------------ ----- ----- ---- ----- ----- ----- ----- ----
salvator- Linux 4.6.0-y 3.460  10.9 13.3  22.2        28.9        50.

*Remote* Communication latencies in microseconds - smaller is better
---------------------------------------------------------------------
Host                 OS   UDP  RPC/  TCP   RPC/ TCP
                               UDP         TCP  conn
--------- ------------ ----- ----- ----- ----- ----
salvator- Linux 4.6.0-y

File & VM system latencies in microseconds - smaller is better
-------------------------------------------------------------------------------
Host                 OS   0K File      10K File     Mmap    Prot   Page   100fd
                        Create Delete Create Delete Latency Fault  Fault  selct
--------- ------------ ------ ------ ------ ------ ------- ----- ------- -----
salvator- Linux 4.6.0-y                              511.0 0.351 0.65920 2.576

*Local* Communication bandwidths in MB/s - bigger is better
-----------------------------------------------------------------------------
Host                OS  Pipe AF    TCP  File   Mmap  Bcopy  Bcopy  Mem   Mem
                             UNIX      reread reread (libc) (hand) read write
--------- ------------ ---- ---- ---- ------ ------ ------ ------ ---- -----
salvator- Linux 4.6.0-y 1940 3338 912. 1243.8 1871.4 2133.0 1969.5 1994 5259.

Memory latencies in nanoseconds - smaller is better
    (WARNING - may not be correct, check graphs)
------------------------------------------------------------------------------
Host                 OS   Mhz   L1 $   L2 $    Main mem    Rand mem    Guesses
--------- ------------   ---   ----   ----    --------    --------    -------
salvator- Linux 4.6.0-y  1499 2.6690 7.2900        32.9       330.6
```

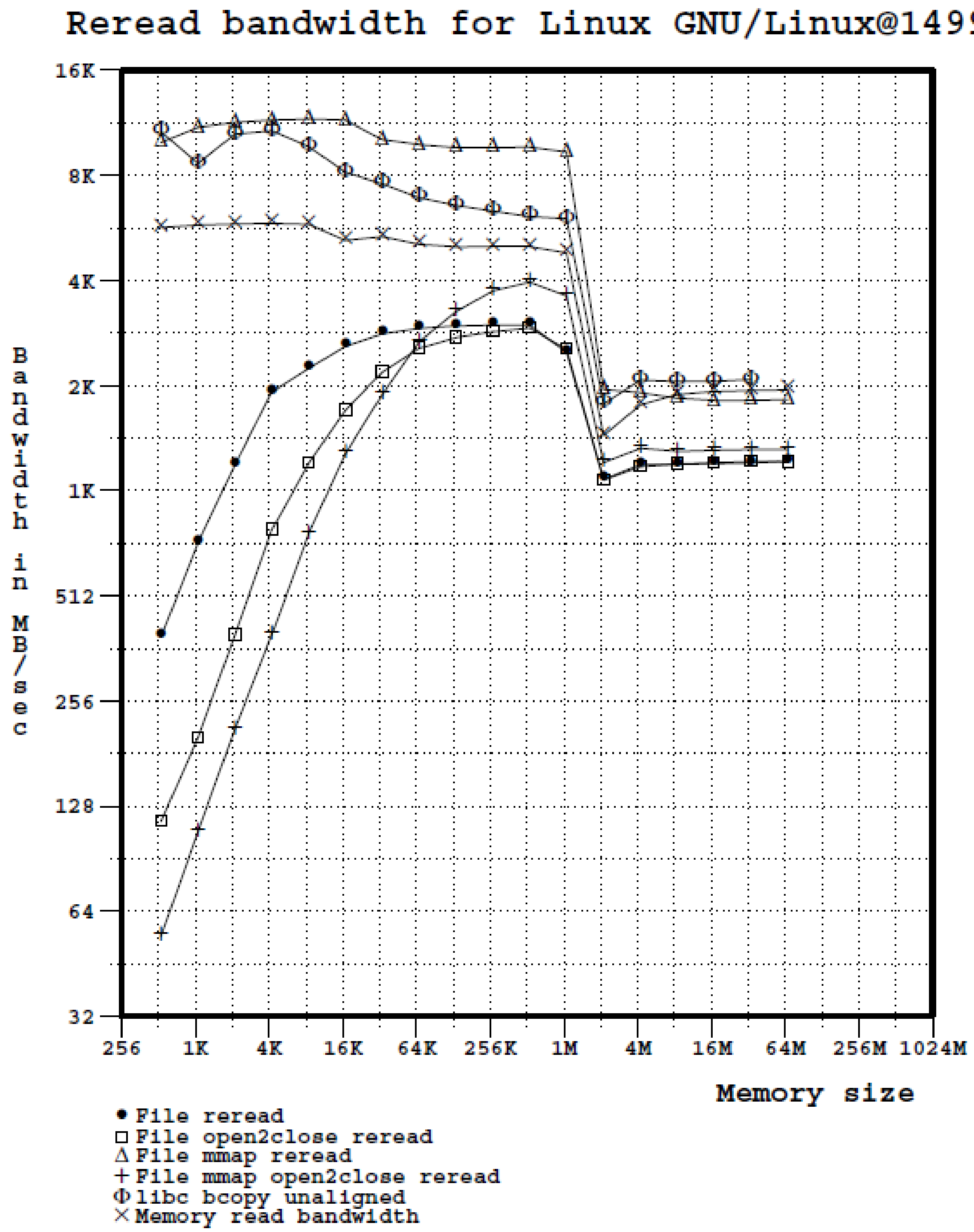
- H3(CA57) Graph(PS.pdf)



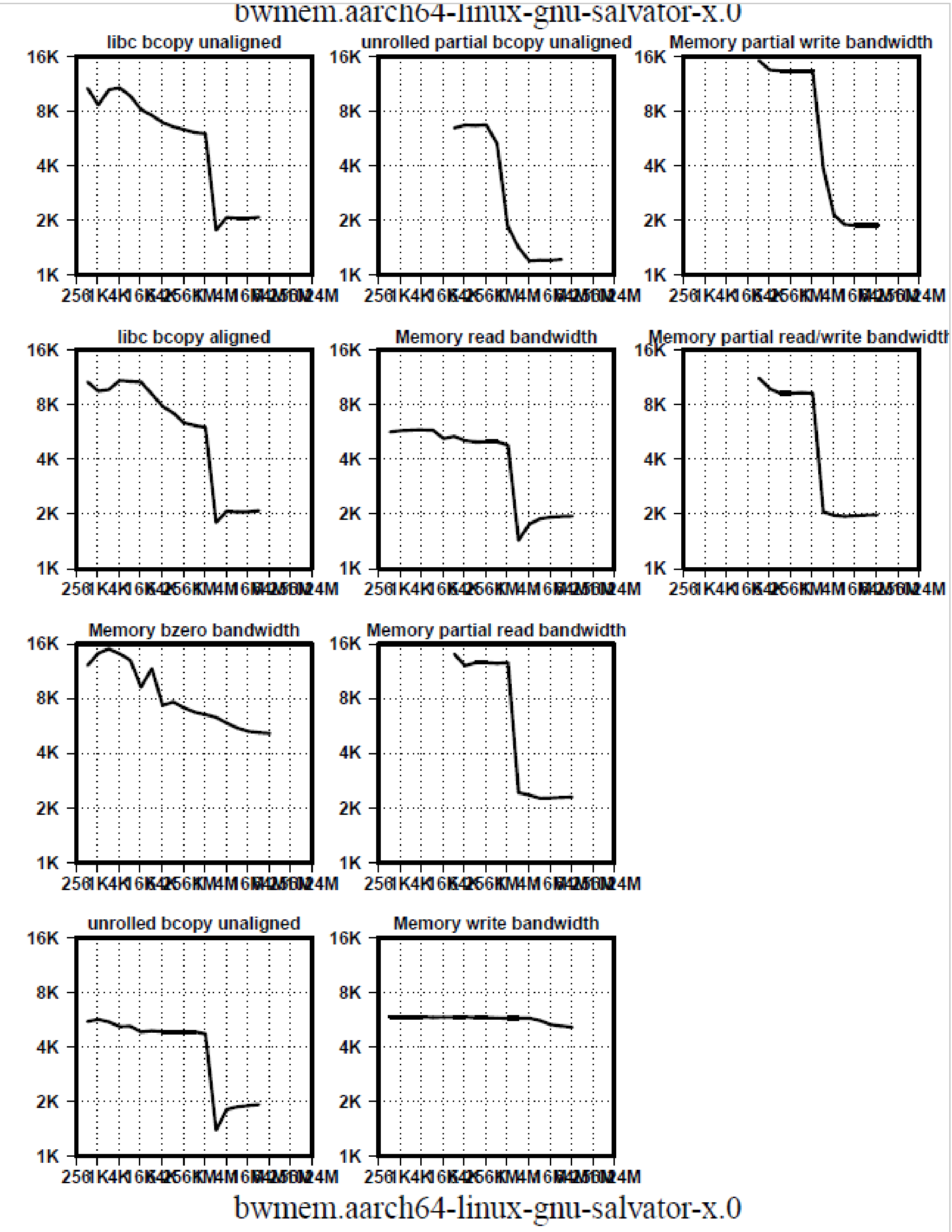
- H3(CA57) Graph(PS1.pdf)



- H3(CA57) Graph(PS2.pdf)



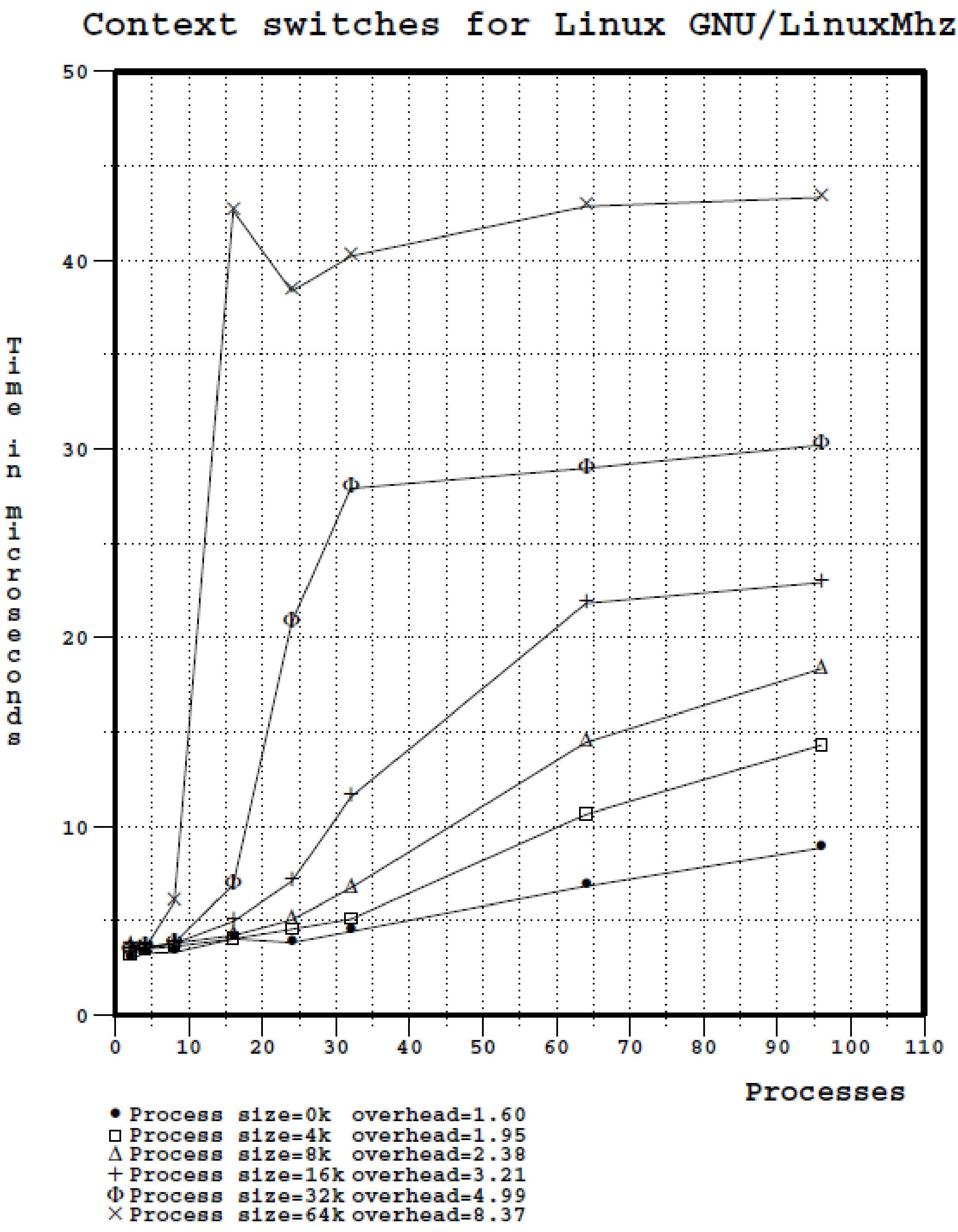
- H3(CA57) Graph(PS3.pdf)



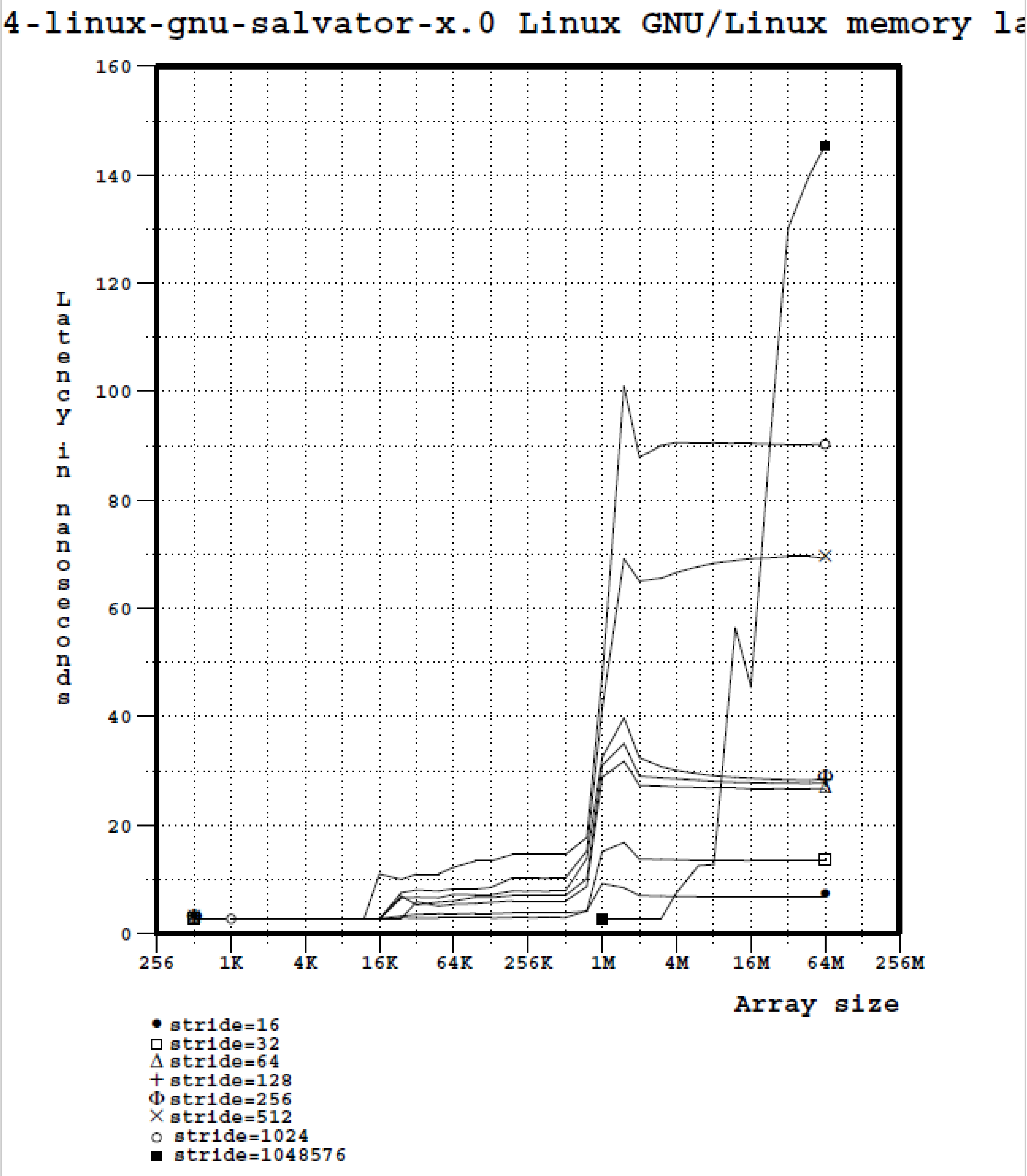
- M3(CA57) Text

```
$ make summary

                 L M B E N C H  3 . 0   S U M M A R Y
                 ------------------------------------
                 (Alpha software, do not distribute)

Basic system parameters
------------------------------------------------------------------------------
Host                 OS Description              Mhz  tlb  cache  mem   scal
                                                     pages line   par   load
                                                           bytes
--------- ------------- ----------------------- ---- ----- ----- ------ ----
salvator- Linux 4.6.0-y       aarch64-linux-gnu 1499    32    64 5.7800    1

Processor, Processes - times in microseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS  Mhz null null      open slct sig  sig  fork exec sh
                             call  I/O stat clos TCP  inst hndl proc proc proc
--------- ------------- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
salvator- Linux 4.6.0-y 1499 0.36 0.46 1.15 2.90 7.08 0.52 2.28 317. 948. 3333

Basic integer operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  intgr intgr  intgr  intgr  intgr
                          bit   add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
salvator- Linux 4.6.0-y 0.6700 0.6700        7.3400 6.0000

Basic uint64 operations - times in nanoseconds - smaller is better
------------------------------------------------------------------
Host                 OS int64  int64  int64  int64  int64
                         bit    add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
salvator- Linux 4.6.0-y  0.670                 10.3 7.3400

Basic float operations - times in nanoseconds - smaller is better
-----------------------------------------------------------------
Host                 OS  float  float  float  float
                         add    mul    div    bogo
--------- ------------- ------ ------ ------ ------
salvator- Linux 4.6.0-y 3.3400 4.0000   12.0 5.3500

Basic double operations - times in nanoseconds - smaller is better
------------------------------------------------------------------
Host                 OS  double double double double
                         add    mul    div    bogo
--------- ------------- ------  ------ ------ ------
salvator- Linux 4.6.0-y 3.3400 4.0000   21.4   20.0

Context switching - times in microseconds - smaller is better
-------------------------------------------------------------------------
Host                 OS  2p/0K 2p/16K 2p/64K 8p/16K 8p/64K 16p/16K 16p/64K
                         ctxsw  ctxsw  ctxsw ctxsw  ctxsw   ctxsw   ctxsw
--------- ------------- ------ ------ ------ ------ ------ ------- -------
salvator- Linux 4.6.0-y 2.9600 3.7000 3.4000 3.7800 5.9800 4.96000    42.6

*Local* Communication latencies in microseconds - smaller is better
---------------------------------------------------------------------
Host                 OS 2p/0K  Pipe AF     UDP  RPC/   TCP  RPC/ TCP
                        ctxsw       UNIX         UDP         TCP conn
--------- ------------- ----- ----- ---- ----- ----- ----- ----- ----
salvator- Linux 4.6.0-y 2.960  10.9 15.4  21.7        28.1        49.

*Remote* Communication latencies in microseconds - smaller is better
---------------------------------------------------------------------
Host                 OS   UDP  RPC/  TCP   RPC/ TCP
                               UDP         TCP  conn
--------- ------------- ----- ----- ----- ----- ----
salvator- Linux 4.6.0-y

File & VM system latencies in microseconds - smaller is better
-------------------------------------------------------------------------------
Host                 OS   0K File      10K File     Mmap    Prot   Page   100fd
                        Create Delete Create Delete Latency Fault  Fault  selct
--------- ------------- ------ ------ ------ ------ ------- ----- ------- -----
salvator- Linux 4.6.0-y                               511.0 0.288 0.73270 2.525

*Local* Communication bandwidths in MB/s - bigger is better
-----------------------------------------------------------------------------
Host                OS  Pipe AF    TCP  File   Mmap  Bcopy  Bcopy  Mem   Mem
                             UNIX      reread reread (libc) (hand) read write
--------- ------------- ---- ---- ---- ------ ------ ------ ------ ---- -----
salvator- Linux 4.6.0-y 1087 2636 870. 1282.7 2149.3 2376.2 2177.4 2383 5237.

Memory latencies in nanoseconds - smaller is better
    (WARNING - may not be correct, check graphs)
------------------------------------------------------------------------------
Host                 OS   Mhz   L1 $   L2 $    Main mem    Rand mem    Guesses
--------- -------------   ---   ----   ----    --------    --------    -------
salvator- Linux 4.6.0-y  1499 2.6690 6.6390        27.8       274.7
```

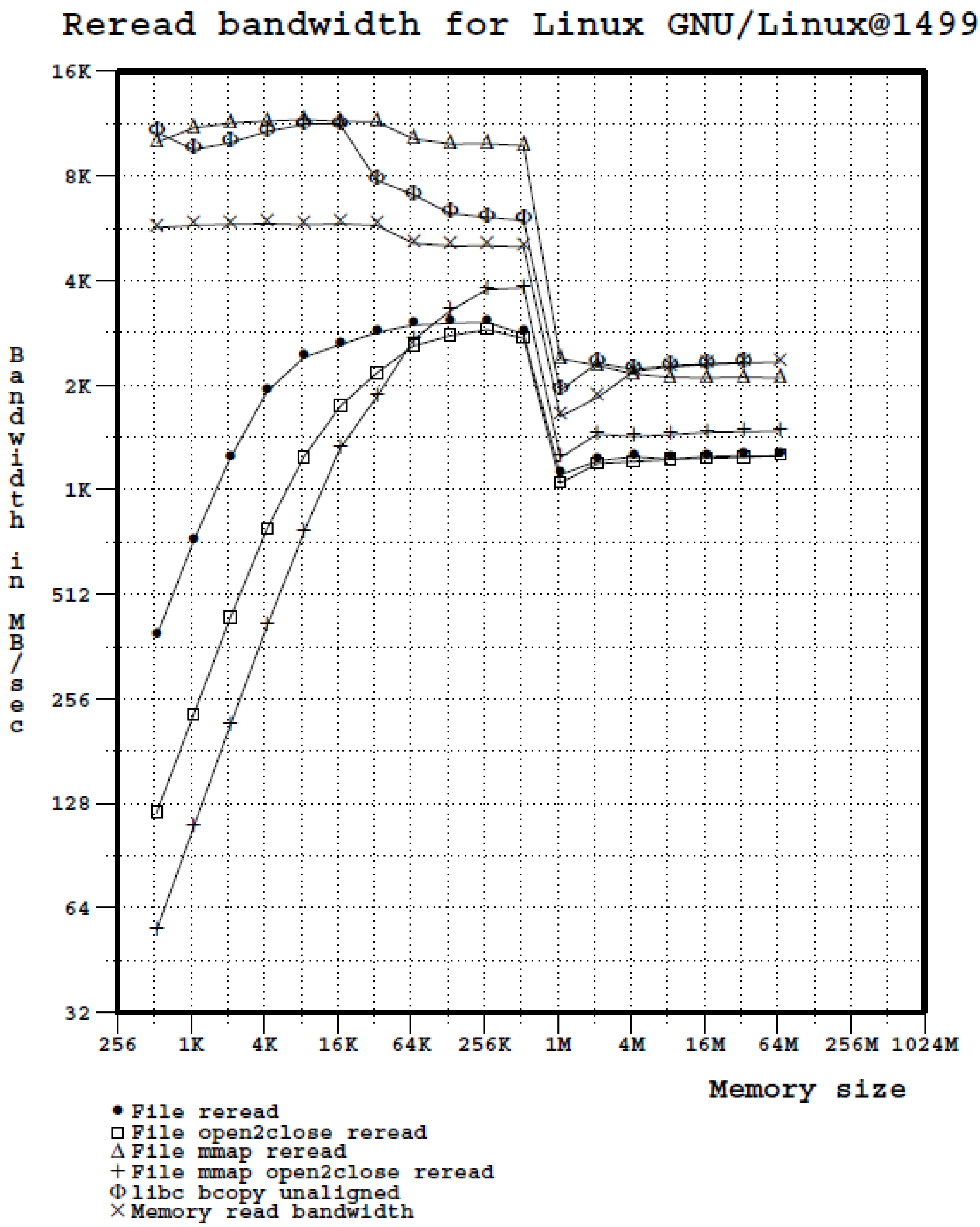
- M3(CA57) Graph(PS.pdf)



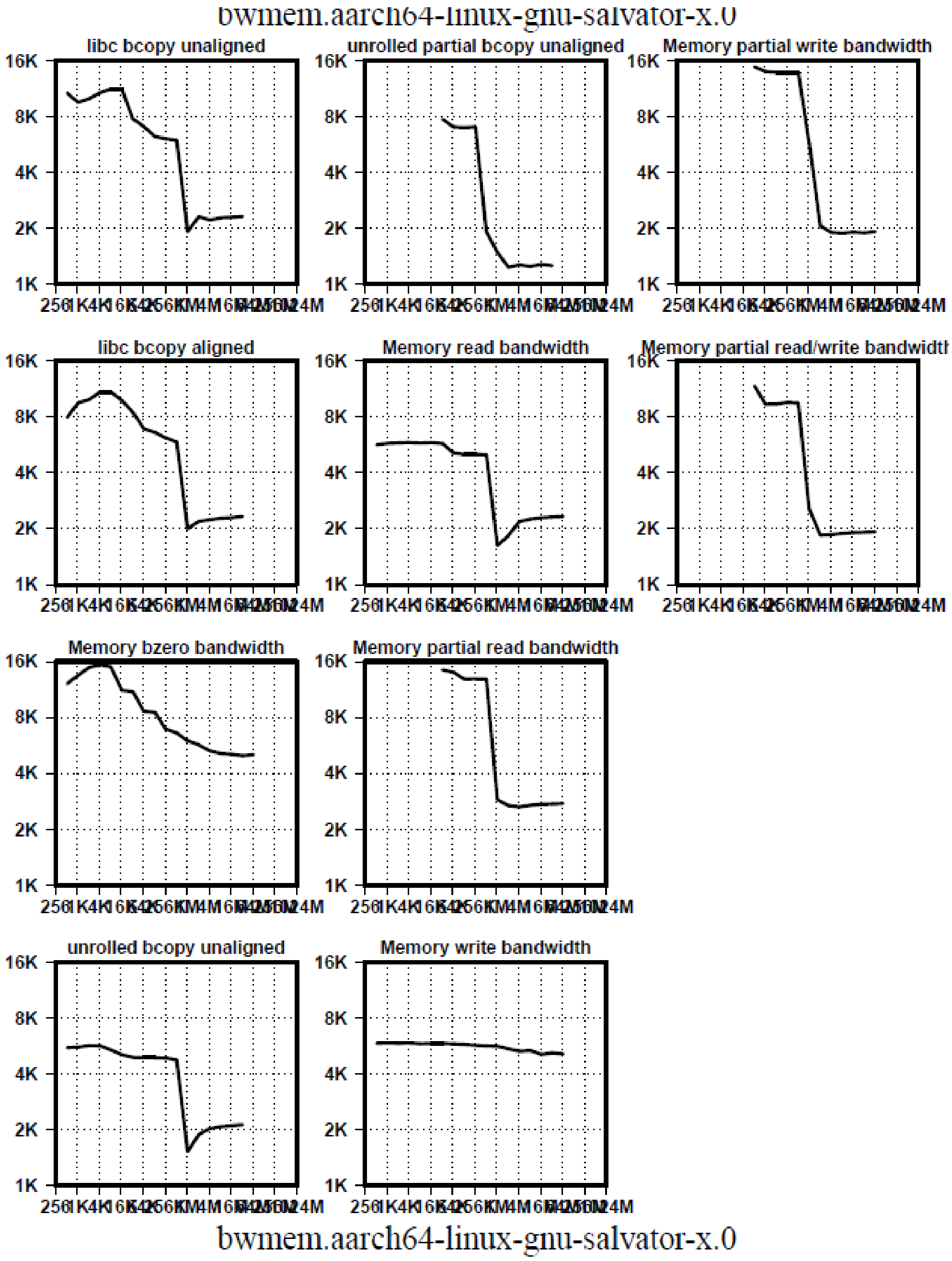
- M3(CA57) Graph(PS1.pdf)



- M3(CA57) Graph(PS2.pdf)



- M3(CA57) Graph(PS3.pdf)



4.3 pmbw (Parallel Memory Bandwidth Benchmark)

The tool pmbw is a set of assembler routines to measure the parallel memory (cache and RAM) bandwidth of modern multi-core machines. Memory bandwidth is one of the key performance factors of any computer system. And today, measuring the memory performance often gives a more realistic view on the overall speed of a machine than pure arithmetic or floating-point benchmarks. This is due to the speed of computation units in modern CPUs growing faster than the memory bandwidth, which however is required to get more information to the CPU. The bigger the processed data amount gets, the more important memory bandwidth becomes!
The pmbw tool contains a set of very basic functions, which are all hand-coded in assembler to avoid any compiler optimizations. These basic functions are modeled after the basic inner loops found in any data processing: sequential scanning and pure random access. Any application will have a memory access pattern that is somewhere between these two extremes.

Besides these two access patterns, the basic functions benchmark different modes of memory access. Depending on the architecture, 16- / 32- / 64- / 128- or 256-bit memory transfers are tested by using different machine instructions, like MMX, SSE or AVX. Furthermore, iterating by pointers is compared against access via array index. The current version of pmbw supports benchmarking x86_32-bit, x86_64-bit and ARMv6 systems.

Most important feature of this benchmark is that it will perform the tests in parallel with growing number of threads. The results of these scalability tests highlight the basic problem which parallel multi-core algorithms must cope with: scanning memory bandwidth does not scale with the number of cores in current systems. The ratio of bandwidth to cache over the bandwidth to RAM determines the amount of local cache-based processing which must be done between RAM accesses for an algorithm to scale well.

4.3.1 Evaluation Procedure

(1) Download and extract pmbw package

- Download the package at the below link:
  https://github.com/bingmann/pmbw/archive/41c497fabf79ffc626fdf258ecd145793c20b849.zip
- Extract the package by Linux command

```
$ unzip 41c497fabf79ffc626fdf258ecd145793c20b849.zip
$ mv pmbw-41c497fabf79ffc626fdf258ecd145793c20b849 pmbw
```

- Or:

```
$ git clone https://github.com/bingmann/pmbw.git
$ cd pmbw/
$ git checkout -b aarch64-tmp 41c497fabf79ffc626fdf258ecd145793c20b849
```

(2) Configure building process

- Replace <toolchain_path> with the path string where the cross toolchain is installed.
  For details, refer to **2.2.1 Path of toolchain**.

```
$ source <toolchain_path>/environment-setup-aarch64-poky-linux
$ export LDFLAGS=""
$ cd pmbw
$ ./configure --host=aarch64-poky-linux
```

(3) Optimization Options

- pmbw/Makefile

```
CXXFLAGS = -O2 -pipe -g -feliminate-unused-debug-types
LDFLAGS = -Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed
```

(4) Compile the package

```
$ make
```

(5) Copy the entire pmbw folder to target file system

```
$ cp -r ../pmbw <rootfs>/home/root/
```

(6) Check build date

- Boot the system and check the build date of the environment from the serial log.
  Please refer to **2.3 Checking build date** for the check points.

(7) Execute the benchmark on target

```
# cd /home/root/pmbw
# nice -n -2 ./pmbw -S 0
```

➢ The option of pmbw is described as below:

```
$ ./pmbw -h
Usage: ./pmbw [options]
Options:
  -f <match>     Run only benchmarks containing this substring, can be used multile times. Try "list".
  -M <size>      Limit the maximum amount of memory allocated at startup [byte].
  -p <nthrs>     Run benchmarks with at least this thread count.
  -P <nthrs>     Run benchmarks with at most this thread count (overrides detected processor count).
  -Q             Run benchmarks with quadratically increasing thread count.
  -s <size>      Limit the _minimum_ test array size [byte]. Set to 0 for no limit.
  -S <size>      Limit the _maximum_ test array size [byte]. Set to 0 for no limit.
```

In this instruction command “nice -n -2 ./pmbw -S 0” mean the niceness value is -2 (making a new process less nice, increasing CPU priority) and run pmbw with no limit the maximum test array size.

For more detail about command “nice”, pleaase refer site: http://bencane.com/2013/09/09/setting-process-cpu-priority-with-nice-and-renice/

4.3.2 Evaluation Result

4.3.2.1 Overview

Running pmbw will immediately start the benchmark. All statistical output of pmbw is written to the file stats.txt in the current directory. To visualize the statistical output, the stats2gnuplot program can create a PDF via gnuplot. The output PDF can be generated by calling:

```
# ./stats2gnuplot stats.txt > test.txt
```

```
$ cat test.txt | gnuplot
```

* gnuplot needs to be executed on the Host PC (gnuplot does not exist on the target file system).
* There are 60 graphs generated for each platform. Only some important ones are shown here.

The names of the benchmark routines is composed of several abbreviated components, which together specify the exact operations:

- Scan indicates scanning operations, while Perm are permutation walking tests.
- Write/Read specifies the operation done.
- 16/32/64/128/256 indicates the number of bits transfered by a single instruction in the benchmark routine.

Which exactly are available depends on the architecture.

- Ptr represents pointer-based iteration, while Index is index-based array access.
- SimpleLoop routines contain only one operation per loop, after which the end condition is checked.

UnrollLoop benchmarks contain 16 operations per loop, followed by the end check.

So that:

"ScanWrite64PtrUnrollLoop" is a benchmark routing with scanning pointer-based access pattern, performing 16 operations per loop, and writing 64-bit per instruction. This benchmark routine comes closest to what memset() would do on a 64-bit machine.

"ScaRead64PtrUnrollLoop" is a benchmark routing with scanning pointer-based access pattern, performing 16 operations per loop, and reading 64-bit per instruction. This benchmark routine comes closest to what memset() would do on a 64-bit machine.

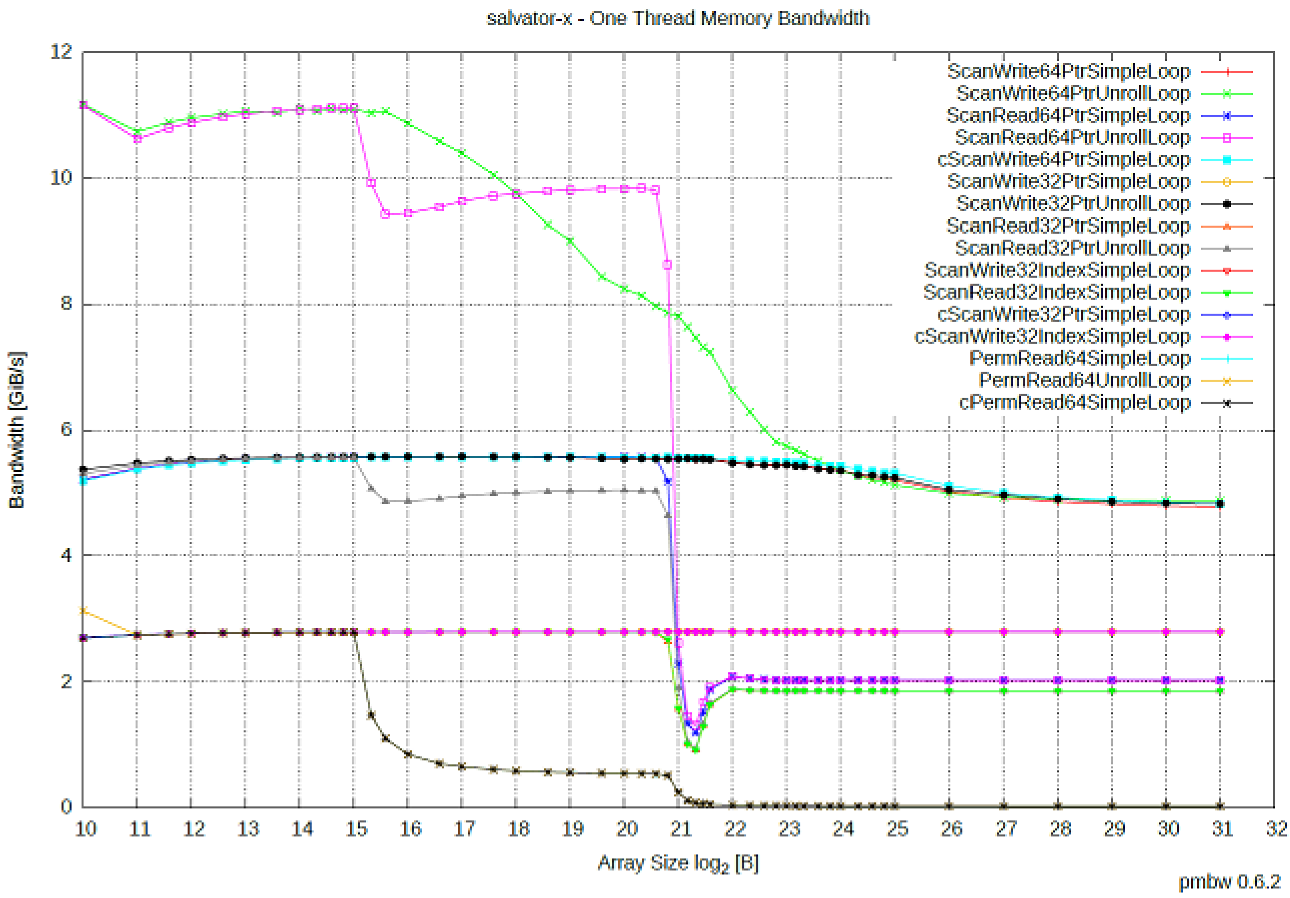
"PermRead64UnrollLoop" is a benchmark tests walk a random pointer permutation (each array cell accessed in the array yields the position of the next access), performing 16 operations per loop, and writing 64-bit per instruction.

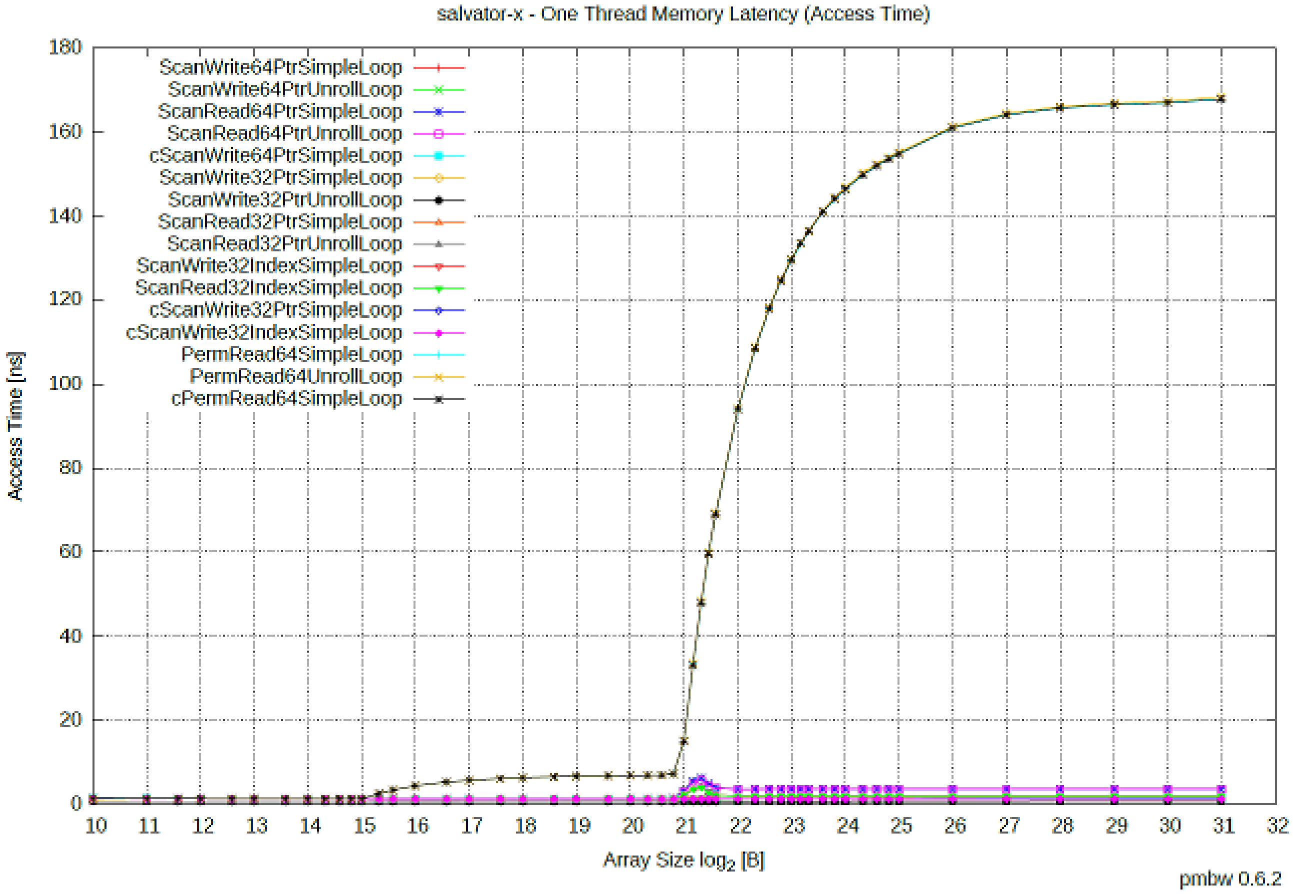
"nthreads=1" the number of thread when cpu run.

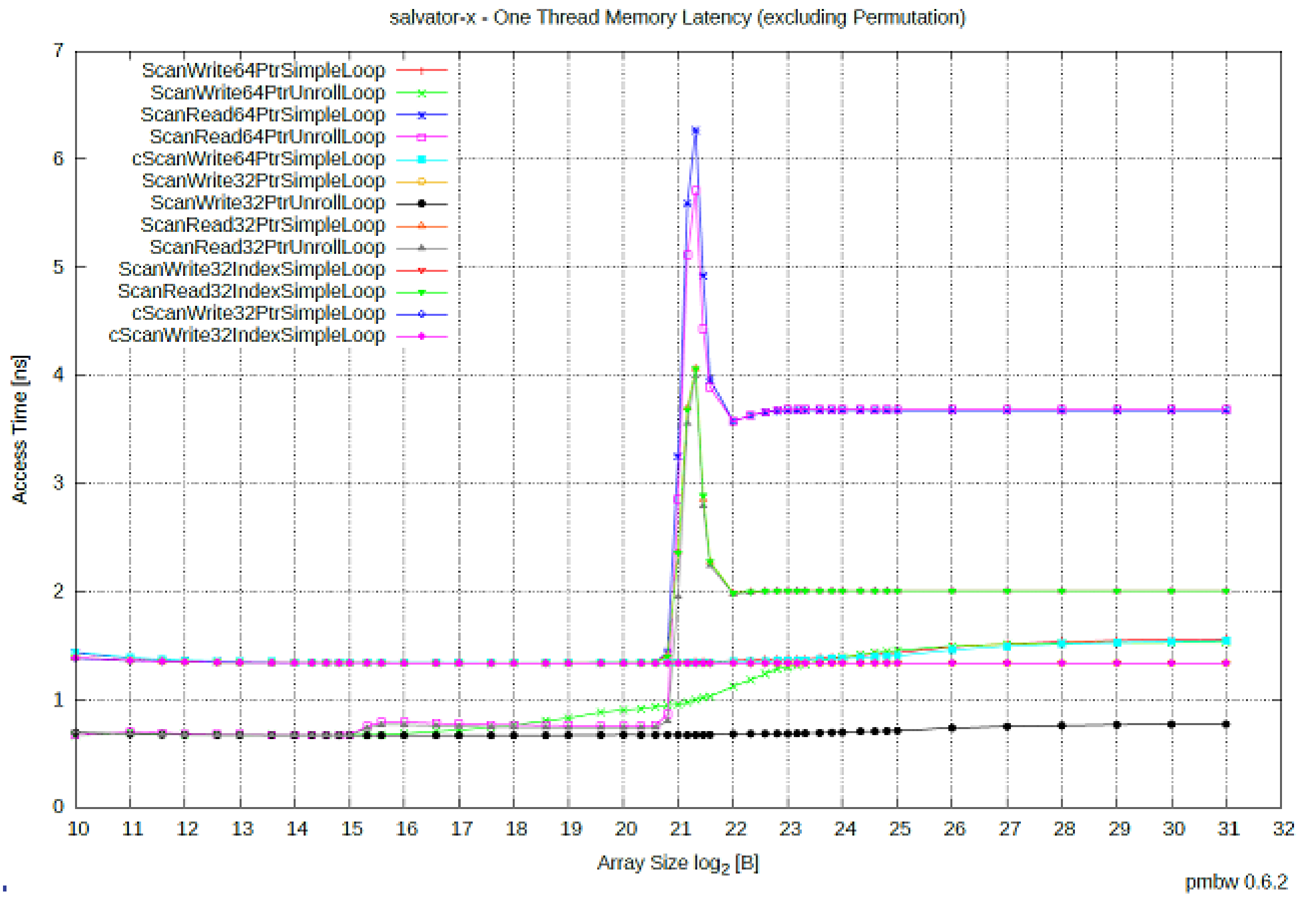
For more detail refer site: https://panthema.net/2013/pmbw/

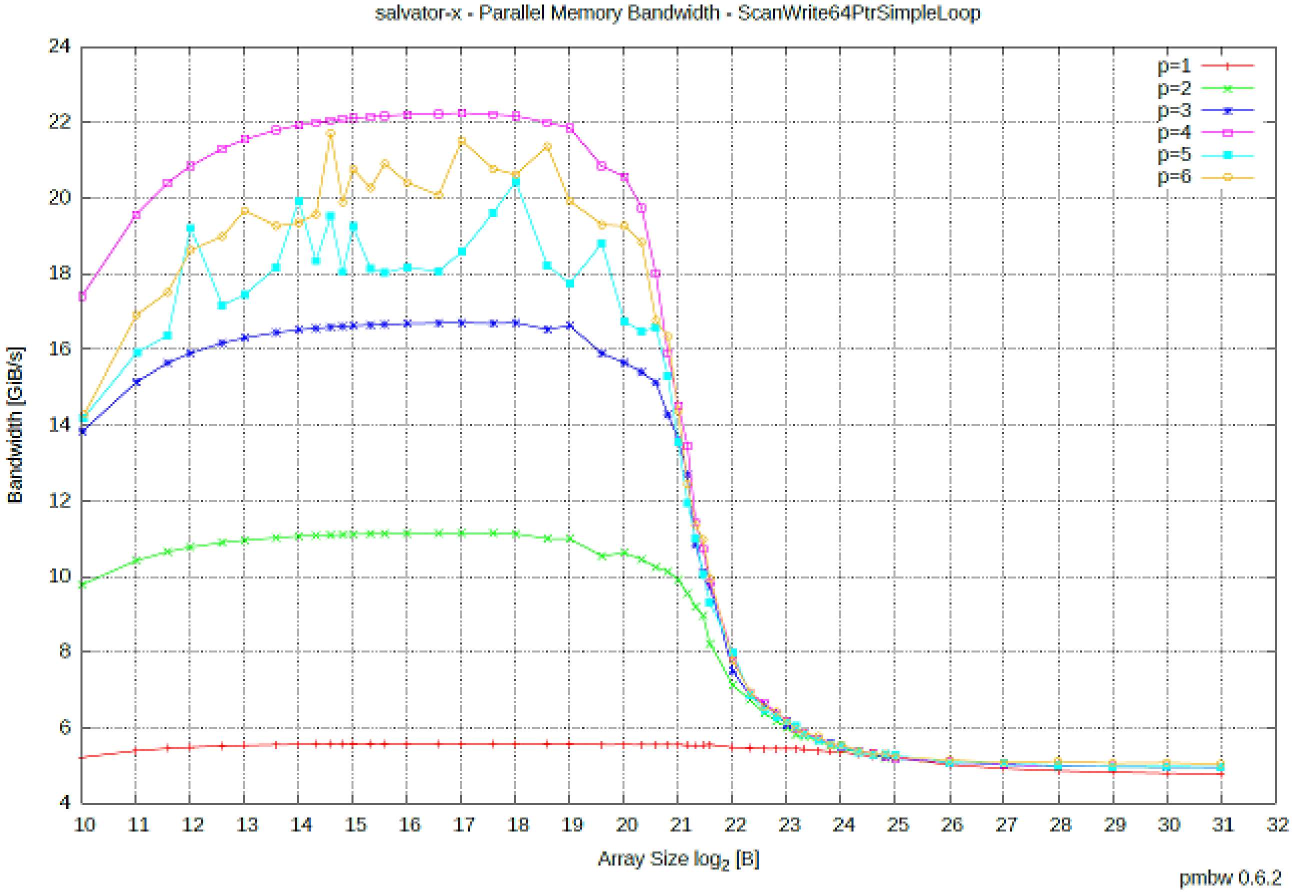
4.3.2.2 H3

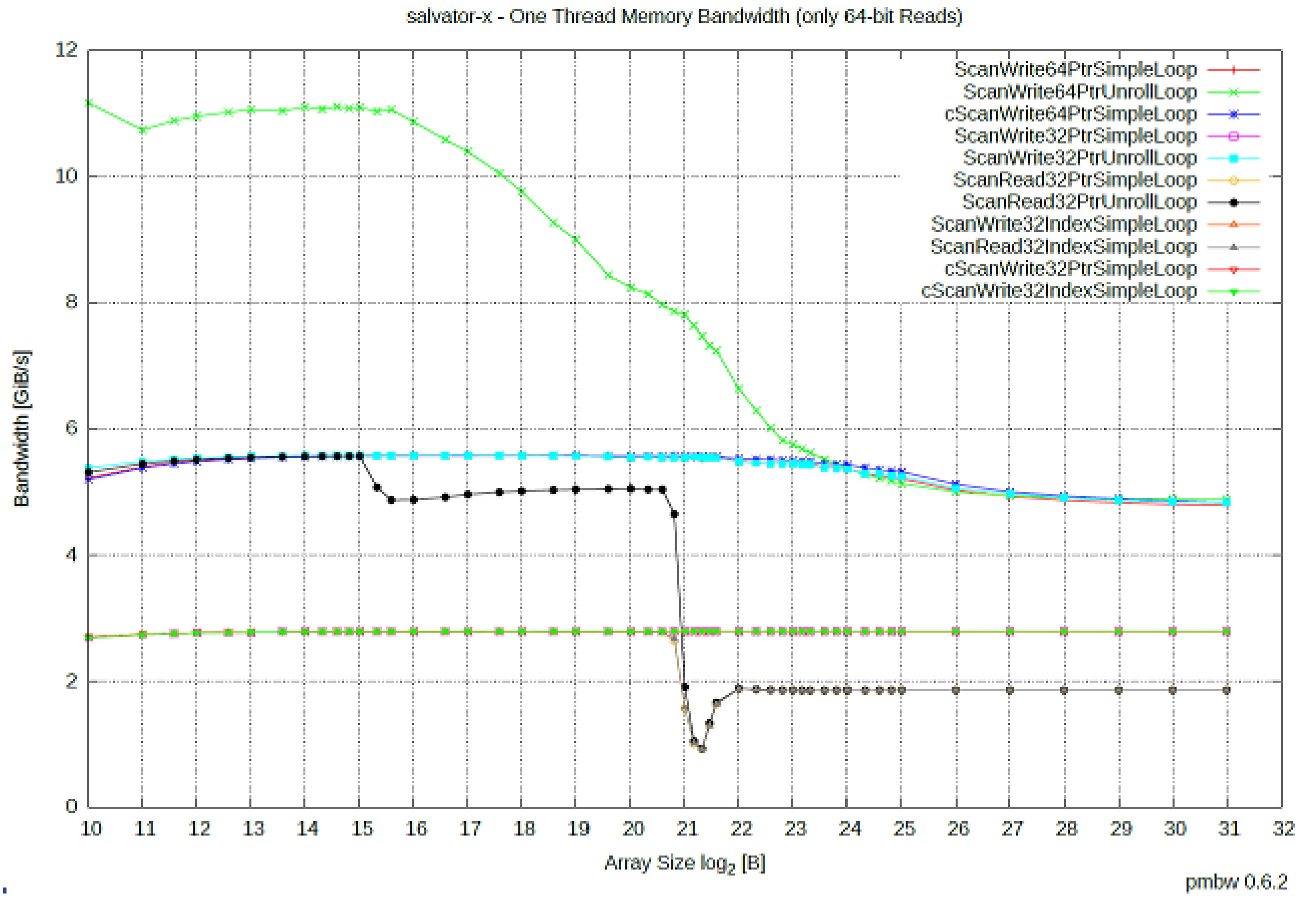
- CA57

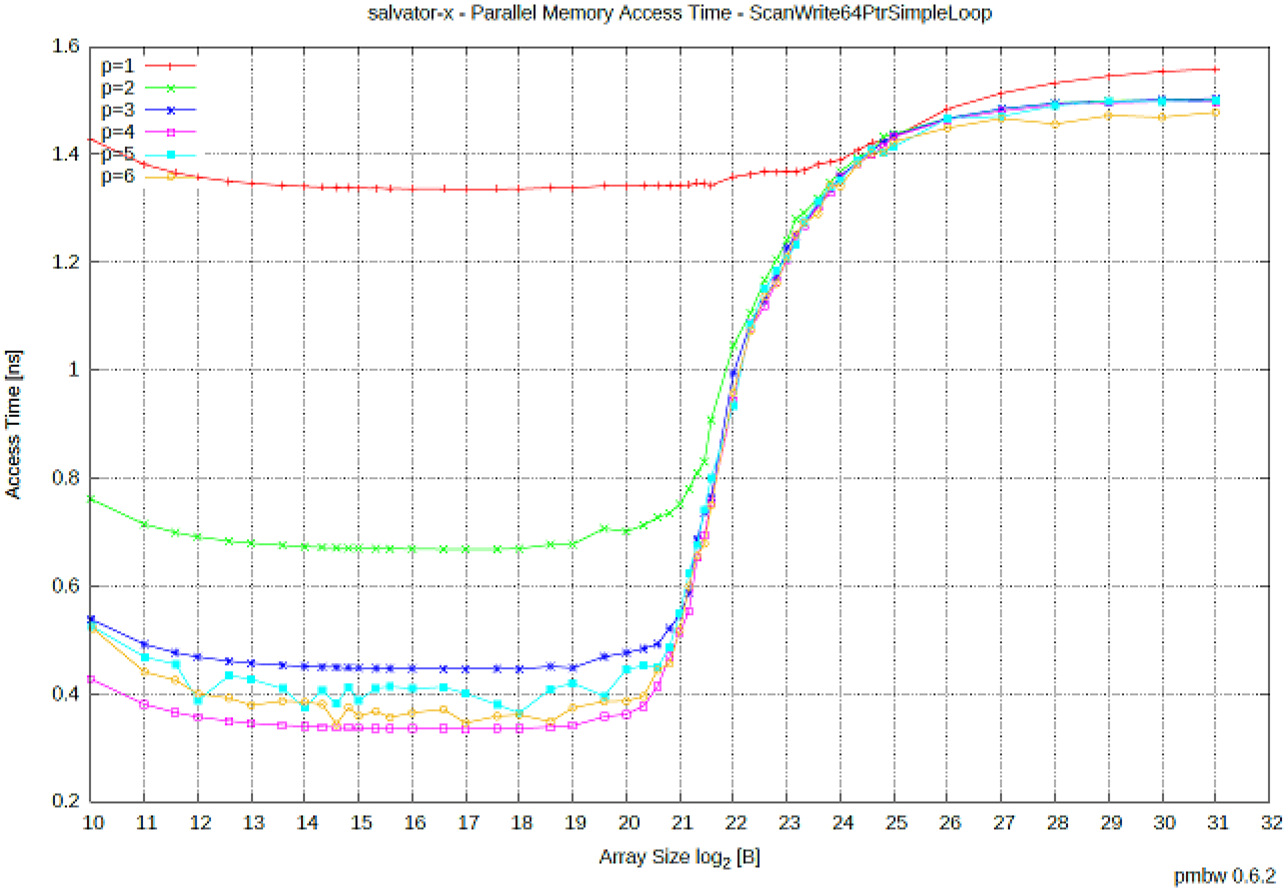

salvator-x - One Thread Memory Bandwidth
Bandwidth [GiB/s]
Array Size log2 [B]
ScanWrite64PtrSimpleLoop
ScanWrite64PtrUnrollLoop
ScanRead64PtrSimpleLoop
ScanRead64PtrUnrollLoop
cScanWrite64PtrSimpleLoop
ScanWrite32PtrSimpleLoop
ScanWrite32PtrUnrollLoop
ScanRead32PtrSimpleLoop
ScanRead32PtrUnrollLoop
ScanWrite32IndexSimpleLoop
ScanRead32IndexSimpleLoop
cScanWrite32PtrSimpleLoop
cScanWrite32IndexSimpleLoop
PermRead64SimpleLoop
PermRead64UnrollLoop
cPermRead64SimpleLoop
pmbw 0.6.2


salvator-x - One Thread Memory Latency (Access Time)
ScanWrite64PtrSimpleLoop
ScanWrite64PtrUnrollLoop
ScanRead64PtrSimpleLoop
ScanRead64PtrUnrollLoop
cScanWrite64PtrSimpleLoop
ScanWrite32PtrSimpleLoop
ScanWrite32PtrUnrollLoop
ScanRead32PtrSimpleLoop
ScanRead32PtrUnrollLoop
ScanWrite32IndexSimpleLoop
ScanRead32IndexSimpleLoop
cScanWrite32PtrSimpleLoop
cScanWrite32IndexSimpleLoop
PermRead64SimpleLoop
PermRead64UnrollLoop
cPermRead64SimpleLoop
Access Time [ns]
Array Size $\log_2$ [B]
pmbw 0.6.2


salvator-x - One Thread Memory Latency (excluding Permutation)
ScanWrite64PtrSimpleLoop
ScanWrite64PtrUnrollLoop
ScanRead64PtrSimpleLoop
ScanRead64PtrUnrollLoop
cScanWrite64PtrSimpleLoop
ScanWrite32PtrSimpleLoop
ScanWrite32PtrUnrollLoop
ScanRead32PtrSimpleLoop
ScanRead32PtrUnrollLoop
ScanWrite32IndexSimpleLoop
ScanRead32IndexSimpleLoop
cScanWrite32PtrSimpleLoop
cScanWrite32IndexSimpleLoop
Access Time [ns]
Array Size $\log_2$ [B]
pmbw 0.6.2

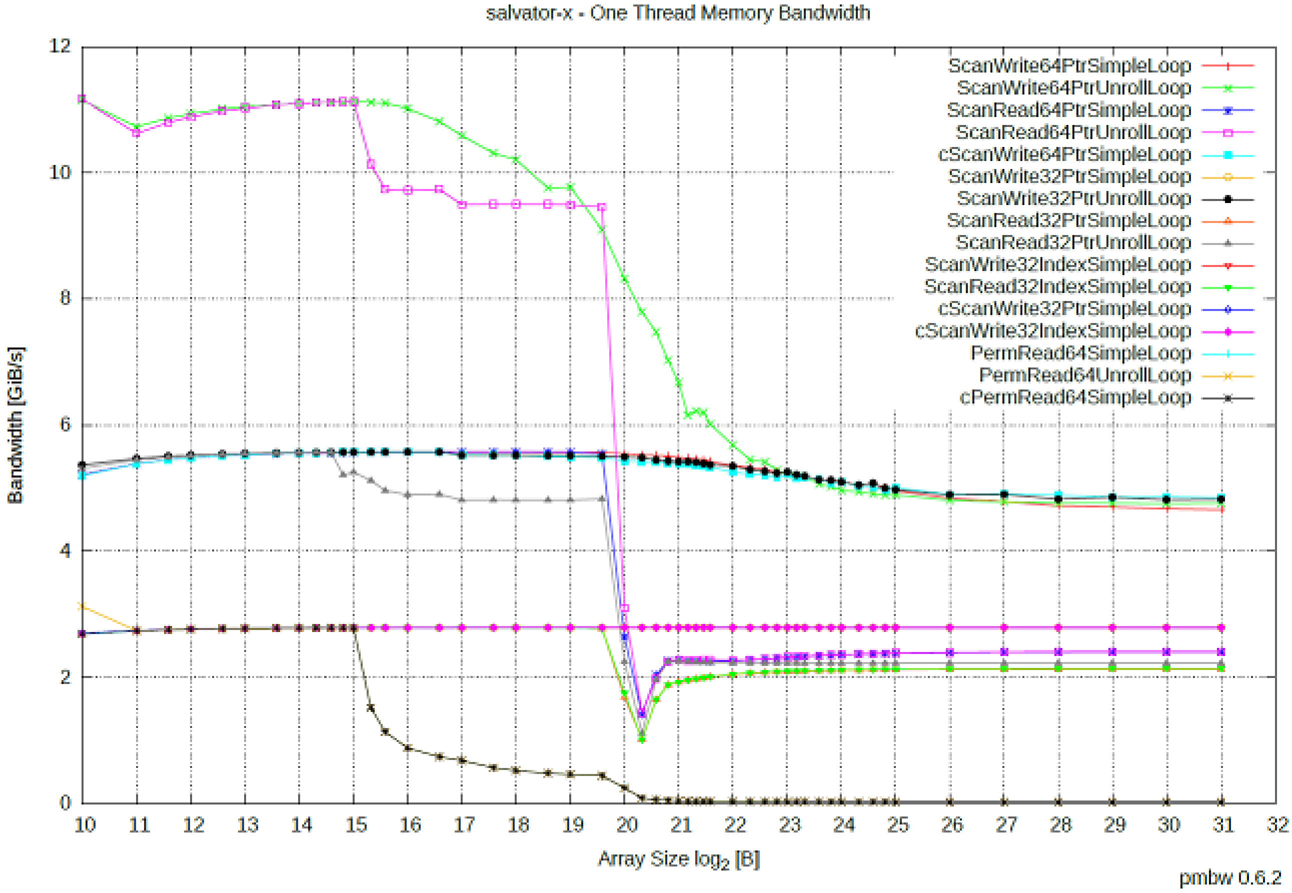

salvator-x - Parallel Memory Bandwidth - ScanWrite64PtrSimpleLoop
p=1
p=2
p=3
p=4
p=5
p=6
Bandwidth [GiB/s]
Array Size log2 [B]
pmbw 0.6.2

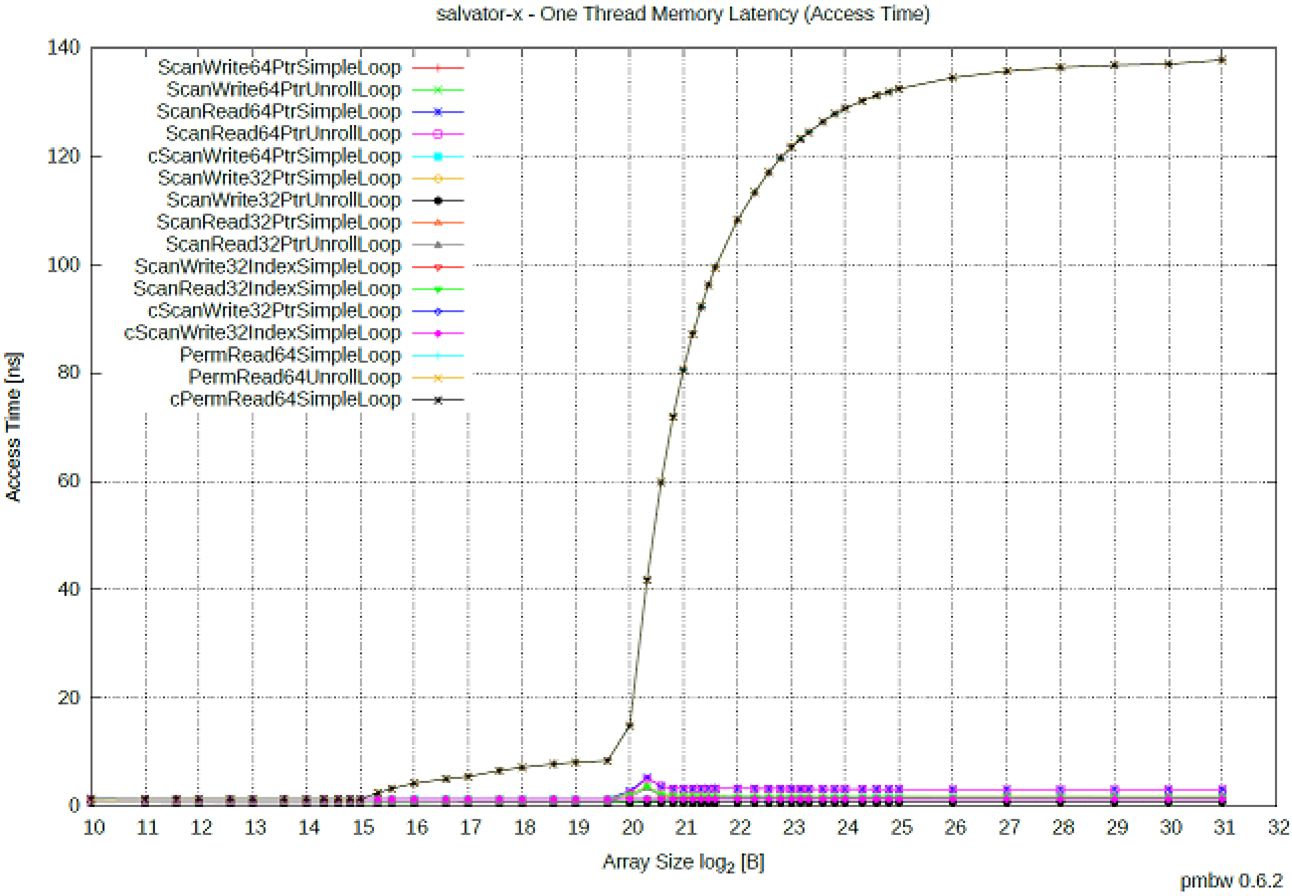

salvator-x - One Thread Memory Bandwidth (only 64-bit Reads)
ScanWrite64PtrSimpleLoop
ScanWrite64PtrUnrollLoop
cScanWrite64PtrSimpleLoop
ScanWrite32PtrSimpleLoop
ScanWrite32PtrUnrollLoop
ScanRead32PtrSimpleLoop
ScanRead32PtrUnrollLoop
ScanWrite32IndexSimpleLoop
ScanRead32IndexSimpleLoop
cScanWrite32PtrSimpleLoop
cScanWrite32IndexSimpleLoop
Bandwidth [GiB/s]
Array Size log2 [B]
pmbw 0.6.2

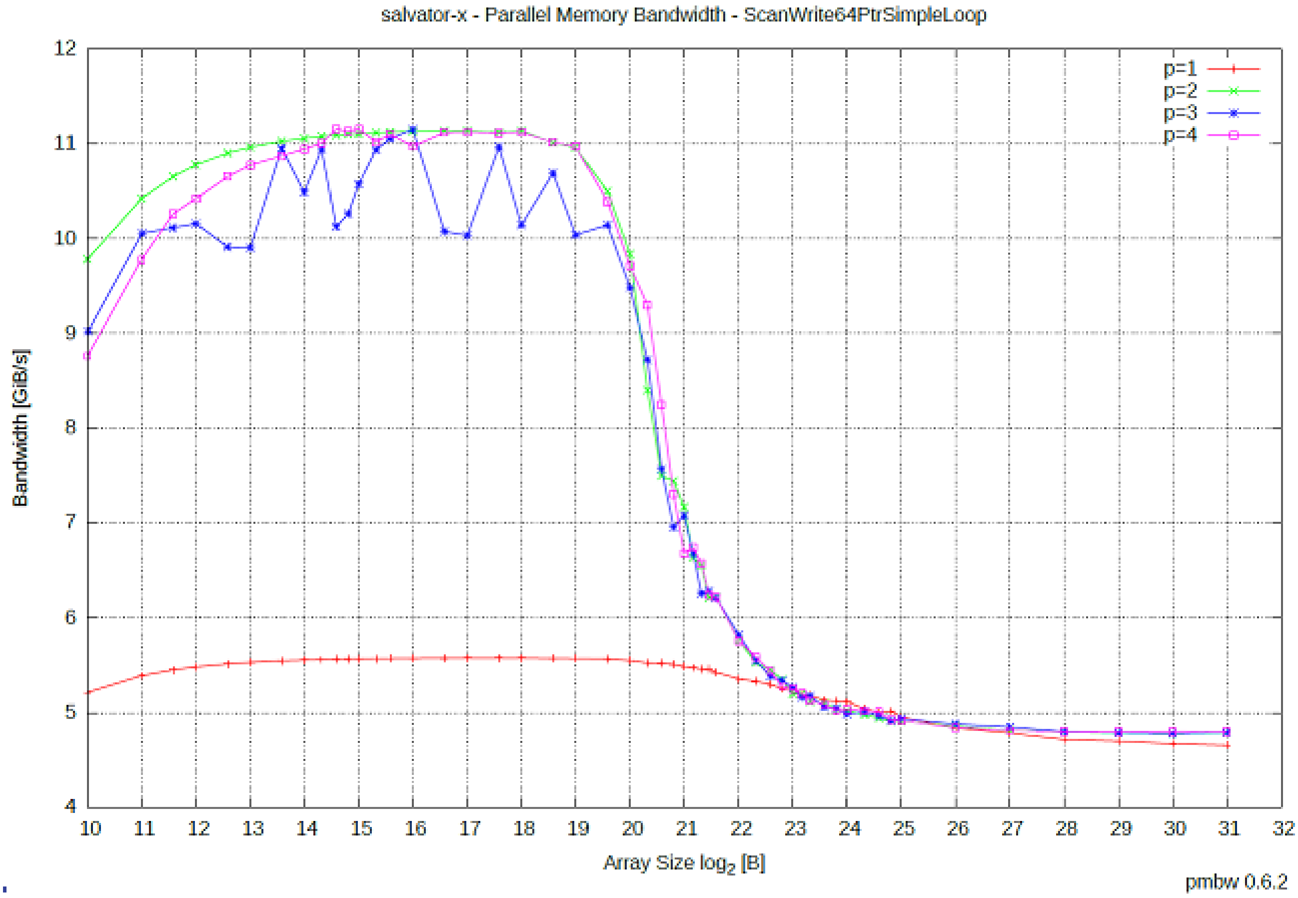

salvator-x - Parallel Memory Access Time - ScanWrite64PtrSimpleLoop
p=1
p=2
p=3
p=4
p=5
p=6
Access Time [ns]
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Array Size log_2 [B]
pmbw 0.6.2

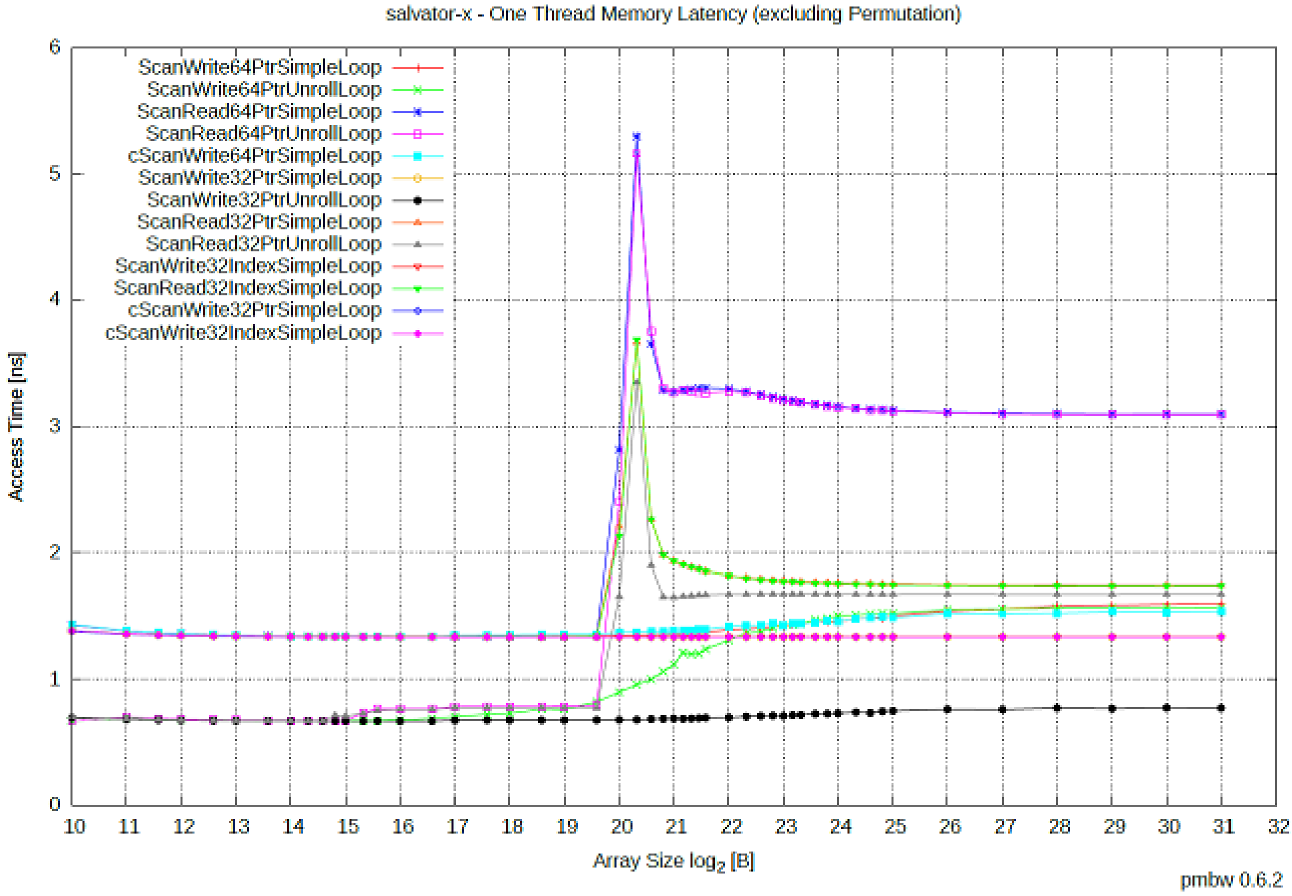
4.3.2.3 M3

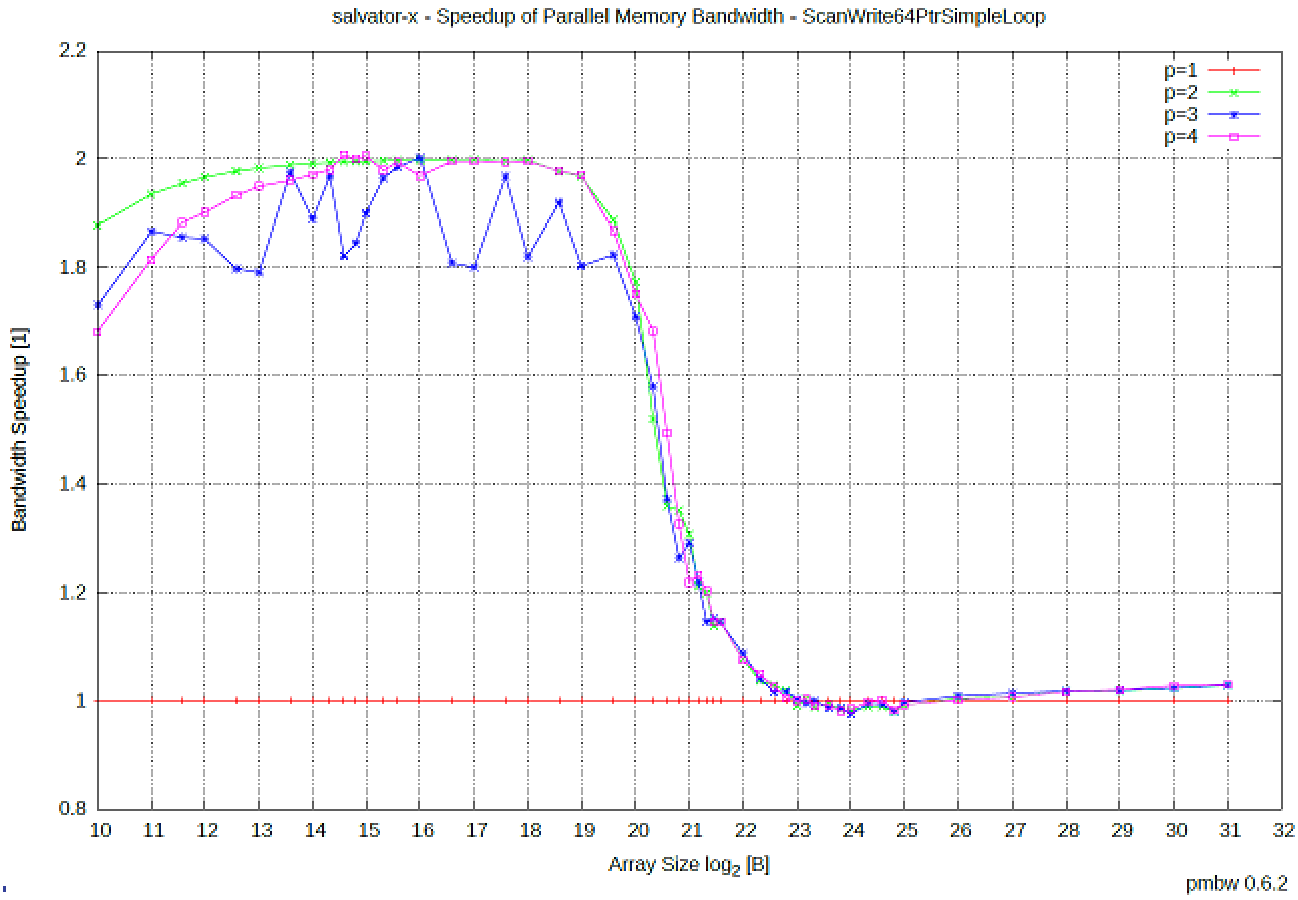
- CA57

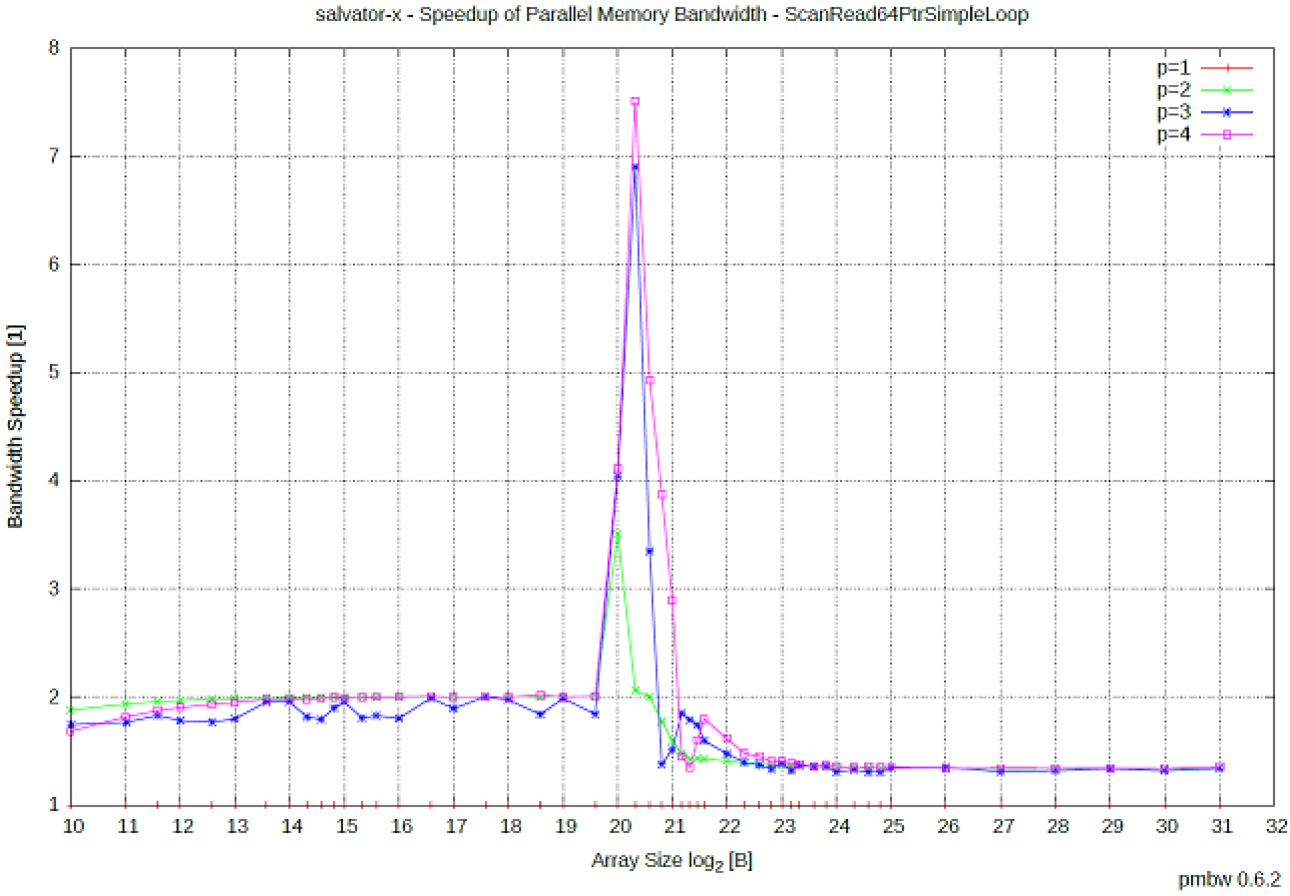

salvator-x - One Thread Memory Bandwidth
ScanWrite64PtrSimpleLoop
ScanWrite64PtrUnrollLoop
ScanRead64PtrSimpleLoop
ScanRead64PtrUnrollLoop
cScanWrite64PtrSimpleLoop
ScanWrite32PtrSimpleLoop
ScanWrite32PtrUnrollLoop
ScanRead32PtrSimpleLoop
ScanRead32PtrUnrollLoop
ScanWrite32IndexSimpleLoop
ScanRead32IndexSimpleLoop
cScanWrite32PtrSimpleLoop
cScanWrite32IndexSimpleLoop
PermRead64SimpleLoop
PermRead64UnrollLoop
cPermRead64SimpleLoop
Bandwidth [GiB/s]
0
2
4
6
8
10
12
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
Array Size log2 [B]
pmbw 0.6.2


salvator-x - One Thread Memory Latency (Access Time)
Access Time [ns]
Array Size log2 [B]
ScanWrite64PtrSimpleLoop
ScanWrite64PtrUnrollLoop
ScanRead64PtrSimpleLoop
ScanRead64PtrUnrollLoop
cScanWrite64PtrSimpleLoop
ScanWrite32PtrSimpleLoop
ScanWrite32PtrUnrollLoop
ScanRead32PtrSimpleLoop
ScanRead32PtrUnrollLoop
ScanWrite32IndexSimpleLoop
ScanRead32IndexSimpleLoop
cScanWrite32PtrSimpleLoop
cScanWrite32IndexSimpleLoop
PermRead64SimpleLoop
PermRead64UnrollLoop
cPermRead64SimpleLoop
pmbw 0.6.2


salvator-x - Parallel Memory Bandwidth - ScanWrite64PtrSimpleLoop
Bandwidth [GiB/s]
Array Size log2 [B]
p=1
p=2
p=3
p=4
pmbw 0.6.2


salvator-x - One Thread Memory Latency (excluding Permutation)
ScanWrite64PtrSimpleLoop
ScanWrite64PtrUnrollLoop
ScanRead64PtrSimpleLoop
ScanRead64PtrUnrollLoop
cScanWrite64PtrSimpleLoop
ScanWrite32PtrSimpleLoop
ScanWrite32PtrUnrollLoop
ScanRead32PtrSimpleLoop
ScanRead32PtrUnrollLoop
ScanWrite32IndexSimpleLoop
ScanRead32IndexSimpleLoop
cScanWrite32PtrSimpleLoop
cScanWrite32IndexSimpleLoop
Access Time [ns]
Array Size $\log_2$ [B]
pmbw 0.6.2


salvator-x - Speedup of Parallel Memory Bandwidth - ScanWrite64PtrSimpleLoop
p=1
p=2
p=3
p=4
Bandwidth Speedup [1]
Array Size $\log_2$ [B]
pmbw 0.6.2


salvator-x - Speedup of Parallel Memory Bandwidth - ScanRead64PtrSimpleLoop
p=1
p=2
p=3
p=4
Bandwidth Speedup [1]
Array Size log2 [B]
pmbw 0.6.2

5. System Performance Evaluation

This chapter describes procedures and results for evaluating system performance.

5.1 UnixBench

The main purpose of this tool is to provide a basic indicator of the performance of a Unix-like operating system. UnixBench can be used to evaluate the performance of your system when running a single or multiple tasks. Please be mindful that this is a system benchmarking tool, not just a CPU, RAM or disk benchmark tool. The results will depend not only on your hardware, but also on your operating system, libraries, and even compiler.

5.1.1 Evaluation Procedure

(1) Download and extract unixbench package

- Download the package at the below link:
  https://github.com/kdlucas/byte-unixbench/archive/v5.1.3.tar.gz
- Extract the source
  * Depending on the command used for downloading, the saved filename may change.
  In that case, replace the filename specified for tar command.
  Example: tar xf byte-unixbench-5.1.3.tar.gz

```
$ wget https://github.com/kdlucas/byte-unixbench/archive/v5.1.3.tar.gz
$ tar xf v5.1.3.tar.gz
$ ls
byte-unixbench-5.1.3
```

(2) Applying patches

- Save the contents of the following text-box as a patch file, and apply the patch to the source package.
- For M3, change the **red boldface part of the light blue background** of the following patch contents from 4 to 2.

```
diff -uprN a/UnixBench/Makefile b/UnixBench/Makefile
--- a/UnixBench/Makefile       2015-06-05 02:20:18.000000000 +0900
+++ b/UnixBench/Makefile       2017-03-22 15:57:55.535499660 +0900
@@ -53,7 +53,7 @@ GL_LIBS = -lGL -lXext -lX11
 # COMPILER CONFIGURATION: Set "CC" to the name of the compiler to use
 # to build the binary benchmarks.  You should also set "$cCompiler" in the
 # Run script to the name of the compiler you want to test.
-CC=gcc
+#CC=gcc

 # OPTIMISATION SETTINGS:

@@ -71,7 +71,8 @@ CC=gcc
 #         -m386 -malign-loops=1 -malign-jumps=1 -malign-functions=1

 ## For Solaris 2, or general-purpose GCC 2.7.x
-OPTON = -O2 -fomit-frame-pointer -fforce-addr -ffast-math -Wall
+#OPTON = -O2 -fomit-frame-pointer -fforce-addr -ffast-math -Wall
+OPTON = -O3 -fomit-frame-pointer -Wall

 ## For Digital Unix v4.x, with DEC cc v5.x
 #OPTON = -O4
diff -uprN a/UnixBench/Run b/UnixBench/Run
--- a/UnixBench/Run  2015-06-05 02:20:18.000000000 +0900
+++ b/UnixBench/Run  2017-03-17 19:24:35.890585721 +0900
@@ -746,7 +746,8 @@ sub getSystemInfo {
     my $cpus = getCpuInfo();
     if (defined($cpus)) {
         $info->{'cpus'} = $cpus;
-        $info->{'numCpus'} = scalar(@$cpus);
+        #$info->{'numCpus'} = scalar(@$cpus);
+        $info->{'numCpus'} = 4;
     }

     # Get graphics hardware info.
@@ -1764,7 +1765,7 @@ sub main {
         $tests = $index;
     }

-    preChecks();
+    #preChecks();
     my $systemInfo = getSystemInfo();

     # If the number of copies to run was not set, set it to 1
```

M3:4 -> 2

- Here is an example of command execution when the patch file is saved as **cross-compile-byte-unixbench-5.1.3.patch**:

```
$ cd byte-unixbench-5.1.3
$ patch -p1 < cross-compile-byte-unixbench-5.1.3.patch
patching file UnixBench/Makefile
patching file UnixBench/Run
```

(3) Optimization Options

- byte-unixbench-5.1.3/UnixBench/Makefile
  Optimization option. The option can be "-O3", "-O2" or empty which means no optimization.
  * This modification is included in **(2) Applying patches**

```
-OPTON = -O2 -fomit-frame-pointer -fforce-addr -ffast-math -Wall
+#OPTON = -O2 -fomit-frame-pointer -fforce-addr -ffast-math -Wall
+OPTON = -O3 -fomit-frame-pointer -Wall
```

(4) Modify runtime script of UnixBench

- byte-unixbench-5.1.3/UnixBench/Run
  The number of CPU is fixed (H3: default 4; M3: default 2) because environment variable cannot be acquired normally.
  * This modification is included in **(2) Applying patches**

```
…
    # Get details on the CPUs, if possible.
    my $cpus = getCpuInfo();
    if (defined($cpus)) {
        $info->{'cpus'} = $cpus;
        #$info->{'numCpus'} = scalar(@$cpus);
        $info->{'numCpus'} = 4;
    }
…
```

M3:4 -> 2

(5) Compile the package

- Replace <toolchain_path> with the path string where the cross toolchain is installed.
  For details, refer to **2.2.1 Path of toolchain**.

```
$ source <toolchain_path>/environment-setup-aarch64-poky-linux
$ export LDFLAGS=""
$ cd byte-unixbench-5.1.3/UnixBench
$ make
```

(6) Copy the entire UnixBench folder to target file system

```
$ cp -r byte-unixbench-5.1.3/UnixBench <rootfs>/home/root/
```

(7) Add perl

perl is needed to run UnixBench. Follow the steps below to add perl to the target file system.

1. Download the following files
   http://www.cpan.org/src/5.0/perl-5.22.0.tar.bz2
   https://github.com/arsv/perl-cross/raw/releases/perl-5.22.0-cross-1.0.0.tar.gz

2. Execute the following commands
   * The options specified for ./configure depend on the Host PC's environment.
     Please execute according to your own environment.
   * Replace <toolchain_path> with the path string where the cross toolchain is installed.
     For details, refer to **2.2.1 Path of toolchain**.

```
$ tar xf perl-5.22.0.tar.bz2
$ tar xf perl-5.22.0-cross-1.0.0.tar.gz
$ cd perl-5.22.0/
$ ./configure --mode=cross --prefix=/usr --target=aarch64-poky-linux --target-tools-prefix=<toolchain_path>/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux- --sysroot=<toolchain_path>/sysroots/aarch64-poky-linux -A cc="-march=armv8-a -mtune=cortex-a57.cortex-a53 --sysroot=<toolchain_path>/sysroots/aarch64-poky-linux"
$ make
$ mkdir ../tmp_perl
$ make DESTDIR=../tmp_perl install
$ cp -r ../tmp_perl/* <rootfs>/
```

(8) Check build date

Boot the system and check the build date of the environment from the serial log.
Please refer to **2.3 Checking build date** for the check points.

(9) Execute unixbench

Execute below command to run UnixBench.

```
# cd /home/root/UnixBench
# chmod +x Run
# ./Run
```

NOTE:

If your environment supports the pids subsystem of cgroup, UnixBench may fail to run.
(Because the pids subsystem limits the number of processes that can be created)
In such a case, you need to execute the following command before running UnixBench.

```
# cd /sys/fs/cgroup/pids/system.slice/system-serial¥¥x2dgetty.slice/serial-getty¥@ttySC0.service
# cat ./pids.max
512
# echo max > ./pids.max
# cat ./pids.max
max
```

5.1.2 Evaluation Result

5.1.2.1 Evaluation Item

The evaluation item which can be measured in UnixBench is indicated in Table 5.1

Table 5.1 UnixBench_Evaluation_Item

#	Item	Explanation
1	Dhrystone	This benchmark is used to measure and compare the performance of computers. The test focuses on string handling, as there are no floating point operations
2	Whetstone	This test measures the speed and efficiency of floating-point operations
3	Execl Throughput	This test measures the number of execl calls that can be performed per second
4	File Copy	This measures the rate at which data can be transferred from one file to another, using various buffer sizes
5	Pipe Throughput	A pipe is the simplest form of communication between processes. Pipe throughput is the number of times (per second) a process can write 512 bytes to a pipe and read them back
6	Pipe-based Context Switching	This test measures the number of times two processes can exchange an increasing integer through a pipe
7	Process Creation	This test measure the number of times a process can fork and reap a child that immediately exits.
8	Shell Scripts	The shells scripts test measures the number of times per minute a process can start and reap a set of one, two, four and eight concurrent copies of a shell scripts where the shell script applies a series of transformation to a data file
9	System Call Overhead	This estimates the cost of entering and leaving the operating system kernel, i.e. the overhead for performing a system call. It consists of a simple program repeatedly calling the getpid (which returns the process id of the calling process) system call

The unit measure was used in text result:

- Lps: lines per second.
- MWIPS: generally single precision Whetstone rating in Millions of Whetstone Instructions Per Second.
- KBps: Kb per second.
- Lpm: lines per minute.

5.1.2.2 Result Overview

When UnixBench is carried out, measurement result like Figure 5.1.

```
   BYTE UNIX Benchmarks (Version 5.1.3)

   System: salvator-x: GNU/Linux
   OS: GNU/Linux -- 4.6.0-yocto-standard -- #1 SMP PREEMPT Sat Mar 25 01:33:28 JST 2017
   Machine: aarch64 (unknown)
   Language:  (charmap="ANSI_X3.4-1968", collate="ANSI_X3.4-1968")
   15:21pm  up 17252 days 15:21,  1 user,  load average: 0.01, 0.01, 0.01; runlevel 3

------------------------------------------------------------------------
Benchmark Run: Mon Mar 27 2017 15:21:29 - 15:49:30
4 CPUs in system; running 1 parallel copy of tests

Dhrystone 2 using register variables       16082988.9 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     1433.5 MWIPS (9.9 s, 7 samples)
<...>
System Call Overhead                         605854.7 lps   (10.0 s, 7 samples)

System Benchmarks Index Values               BASELINE       RESULT    INDEX
Dhrystone 2 using register variables         116700.0   16082988.9   1378.1
<...>
System Call Overhead                          15000.0     605854.7    403.9
                                                                   ========
System Benchmarks Index Score                                         584.5

------------------------------------------------------------------------
Benchmark Run: Mon Mar 27 2017 15:49:30 - 16:17:37
4 CPUs in system; running 4 parallel copies of tests

Dhrystone 2 using register variables       63904935.9 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     5735.0 MWIPS (9.9 s, 7 samples)
<...>
System Call Overhead                        2230382.8 lps   (10.0 s, 7 samples)

System Benchmarks Index Values               BASELINE       RESULT    INDEX
Dhrystone 2 using register variables         116700.0   63904935.9   5476.0
<...>
System Call Overhead                          15000.0    2230382.8   1486.9
                                                                   ========
System Benchmarks Index Score                                        1519.8
```

1 CPU result

Multiple CPU result

Figure 5.1 UnixBench_Evaluation_result

5.1.2.3 H3

(1) Evaluation Result with "-O3" Optimization

```
   BYTE UNIX Benchmarks (Version 5.1.3)

   System: salvator-x: GNU/Linux
   OS: GNU/Linux -- 4.6.0-yocto-standard -- #1 SMP PREEMPT Sat Mar 25 01:33:28 JST 2017
   Machine: aarch64 (unknown)
   Language:  (charmap="ANSI_X3.4-1968", collate="ANSI_X3.4-1968")
   15:21pm  up 17252 days 15:21,  1 user,  load average: 0.01, 0.01, 0.01; runlevel 3

------------------------------------------------------------------------
Benchmark Run: Mon Mar 27 2017 15:21:29 - 15:49:30
4 CPUs in system; running 1 parallel copy of tests

Dhrystone 2 using register variables       16082988.9 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     1433.5 MWIPS (9.9 s, 7 samples)
Execl Throughput                               2604.3 lps   (30.0 s, 2 samples)
File Copy 1024 bufsize 2000 maxblocks        249113.8 KBps  (30.0 s, 2 samples)
File Copy 256 bufsize 500 maxblocks           88090.2 KBps  (30.0 s, 2 samples)
File Copy 4096 bufsize 8000 maxblocks        541024.4 KBps  (30.0 s, 2 samples)
Pipe Throughput                              604760.4 lps   (10.0 s, 7 samples)
Pipe-based Context Switching                  90370.6 lps   (10.0 s, 7 samples)
Process Creation                               5162.1 lps   (30.0 s, 2 samples)
Shell Scripts (1 concurrent)                   3529.9 lpm   (60.0 s, 2 samples)
Shell Scripts (8 concurrent)                    927.5 lpm   (60.0 s, 2 samples)
System Call Overhead                         605854.7 lps   (10.0 s, 7 samples)

System Benchmarks Index Values               BASELINE       RESULT    INDEX
Dhrystone 2 using register variables         116700.0   16082988.9   1378.1
Double-Precision Whetstone                       55.0       1433.5    260.6
Execl Throughput                                 43.0       2604.3    605.7
File Copy 1024 bufsize 2000 maxblocks          3960.0     249113.8    629.1
File Copy 256 bufsize 500 maxblocks            1655.0      88090.2    532.3
File Copy 4096 bufsize 8000 maxblocks          5800.0     541024.4    932.8
Pipe Throughput                               12440.0     604760.4    486.1
Pipe-based Context Switching                   4000.0      90370.6    225.9
Process Creation                                126.0       5162.1    409.7
Shell Scripts (1 concurrent)                     42.4       3529.9    832.5
Shell Scripts (8 concurrent)                      6.0        927.5   1545.8
System Call Overhead                          15000.0     605854.7    403.9
                                                                   ========
System Benchmarks Index Score                                         584.5
```

```
------------------------------------------------------------------------
Benchmark Run: Mon Mar 27 2017 15:49:30 - 16:17:37
4 CPUs in system; running 4 parallel copies of tests

Dhrystone 2 using register variables       63904935.9 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     5735.0 MWIPS (9.9 s, 7 samples)
Execl Throughput                               8647.8 lps   (30.0 s, 2 samples)
File Copy 1024 bufsize 2000 maxblocks        414340.4 KBps  (30.0 s, 2 samples)
File Copy 256 bufsize 500 maxblocks          128987.5 KBps  (30.0 s, 2 samples)
File Copy 4096 bufsize 8000 maxblocks       1005210.3 KBps  (30.0 s, 2 samples)
Pipe Throughput                             2421198.8 lps   (10.0 s, 7 samples)
Pipe-based Context Switching                 438477.1 lps   (10.0 s, 7 samples)
Process Creation                              13297.3 lps   (30.0 s, 2 samples)
Shell Scripts (1 concurrent)                   7455.4 lpm   (60.0 s, 2 samples)
Shell Scripts (8 concurrent)                    953.6 lpm   (60.1 s, 2 samples)
System Call Overhead                        2230382.8 lps   (10.0 s, 7 samples)

System Benchmarks Index Values               BASELINE       RESULT    INDEX
Dhrystone 2 using register variables         116700.0   63904935.9   5476.0
Double-Precision Whetstone                       55.0       5735.0   1042.7
Execl Throughput                                 43.0       8647.8   2011.1
File Copy 1024 bufsize 2000 maxblocks          3960.0     414340.4   1046.3
File Copy 256 bufsize 500 maxblocks            1655.0     128987.5    779.4
File Copy 4096 bufsize 8000 maxblocks          5800.0    1005210.3   1733.1
Pipe Throughput                               12440.0    2421198.8   1946.3
Pipe-based Context Switching                   4000.0     438477.1   1096.2
Process Creation                                126.0      13297.3   1055.3
Shell Scripts (1 concurrent)                     42.4       7455.4   1758.4
Shell Scripts (8 concurrent)                      6.0        953.6   1589.3
System Call Overhead                          15000.0    2230382.8   1486.9
                                                                   ========
System Benchmarks Index Score                                        1519.8
```

5.1.2.4 M3

(1) Performance Result with “-O3” Optimization

```
   BYTE UNIX Benchmarks (Version 5.1.3)

   System: salvator-x: GNU/Linux
   OS: GNU/Linux -- 4.6.0-yocto-standard -- #1 SMP PREEMPT Thu Mar 23 23:43:08 JST 2017
   Machine: aarch64 (unknown)
   Language:  (charmap="ANSI_X3.4-1968", collate="ANSI_X3.4-1968")
   16:15pm  up 17253 days 16:15,  1 user,  load average: 0.00, 0.00, 0.00; runlevel 3

------------------------------------------------------------------------
Benchmark Run: Tue Mar 28 2017 16:15:38 - 16:43:39
2 CPUs in system; running 1 parallel copy of tests

Dhrystone 2 using register variables       16081576.4 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     1433.4 MWIPS (9.9 s, 7 samples)
Execl Throughput                               2364.8 lps   (30.0 s, 2 samples)
File Copy 1024 bufsize 2000 maxblocks        255372.8 KBps  (30.0 s, 2 samples)
File Copy 256 bufsize 500 maxblocks           81680.5 KBps  (30.0 s, 2 samples)
File Copy 4096 bufsize 8000 maxblocks        575318.4 KBps  (30.0 s, 2 samples)
Pipe Throughput                              605648.1 lps   (10.0 s, 7 samples)
Pipe-based Context Switching                  91039.4 lps   (10.0 s, 7 samples)
Process Creation                               4373.6 lps   (30.0 s, 2 samples)
Shell Scripts (1 concurrent)                   2887.5 lpm   (60.0 s, 2 samples)
Shell Scripts (8 concurrent)                    519.4 lpm   (60.1 s, 2 samples)
System Call Overhead                         604506.3 lps   (10.0 s, 7 samples)

System Benchmarks Index Values               BASELINE       RESULT    INDEX
Dhrystone 2 using register variables         116700.0   16081576.4   1378.0
Double-Precision Whetstone                       55.0       1433.4    260.6
Execl Throughput                                 43.0       2364.8    549.9
File Copy 1024 bufsize 2000 maxblocks          3960.0     255372.8    644.9
File Copy 256 bufsize 500 maxblocks            1655.0      81680.5    493.5
File Copy 4096 bufsize 8000 maxblocks          5800.0     575318.4    991.9
Pipe Throughput                               12440.0     605648.1    486.9
Pipe-based Context Switching                   4000.0      91039.4    227.6
Process Creation                                126.0       4373.6    347.1
Shell Scripts (1 concurrent)                     42.4       2887.5    681.0
Shell Scripts (8 concurrent)                      6.0        519.4    865.7
System Call Overhead                          15000.0     604506.3    403.0
                                                                   ========
System Benchmarks Index Score                                         536.6
```

```
------------------------------------------------------------------------
Benchmark Run: Tue Mar 28 2017 16:43:39 - 17:11:47
2 CPUs in system; running 2 parallel copies of tests

Dhrystone 2 using register variables       32065890.1 lps   (10.0 s, 7 samples)
Double-Precision Whetstone                     2877.4 MWIPS (9.9 s, 7 samples)
Execl Throughput                               3876.1 lps   (29.9 s, 2 samples)
File Copy 1024 bufsize 2000 maxblocks        389004.6 KBps  (30.0 s, 2 samples)
File Copy 256 bufsize 500 maxblocks          134314.2 KBps  (30.0 s, 2 samples)
File Copy 4096 bufsize 8000 maxblocks        952870.3 KBps  (30.0 s, 2 samples)
Pipe Throughput                             1204995.5 lps   (10.0 s, 7 samples)
Pipe-based Context Switching                 225421.3 lps   (10.0 s, 7 samples)
Process Creation                               8011.6 lps   (30.0 s, 2 samples)
Shell Scripts (1 concurrent)                   3962.1 lpm   (60.0 s, 2 samples)
Shell Scripts (8 concurrent)                    523.3 lpm   (60.2 s, 2 samples)
System Call Overhead                        1139886.9 lps   (10.0 s, 7 samples)

System Benchmarks Index Values               BASELINE       RESULT    INDEX
Dhrystone 2 using register variables         116700.0   32065890.1   2747.7
Double-Precision Whetstone                       55.0       2877.4    523.2
Execl Throughput                                 43.0       3876.1    901.4
File Copy 1024 bufsize 2000 maxblocks          3960.0     389004.6    982.3
File Copy 256 bufsize 500 maxblocks            1655.0     134314.2    811.6
File Copy 4096 bufsize 8000 maxblocks          5800.0     952870.3   1642.9
Pipe Throughput                               12440.0    1204995.5    968.6
Pipe-based Context Switching                   4000.0     225421.3    563.6
Process Creation                                126.0       8011.6    635.8
Shell Scripts (1 concurrent)                     42.4       3962.1    934.5
Shell Scripts (8 concurrent)                      6.0        523.3    872.2
System Call Overhead                          15000.0    1139886.9    759.9
                                                                   ========
System Benchmarks Index Score                                         919.4
```

5.1.2.5 Summary

Platform	Single	SMP
	O3	O3
H3	584.5	1519.8
M3	536.6	919.4

5.1.2.6 Comment

The reason that the difference between the results of H3 and M3 is increased by SMP is because the number of execution cores of CA57 is different.
* H3 (CA57x4), M3(CA57x2)

6. File System Performance Evaluation

This chapter describes procedures and results for evaluating file-system performance.

6.1 IOzone

IOzone is a filesystem benchmark tool. The benchmark generates and measures a variety of file operations. Iozone does a benchmarking on different types of file system performance metrics (e.g. Read, Write, Random read).

By using Iozone to get a broad filesystem performance coverage the buyer is much more likely to see any hot or cold spots and pick a platform and operating system that is well balanced.

6.1.1 Evaluation Procedure

(1) Download and extract source package

- Download the file at the below link:
  http://www.iozone.org/src/current/iozone3_434.tar
- Extract the downloaded archive file to any directory.

```
$ tar xf iozone3_434.tar
$ ls
iozone3_434
```

(2) Applying patches

- Save the contents of the following text-box as a patch file, and apply the patch to the source package.

```
diff -urpN a/src/current/makefile b/src/current/makefile
--- a/src/current/makefile     2015-10-20 23:12:13.000000000 +0900
+++ b/src/current/makefile     2017-04-13 19:21:45.676139909 +0900
@@ -9,9 +9,10 @@
 #                  convex, FreeBSD, OpenBSD, OSFV3, OSFV4, OSFV5, SCO
 #                  SCO_Unixware_gcc,NetBSD,TRU64, Mac OS X

-CC         = cc
+#CC        = cc
 C89        = c89
-GCC        = gcc
+#GCC       = gcc
+GCC        = $(CC)
 CCS        = /usr/ccs/bin/cc
 NACC       = /opt/ansic/bin/cc
 CFLAGS     =
```

- Here is an example of command execution when the patch file is saved as **cross-compile-iozone3_434.patch**:

```
$ cd iozone3_434
$ patch -p1 < cross-compile-iozone3_434.patch
File src/current/makefile is read-only; trying to patch anyway
patching file src/current/makefile
```

(3) Compile the package

- Replace <toolchain_path> with the path string where the cross toolchain is installed.
  For details, refer to **2.2.1 Path of toolchain**.

```
$ source <toolchain_path>/environment-setup-aarch64-poky-linux
$ export LDFLAGS=""
$ cd iozone3_434/src/current/
$ make linux
```

(4) Archive source package and copy it to the target file system

```
$ tar czf iozone3_434.tgz iozone3_434
$ cp iozone3_434.tgz <rootfs>/home/root/
```

(5) Check build date

- Boot the system and check the build date of the environment from the serial log.
  Please refer to **2.3 Checking build date** for the check points.

(6) Execute the benchmark on target

- The benchmark can be executed on the following memory devices:
  - USB flash drive / SD card / eMMC / RAM disk (tmpfs)

  Of these, the USB flash drive, SD card, eMMC must be formatted with **ext4**.
- The device file name (e.g. **/dev/mmcblk0p1**) in the following procedure may be different depending on each environment.
- If the device to be measured and the device where rootfs is expanded are the same, the procedure of mount / unmount in the following procedure is unnecessary.

❖ The option of iozone:
The optione of iozone is too large, for more detail refer site: https://linux.die.net/man/1/iozone

In this instruction using example command: iozone -i 0 -i 1 -g 1048576 -a -R -b usb_result.xls mean

- -i 0 i 1: used to specify which tests to run. (0=write/rewrite, 1=read/re-read).
- -g 1048576: set maximum file size (in Kbytes) is 1048576 Kb for auto mode.
- -a: used to select full automatic mode. Produces output that covers all tested file operations for record sizes of 4k to 16M for file sizes of 64k to 512M.
- -R: generate Excel report. Iozone will generate an Excel compatible report to standard out.
- -b usb_result.xls: with option –R used to specify a filename “usb_result.xls” that will be used for output of an Excel compatible file that contains the results, file name may be changed when starting testing on USB flash drive / SD card / eMMC / RAM disk (tmpfs).

A) Benchmark execution with USB flash drive

1. Insert USB flash drive into **CN10**
2. Create mount point and mount the USB flash drive
3. Extract the source package
4. Execute benchmark and save results
5. Unmount USB flash drive

The following is an example of command execution:

```
# # 2. Command exmaple
# mkdir /home/root/usb
# mount /dev/sda1 /home/root/usb
# # 3. Command exmaple
# cd /home/root/usb
# tar xf /home/root/iozone3_434.tgz
# # 4. Command exmaple
# cd /home/root/usb/iozone3_434/src/current/
# ./iozone -i 0 -i 1 -g 1048576 -a -R -b usb_result.xls
<... snip ...>
# cp usb_result.xls /home/root/
# # 5. Command exmaple
# cd /home/root
# umount /home/root/usb
```

B) Benchmark execution with SD card
1. Insert SD card into **CN13**
2. Create mount point and mount the SD card
3. Extract the source package
4. Execute benchmark and save results
5. Unmount SD card

The following is an example of command execution:

```
# # 2. Command exmaple
# mkdir /home/root/sd
# mount /dev/mmcblk1p1 /home/root/sd
# # 3. Command exmaple
# cd /home/root/sd
# tar xf /home/root/iozone3_434.tgz
# # 4. Command exmaple
# cd /home/root/sd/iozone3_434/src/current/
# ./iozone -i 0 -i 1 -g 1048576 -a -R -b sd_result.xls
<... snip ...>
# cp sd_result.xls /home/root/
# # 5. Command exmaple
# cd /home/root
# umount /home/root/sd
```

C) Benchmark execution with eMMC
1. Create mount point and mount the eMMC
2. Extract the source package
3. Execute benchmark and save results
4. Unmount eMMC

The following is an example of command execution:

```
# # 1. Command exmaple
# mkdir /home/root/emmc
# mount /dev/mmcblk0p1 /home/root/emmc
# # 2. Command exmaple
# cd /home/root/emmc
# tar xf /home/root/iozone3_434.tgz
# # 3. Command exmaple
# cd /home/root/emmc/iozone3_434/src/current/
# ./iozone -i 0 -i 1 -g 1048576 -a -R -b emmc_result.xls
<... snip ...>
# cp emmc_result.xls /home/root/
# # 4. Command exmaple
# cd /home/root
# umount /home/root/emmc
```

D) Benchmark execution with RAM disk (tmpfs)
1. Extract the source package
2. Execute benchmark and save results

The following is an example of command execution:

```
# # 1. Command exmaple
# cd /tmp
# tar xf /home/root/iozone3_434.tgz
# # 2. Command exmaple
# cd /tmp/iozone3_434/src/current/
# ./iozone -i 0 -i 1 -g 1048576 -a -R -b ram_result.xls
<... snip ...>
# cp ram_result.xls /home/root/
```

6.1.2 Evaluation Result

- After finished testing, the result will be displayed on terminal and import to excel file (if include –R and –b option), the format of result in this instruction is like the table below:

kB	reclen	write	rewrite	read	reread

- kB: display the size of data was tranfered.
- Reclen: the record lengh value was used for testing in each type of size of data.
- Read: indicates the performance of reading a file that already exists in the filesystem.
- Write: indicates the performance of writing a new file to the filesystem.
- Re-read: after reading a file, this indicates the performance of reading a file again.
- Re-write: indicates the performance of writing to an existing file.

- Note: reclen, read, re-read, write, re-write use Kbyte for unit measurement.

6.2 bonnie++

Bonnie++ allows you to benchmark how your filesystems perform various tasks, which makes it a valuable tool when you are making changes to how your RAID is set up, how your filesystems are created, or how your network filesystems perform.

Bonnie++ benchmarks three things: data read and write speed, number of seeks that can be performed per second, and number of file metadata operations that can be performed per second. Metadata operations include file creation and deletion as well as getting metadata such as the file size or owner.

6.2.1 Evaluation Procedure

(1) Download and extract source package

- Download the file at the below link:
  http://www.coker.com.au/bonnie++/bonnie++-1.03e.tgz
- Extract the downloaded archive file to any directory.
  * Depending on the command used for downloading, the saved filename may change.
  In that case, replace the filename specified for tar command.
  Example: tar xf bonnie++-1.03e.gz

```
$ wget http://www.coker.com.au/bonnie++/bonnie++-1.03e.tgz
$ tar xf bonnie++-1.03e.tgz
$ ls
bonnie++-1.03e
```

(2) Execute configure script

- Replace <toolchain_path> with the path string where the cross toolchain is installed.
  For details, refer to **2.2.1 Path of toolchain**.
- Some files (Makefile, bonnie.h, etc ...) are overwritten by configure script.

```
$ source <toolchain_path>/environment-setup-aarch64-poky-linux
$ export LDFLAGS=""
$ cd bonnie++-1.03e
$ ./configure --host=aarch64-poky-linux
```

(3) Applying patches

- Save the contents of the following text-box as a patch file, and apply the patch to the source package.
- This patch prevents the result from being displayed as "+++ ..." if the measurement time is too short.

```
diff -urpN a/bonnie.h b/bonnie.h
--- a/bonnie.h       2008-12-10 07:19:43.000000000 +0900
+++ b/bonnie.h       2017-04-12 13:54:34.861887400 +0900
@@ -14,7 +14,7 @@
 // data includes index to which directory (6 bytes) and terminating '¥0' for
 // the name and pointer to file name
 #define MaxDataPerFile (MaxNameLen + 6 + 1 + 4)
-#define MinTime (0.5)
+#define MinTime (0.01)
 #define Seeks (8192)
 #define UpdateSeek (10)
 #define SeekProcCount (3)
```

- Here is an example of command execution when the patch file is saved as **cross-compile-bonnie++-1-03e.patch**:

```
$ cd bonnie++-1.03e
$ patch -p1 < cross-compile-bonnie++-1-03e.patch
patching file bonnie.h
```

(4) Compile the package

- To statically link the required libraries, set the MORECFLAGS environment variable to "-static".

```
$ cd bonnie++-1.03e
$ export MORECFLAGS=-static
$ make
```

(5) Archive source package and copy it to the target file system

(6) Check build date

- Boot the system and check the build date of the environment from the serial log.
  Please refer to **2.3 Checking build date** for the check points.

(7) Execute the benchmark on target

- The benchmark can be executed on the following memory devices:
  - ➢ USB flash drive / SD card / eMMC / RAM disk (tmpfs)

  Of these, the USB flash drive, SD card, eMMC must be formatted with **ext4**.
- The device file name (e.g. **/dev/mmcblk0p1**) in the following procedure may be different depending on each environment.
- If the device to be measured and the device where rootfs is expanded are the same, the procedure of mount / unmount in the following procedure is unnecessary.
- Benchmark results are output to the console. Save them appropriately.

❖ The option of bonnie++:
The optione of bonnie++ is too large, for more detail refer site: https://linux.die.net/man/8/bonnie++

In this instruction using example command: bonnie++ -u 0:0 -b -s 20M -r 10M -d /home/root/usb mean

- -u 0:0: user 0 and id=0 is used when run the app as root.
- -b: no write buffering. fsync() after every write.
- -s 20M: the size of the file(s) for IO performance measures in megabytes is 20 Mb.
- -r 10M: RAM size in megabytes is used in current testing is 10 Mbyte.
- -d /home/root/usb: the directory to use for the tests is /home/root/usb, the directoty will be changed when starting testing on USB flash drive / SD card / eMMC / RAM disk (tmpfs).

A) Benchmark execution with USB flash drive

1. Insert USB flash drive into **CN10**
2. Create mount point and mount the USB flash drive
3. Extract the source package
4. Execute benchmark and save results
5. Unmount USB flash drive

The following is an example of command execution:

```
# # 2. Command exmaple
# mkdir /home/root/usb
# mount /dev/sda1 /home/root/usb
# # 3. Command exmaple
# cd /home/root/usb
# tar xf /home/root/bonnie++-1.03e.tgz
# # 4. Command exmaple
# cd /home/root/usb/bonnie++-1.03e/
# ./bonnie++ -u 0:0 -b -s 20M -r 10M -d /home/root/usb
<... Benchmark results are output here. ...>
# # 5. Command exmaple
# cd /home/root
# umount /home/root/usb
```

B) Benchmark execution with SD card
1. Insert SD card into **CN13**
2. Create mount point and mount the SD card
3. Extract the source package
4. Execute benchmark and save results
5. Unmount SD card

The following is an example of command execution:

```
# # 2. Command exmaple
# mkdir /home/root/sd
# mount /dev/mmcblk1p1 /home/root/sd
# # 3. Command exmaple
# cd /home/root/sd
# tar xf /home/root/bonnie++-1.03e.tgz
# # 4. Command exmaple
# cd /home/root/sd/bonnie++-1.03e/
# ./bonnie++ -u 0:0 -b -s 20M -r 10M -d /home/root/sd
<... Benchmark results are output here. ...>
# # 5. Command exmaple
# cd /home/root
# umount /home/root/sd
```

C) Benchmark execution with eMMC
1. Create mount point and mount the eMMC
2. Extract the source package
3. Execute benchmark and save results
4. Unmount eMMC

The following is an example of command execution:

```
# # 1. Command exmaple
# mkdir /home/root/emmc
# mount /dev/mmcblk0p1 /home/root/emmc
# # 2. Command exmaple
# cd /home/root/emmc
# tar xf /home/root/bonnie++-1.03e.tgz
# # 3. Command exmaple
# cd /home/root/emmc/bonnie++-1.03e/
# ./bonnie++ -u 0:0 -b -s 20M -r 10M -d /home/root/emmc
<... Benchmark results are output here. ...>
# # 4. Command exmaple
# cd /home/root
# umount /home/root/emmc
```

D) Benchmark execution with RAM disk (tmpfs)
1. Extract the source package
2. Execute benchmark and save results

The following is an example of command execution:

```
# # 1. Command exmaple
# cd /tmp
# tar xf /home/root/bonnie++-1.03e.tgz
# # 2. Command exmaple
# cd /tmp/bonnie++-1.03e/
# ./bonnie++ -u 0:0 -b -s 20M -r 10M -d /tmp
<... Benchmark results are output here. ...>
```

6.2.2 Evaluation Result

- After finished testing, the result will be displayed on terminal and have the format like below:

```
              ------Sequential Output------ --Sequential Input- --Random-
                -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine       Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP  /sec %CP
salvator-x     20M 14216 100 96378  26 92196  19 15063 100 1560274  91 16157  29
                ------Sequential Create------ --------Random Create--------
                -Create-- --Read--- -Delete-- -Create-- --Read--- -Delete--
          files  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
             16   856   5 301442 103   759   4   846   5 382018 102   761   4
```

- Sequential Output
  - ✓ Per-Character: the file is written using the putc() stdio macro.
  - ✓ Block: the file is created using block write().
  - ✓ Rewrite: each Chunk (currently, the size is 16384) of the file is read with read() and rewritten with write(2), requiring an lseek().
- Sequential Input:
  - ✓ Per-Character: the file is read using the getc() stdio macro.
  - ✓ Block: the file is read using block read().

- Random Seeks:
  - ✓ This test runs SeekProcCount (currently 4) processes in parallel, doing a total of 4000 lseek()s to random locations in the file.
- Sequential Create:
  - ✓ Create/read/detele: is the action to the file system (in this instruction is 16 files).
- Random Create: like Sequential Create but the action is being done in ramdom files.
- k/sec mean uses Kbyte per second for unit measurement.
- %CP mean the performance of CPU used when run this process.

7. Driver Performance Evaluation

This chapter describes procedures and results for evaluating driver performance.

7.1 Dd

You won't find a more versatile utility than tar to create a file system–based backup. In some cases, however, you don't need a backup based on a file system; instead, you want to create a backup of a complete device or parts of it. This is where the dd command comes in handy.

The Linux dd command is one of the most powerful utility which can be used in a variety of ways. This tool is mainly used for copying and converting data, hence it stands for data duplicator. This tool can be used for:

- Backing up and restoring an entire hard drive or a partition.
- Creating virtual filesystem and backup images of CD or DVDs called ISO files
- Copy regions of raw device files like backing up MBR (master boot record).
- Converting data formats like ASCII to EBCDIC.
- Converting lowercase to uppercase and vice versa.

7.1.1 Evaluation Procedure

(1) Check build date

- Boot the system and check the build date of the environment from the serial log.
  Please refer to **2.3 Checking build date** for the check points.

(2) Execute the benchmark on target

- The benchmark can be executed on the following memory devices:
  - USB flash drive / SD card / eMMC
- The device file name (e.g. **/dev/mmcblk0**) in the following procedure may be different depending on each environment.
- If the device to be measured and the device where rootfs is expanded are the same, the procedure of insert in the following procedure is unnecessary.
- Benchmark results are output to the console. Save them appropriately.

❖ The option of dd command:
The optione of dd command is too large, for more detail refer site:
https://www.computerhope.com/unix/dd.htm

In this instruction using example command: dd if=/dev/sda of=/dev/null bs=1M count=100 iflag=direct mean

- if=/dev/sda: read from FILE instead of stdin (source file name from /dev/sda).
- of=/dev/null: write to FILE instead of stdout (target file name from /dev/null).
- bs=1M: read and write 1Mbyte at a time.
- count=100: copy only 100 input blocks.
- iflag=direct: use direct I/O for data

A) Benchmark execution with USB flash drive
   1. Insert USB flash drive into **CN10**
   2. Execute benchmark and save results

The following is an example of command execution:

```
# # 2. Command exmaple
# dd if=/dev/sda of=/dev/null bs=1M count=100 iflag=direct
<... Benchmark results are output here. ...>
```

B) Benchmark execution with SD card
   1. Insert SD card into **CN13**
   2. Execute benchmark and save results

The following is an example of command execution:

```
# # 2. Command exmaple
# dd if=/dev/mmcblk1 of=/dev/null bs=1M count=100 iflag=direct
<... Benchmark results are output here. ...>
```

C) Benchmark execution with eMMC
   1. Execute benchmark and save results

The following is an example of command execution:

```
# # 1. Command exmaple
# dd if=/dev/mmcblk0 of=/dev/null bs=1M count=100 iflag=direct
<... Benchmark results are output here. ...>
```

7.1.2 Evaluation Result

- After finished testing, the result will be displayed on terminal and have the format like below:
- This is the result sample when run command on eMMC situation:
  100+0 records in
  100+0 records out
  104857600 bytes (105 MB, 100 MiB) copied, 0.731436 s, 143 MB/s

  ✓ X records in: the number of input blocks (set by count option).
  ✓ X records in: the number of output blocks (set by count option).
  ✓ 104857600 bytes (105 MB, 100 MiB): the total data have been tranfered when testing (set by bs option).
  ✓ 0.731436 s: the total time to transfer all data.
  ✓ 143 MB/s: the speed to transfer data, it is also the result for this testing.

8. Boot Process Performance Evaluation

This chapter describes procedures and results for evaluating boot-process performance.

8.1 Bootchart

Bootchart is a tool for performance analysis and visualization of the GNU/Linux boot process. Resource utilization and process information are collected during the boot process and are later rendered in a PNG, SVG or EPS encoded chart.

Bootchart provides a shell script to be run by the kernel in the init phase. The script will run in background and collect process information, CPU statistics and disk usage statistics from the /proc file system. The performance data are stored in memory and are written to disk once the boot process completes.

The boot log file is later processed using a Java application (or the web form) which builds the process tree and renders a performance chart in different formats.

8.1.1 Evaluation Procedure

8.1.1.1 Building bootchart

(1) Download and extract source package

- Download the file at the below link:
  https://github.com/sofar/bootchart/archive/v1.16.tar.gz
- Extract the downloaded archive file to any directory.
  * Depending on the command used for downloading, the saved filename may change.
  In that case, replace the filename specified for tar command.
  Example: tar xf bootchart-1.16.tar.gz

```
$ wget https://github.com/sofar/bootchart/archive/v1.16.tar.gz
$ tar xf v1.16.tar.gz
$ ls
bootchart-1.16
```

(2) Applying patches

- Save the contents of the following text-box as a patch file, and apply the patch to the source package.

```
diff -urpN a/Makefile b/Makefile
--- a/Makefile      2012-03-29 02:26:52.000000000 +0900
+++ b/Makefile      2017-04-18 15:33:12.549025960 +0900
@@ -1,7 +1,7 @@

 VERSION = 1.16

-CC := gcc
+#CC := gcc

 all: bootchartd
```

- Here is an example of command execution when the patch file is saved as **cross-compile-bootchart-1-16.patch**:

```
$ cd bootchart-1.16
$ patch -p1 < cross-compile-bootchart-1-16.patch
patching file Makefile
```

(3) Compile the package

- Replace <toolchain_path> with the path string where the cross toolchain is installed.
  For details, refer to **2.2.1 Path of toolchain**.

```
$ source <toolchain_path>/environment-setup-aarch64-poky-linux
$ export LDFLAGS=""
$ cd bootchart-1.16
$ make
```

(4) Copy bootchartd to /usr/sbin of target file system

```
$ cp bootchart-1.16/bootchartd <rootfs>/usr/sbin/
```

8.1.1.2 Building kernel Image

- To execute bootchart, you need a kernel image with the following config enabled:
  - CONFIG_PROC_FS
  - CONFIG_SCHED_DEBUG
  - CONFIG_SCHEDSTATS
- Follow the steps below to build the kernel with the above config enabled.

NOTE:
"**$WORK**" in the following procedure has the same meaning as that of **1.3 Reference** No.3.

(1) Execute the Yocto build procedure

- Execute the procedure up to ***3.1 In case of BSP + 3D Graphics + Multimedia package*** - ***Step 10 select SoC*** of **1.3 Reference** No.3.

(2) Add meta-bootchart1 layer

- Save the contents of the following text-box as a patch file, and apply the patch to the **$WORK** directory.

```
diff -uprN a/meta-bootchart1/conf/layer.conf b/meta-bootchart1/conf/layer.conf
--- a/meta-bootchart1/conf/layer.conf     1970-01-01 09:00:00.000000000 +0900
+++ b/meta-bootchart1/conf/layer.conf     2017-04-18 15:55:24.681054259 +0900
@@ -0,0 +1,10 @@
+# We have a conf and classes directory, add to BBPATH
+BBPATH .= ":${LAYERDIR}"
+
+# We have recipes-* directories, add to BBFILES
+BBFILES += "${LAYERDIR}/recipes-*/*/*.bb ¥
+          ${LAYERDIR}/recipes-*/*/*.bbappend"
+
+BBFILE_COLLECTIONS += "bootchart1"
+BBFILE_PATTERN_bootchart1 = "^${LAYERDIR}/"
+BBFILE_PRIORITY_bootchart1 = "7"
diff -uprN a/meta-bootchart1/recipes-kernel/linux/linux-renesas/salvator-x/linux-renesas_%.bootchart1.cfg b/meta-
bootchart1/recipes-kernel/linux/linux-renesas/salvator-x/linux-renesas_%.bootchart1.cfg
--- a/meta-bootchart1/recipes-kernel/linux/linux-renesas/salvator-x/linux-renesas_%.bootchart1.cfg      1970-01-01
09:00:00.000000000 +0900
+++ b/meta-bootchart1/recipes-kernel/linux/linux-renesas/salvator-x/linux-renesas_%.bootchart1.cfg      2017-04-10
19:32:43.014647451 +0900
@@ -0,0 +1,2 @@
+CONFIG_SCHED_DEBUG=y
+CONFIG_SCHEDSTATS=y
diff -uprN a/meta-bootchart1/recipes-kernel/linux/linux-renesas_%.bbappend b/meta-bootchart1/recipes-kernel/linux/linux-
renesas_%.bbappend
--- a/meta-bootchart1/recipes-kernel/linux/linux-renesas_%.bbappend      1970-01-01 09:00:00.000000000 +0900
+++ b/meta-bootchart1/recipes-kernel/linux/linux-renesas_%.bbappend      2017-04-18 16:11:57.269075346 +0900
@@ -0,0 +1,7 @@
+
+FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
+
+CFG_FILE_BOOTCHART1 = "file://${PN}_%.bootchart1.cfg"
+SRC_URI_append = " ¥
+    ${CFG_FILE_BOOTCHART1} ¥
+"
```

- Here is an example of command execution when the patch file is saved as **meta-bootchart1.patch**:

```
$ cd $WORK
$ patch -p1 < meta-bootchart1.patch
patching file meta-bootchart1/conf/layer.conf
patching file meta-bootchart1/recipes-kernel/linux/linux-renesas/salvator-x/linux-renesas_%.bootchart1.cfg
patching file meta-bootchart1/recipes-kernel/linux/linux-renesas_%.bbappend
```

- The contents of **$WORK** after applying the patch:

```
$ ls $WORK
build  meta-bootchart1  meta-linaro  meta-openembedded  meta-renesas  poky
```

(3) Adding a layer path

- Add a layer path to **$WORK/build/conf/bblayers.conf**.
  (Add **red boldface part**)

```
BBLAYERS ?= " ¥
  ${TOPDIR}/../poky/meta ¥
  ${TOPDIR}/../poky/meta-poky ¥
  ${TOPDIR}/../poky/meta-yocto-bsp ¥
  ${TOPDIR}/../meta-renesas/meta-rcar-gen3 ¥
  ${TOPDIR}/../meta-linaro/meta-linaro-toolchain ¥
  ${TOPDIR}/../meta-linaro/meta-optee ¥
  ${TOPDIR}/../meta-openembedded/meta-oe ¥
  ${TOPDIR}/../meta-bootchart1 ¥
  "
```

- Make sure that the layer is added.
  (**$WORK** in the following execution example is actually an absolute path.)

```
$ cd $WORK/build
$ bitbake-layers show-layers
layer                 path                                      priority
==========================================================================
meta                  $WORK/build/../poky/meta  5
meta-poky             $WORK/build/../poky/meta-poky  5
meta-yocto-bsp        $WORK/build/../poky/meta-yocto-bsp  5
meta-rcar-gen3        $WORK/build/../meta-renesas/meta-rcar-gen3  6
meta-linaro-toolchain  $WORK/build/../meta-linaro/meta-linaro-toolchain  30
meta-optee            $WORK/build/../meta-linaro/meta-optee  8
meta-oe               $WORK/build/../meta-openembedded/meta-oe  6
meta-bootchart1       $WORK/build/../meta-bootchart1  7
```

(4) Execute bitbake command

- Execute the bitbake command to build the Linux BSP.

```
$ cd $WORK/build
$ bitbake core-image-weston
```

8.1.1.3 Execute bootchart

(1) Check build date

- Boot the system and check the build date of the environment from the serial log.
  Please refer to **2.3 Checking build date** for the check points.

(2) Changing kernel boot parameter

- To execute bootchart at system bootup, add "init=..." to kernel boot parameter.
- The following is an example of setting boot parameters when boot from u-boot:

```
U-Boot 2015.04 (Apr 10 2017 - 20:19:20)

CPU: Renesas Electronics R8A7796 rev 1.0
Board: Salvator-X
I2C:   ready
DRAM:  3.9 GiB
MMC:   sh-sdhi: 0, sh-sdhi: 1, sh-sdhi: 2
In:    serial
Out:   serial
Err:   serial
Net:   ravb
Hit any key to stop autoboot:  3  0
=> setenv bootargs 'console=ttySC0,115200 rw root=/dev/mmcblk1p1 rootwait rootfstype=ext4 consoleblank=0
init=/usr/sbin/bootchartd'
```

(3) Preparing to execute the bootchart

- **In the Yocto 2.12.0 or 2.19.0 environment on the M3 board**, weston will not start up.
  To prevent this, execute the following command:

```
# echo pvrsrvkm > /etc/modules-load.d/pvrsrvkm.conf
# sed -i 's/¥(^After.*¥)/¥1 multi-user.target/g' /lib/systemd/system/weston.service
# grep After /lib/systemd/system/weston.service
After=rc.pvr.service multi-user.target
```

- When the system is shut down without properly unmounting the boot device, journal recovery is performed the next time it is booted up.
  To prevent this from affecting the results of bootchart, execute the following command and shut it down properly.

```
# sync
# shutdown -h now
```

(4) Boot system and save result

- After the system is booted, the result is written to /var/log, so save it.

```
salvator-x login: root
#
# bootchartd: Wrote /var/log/bootchart-20170407-0643.svg

# cp /var/log/bootchart-20170407-0643.svg /home/root/
```

8.1.2 Evaluation Result

- The output of bootchart is a file .svg, you can use browser chrome or firefox to open this file a show how the process have done when starting system boot.
- In this instruction we only forcus on the systemd start point and Weston start point.