Linux development evolves rapidly. The performance and scalability of the OS kernel has been a key part of its success. However, discussions have appeared on LKML (Linux Kernel Mailing List) regarding large performance regression between kernel versions. These discussions underscore the need for a systematic and disciplined way to characterize, improve, and test Linux kernel performance. Our goal is to work with the Linux community to further enhance the Linux kernel with consistent performance increases (avoiding degradations) across releases. The information available on this site gives community members better information about what 0-Day and LKP (Linux Kernel Performance) are doing to preserve performance integrity of the kernel.

ADDRESS RANGE PARTIAL MEMORY MIRRORING ON LINUX*

BY **TONY LUCK** ON NOV 15, 2016

INTRODUCTION

Mission critical servers must avoid down time at all costs, either because they are a single point of failure in a broader system, or because the time to switch to a backup server is unacceptable for the running applications. Intel® Architecture Processors have supported recoverable machine checks for several generations, and Linux* will respond by signaling or terminating just the affected application. But there is a small, but important, gap in the recovery path. If the error was triggered while executing OS kernel code, then the error is fatal. It is simply not possible to rewrite every part of the Linux kernel to check for errors on every memory access and take some recovery action. Instead we turn to hardware memory mirrors to solve this problem.

In its simplest form, memory mirroring simply keeps two copies (often called the *primary* and *secondary*) of the data.

On a large server, memory might make up a significant portion of the hardware cost. Reducing the amount of memory available to the OS and applications by a factor of two invariably has a large performance cost as well. Therefore, only the most mission-critical systems mirror all of their memory.

Partial memory mirroring technology minimizes the performance and hardware costs of full memory mirroring. With partial memory mirroring, the hardware can be configured to mirror just certain address ranges on each memory controller, leaving the rest of memory unmirrored. This solution provides Linux servers with the stability and reliability of mirrored memory on a granular, as-needed basis, while minimizing the additional memory (and cost) required.

The Linux operating system has been modified to negotiate how much memory should be mirrored with the platform firmware. It is not possible to adjust this amount at run time; a reboot is required each time you wish to change the amount of mirrored memory. Because Linux can already recover from errors in application memory without the entire system crashing, on partially mirrored systems Linux uses the mirrored memory for all *kernel* allocations and the unmirrored memory for *application* allocations. Partial memory mirroring gives Linux servers higher system availability than non-mirrored servers at a lower cost than servers with full memory mirroring.

MIRRORING MECHANICS

In each mirrored area of memory the memory controller keeps two copies of its data in memory. Any time the memory is updated (either by the CPU or by I/O) the memory controller writes the new value to both copies. When the data is read, the memory controller first tries to read the primary copy. If that successfully passes the ECC checks, the data is provided to the reader immediately. If the primary read fails, then the memory controller begins the recovery process described below:

- Read the secondary copy. If that also fails, then the data is lost and a machine check is logged and signaled.
- If the secondary copy is good, then the data is returned to the reader. The memory controller also attempts to fix the primary copy by updating it with the correct data.
- The memory controller rereads the primary copy. If the error was temporary, then this read will succeed and the memory controller logs a corrected error with an indication that the mirror is still good. If rereading the primary copy gives an error again, then the error is more serious. This is still logged as a corrected error, but includes an indication that the mirror is failing.

For performance reasons the memory controller might interleave the primary copies across channels so that read operations on the mirrored area take advantage of aggregate bandwidth of all channels. For example, Figure 1 shows a mirror split across two channels and a trivial alternating interleave scheme:

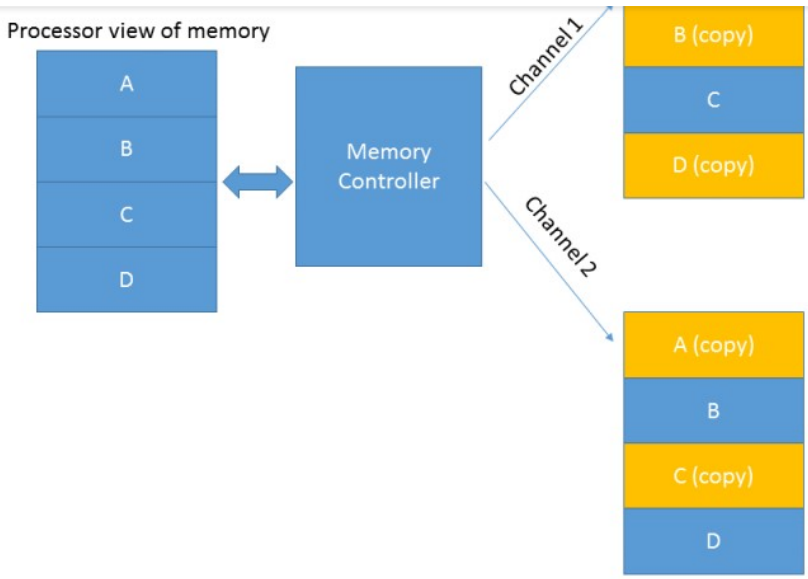


Figure 1: Mirror channel interleave

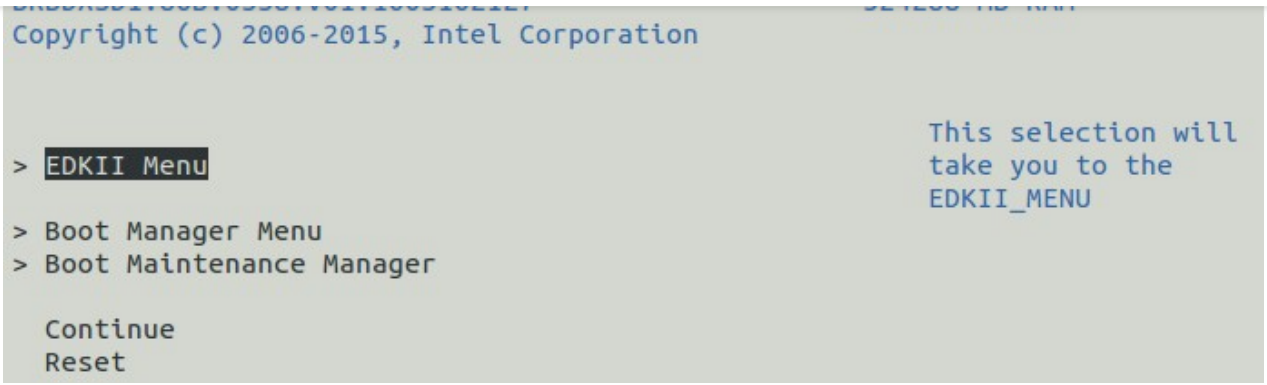
GUIDE TO ENABLING PARTIAL RANGE MEMORY MIRRORING

The remainder of this document is a step-by-step guide to setting up a Linux system using mirrored memory.

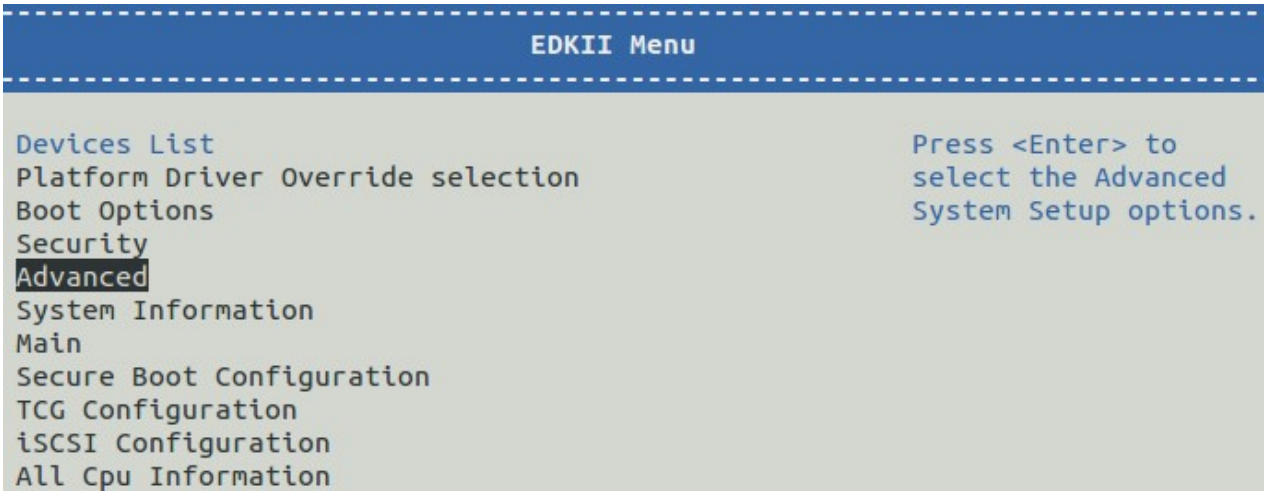
BIOS SETTINGS

The steps and menu items used to enable memory mirroring in the following example might differ depending on your system hardware; refer to your own system's BIOS setup for specific instructions. All of the following work was done on an Intel® software development platform code named Thunder Ridge with four Intel® Xeon® E7-8890 processors. Memory mirroring is disabled by default, so first we need to enable partial memory mirroring through the BIOS setup menu.

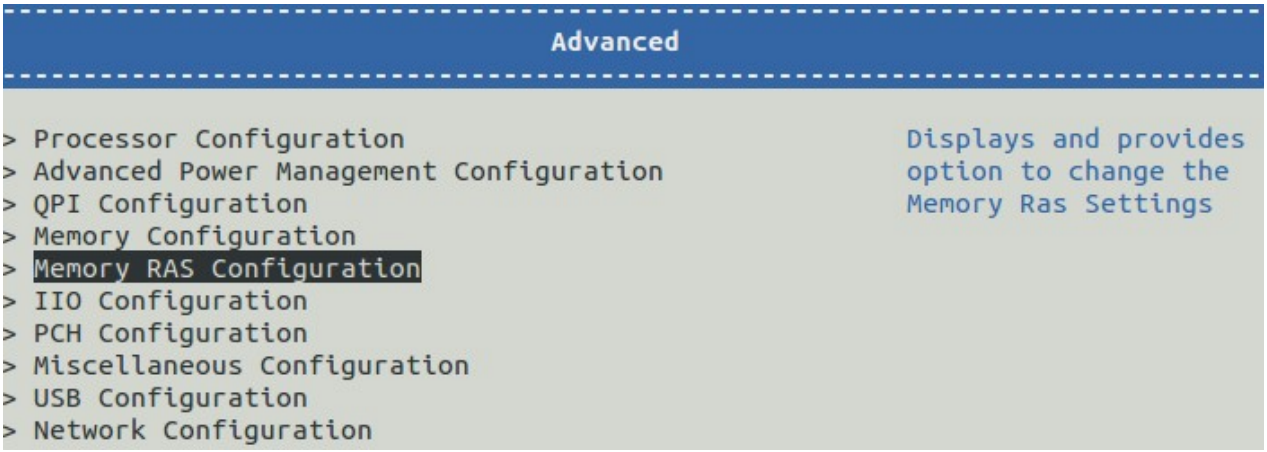
1. Select the *EDKII Menu* option from the top level menu:



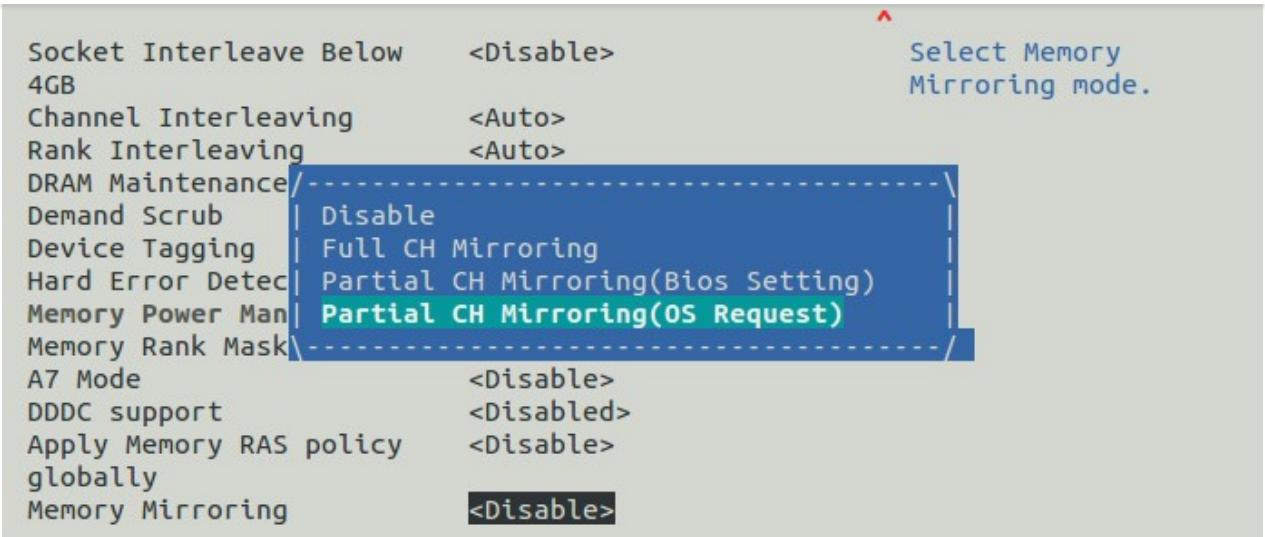
2. In the EDKII menu, choose *Advanced*:



3. Select *Memory RAS Configuration*:



4. Scroll all the way to the bottom of the *Memory RAS Configuration* menu to the *Memory Mirroring* option. Change this option from *<Disable>* to *Partial CH Mirroring (OS Request)*.



5. Press the F10 key to save settings and return to the top level menu to reset the system. Then boot to Linux.

THE EFIBOOTMGR(8) TOOL

This tool communicates with the BIOS using UEFI persistent variables[1] to pass requests to the BIOS that take effect on the next reboot, and also allows the BIOS to pass information back to Linux. Partial memory mirroring is a new feature, so the version of this tool on your system might not have support for mirroring.

You can check to see if your version supports memory mirroring by running the tool with no arguments. If it it reports `MirroredPercentageAbove4G` and `MirrorMemoryBelow4GB` values, memory mirroring is supported.

If it does not, you can build the latest version using the following steps:

```
$ git clone git://github.com/rhinstaller/efivar
$ cd efivar
$ make
$ su
Password:
# make install
# exit
$ cd ..
$ git clone git://github.com/rhinstaller/efibootmgr
$ cd efibootmgr
$ make
$ su
Password:
# make install
# efibootmgr
BootCurrent: 0000
Timeout: 10 seconds
```

```
Boot0002 Boot Device List
Boot0003* UEFI Kingston DataTraveler 2.0 000AEB91BD07A980450F0108
Boot0004* UEFI TEAC DV-W28S-W
Boot0005* UEFI SAS 031A92CD1AD5A961-SAS-EXTERNAL-EXPANDED
MirroredPercentageAbove4G: 0.00
MirrorMemoryBelow4GB: false
```

The last two lines of output here are a message from the BIOS contained in a UEFI boot variable that tells us that the platform supports memory mirroring, memory mirroring is enabled in the BIOS, but that no memory is currently being mirrored.

We use the efibootmgr tool to update the UEFI boot variable with a request that the BIOS will fulfill during the next boot:

```
# efibootmgr -m 1 -M 11.25
```

This produces the same output as shown above, plus two extra lines:

```
RequestMirroredPercentageAbove4G: 11.25
RequestMirrorMemoryBelow4GB: true
```

The `-m 1` flag specifies whether you want the memory below 4 GB to be mirrored, which you almost always will. The `-M` flag specifies the percentage of memory above 4 GB that should be mirrored. You can specify this as a floating point value with up to two decimal places. The BIOS applies this percentage separately to each ACPI proximity domain to ensure that some mirrored memory will be available on each NUMA node.

KERNEL BOOT PARAMETER

Linux kernel support for mirrored memory is in version 4.6 and higher, but we need to tell Linux to reserve mirrored memory for use by the kernel only. Assuming your Linux distribution uses the grub2 bootloader, this is done by adding the `kernelcore=mirror` parameter to your *grub.cfg* file. The exact location of this file varies between distributions. A reboot is required after adding this parameter so the changes will take effect.

RUNNING A SYSTEM WITH MIRRORED MEMORY

First we need to run a couple of checks to confirm that our setup worked.

We can use the following to check the console log and confirm that the BIOS mirrored the memory we asked for, and that the Linux kernel found and used the mirrored memory:

```
# dmesg | grep mirrored
[ 0.000000] efi: Memory: 60416M/464128M mirrored memory
```

available memory by two gigabytes because we need a primary and secondary copy of all mirrored data.

So if we adjust the total memory to what it would have been without any mirroring we have 60418/(60418+464128), which is 11.5%. Much closer to our request. The remaining difference is because we mirror all of the memory below 4 GB, and because there is some rounding when setting up mirrored sections of memory.

If we rerun the efibootmgr command we'll see the values computed by the BIOS:

```
MirroredPercentageAbove4G: 11.22
MirrorMemoryBelow4GB: true
```

To check the amount of mirrored memory available, we can look in */proc*:

```
# grep -A 1 ", zone" zoneinfo
Node 0, zone    DMA
  pages free  3068
--
Node 0, zone DMA32
  pages free  414446
--
Node 0, zone   Normal
  pages free  0
--
Node 0, zone  Movable
  pages free  32450211
--
Node 1, zone   Normal
  pages free  4454637
--
Node 1, zone  Movable
  pages free  23573294
--
Node 2, zone   Normal
  pages free  4441648
--
Node 2, zone  Movable
  pages free  23535425
--
Node 3, zone   Normal
  pages free  3890024
--
Node 3, zone  Movable
  pages free  23579217
```

At this point an internal detail of the `kernelcore=mirror kernel` parameter becomes evident. The simple

In the kernel's view, we have two kinds of memory: *fixed* and *movable*. The kernel will only ever allocate its own data structures in *fixed* memory because the kernel allocators have no way to reclaim an allocation if the allocated memory is about to be removed. *Movable* memory is used for user processes since we can use page swapping mechanisms to trade out pages from applications.

The `kernelcore=mirror` option simply tells the kernel that all the mirrored ranges of memory are *fixed*, and all the nonmirrored ranges are *movable*. Thus all kernel allocations will be made from fixed (mirrored) memory.

It's important to note that because system memory is divided into these two distinct resource pools and there is no way to change the distribution of mirrored memory and nonmirrored memory at runtime, system performance can suffer if the system runs out of either type of memory. If the kernel runs out of mirrored memory, the system can experience operations failures, and if application memory runs out, paging operations can begin to degrade system performance as well.

TUNING

That makes the obvious question when using mirrored memory be, "How much should I configure?" The answer is the same as it is for every other performance tuning question in computer science: "It depends."

The exact amount of mirrored memory required depends on the server's workload. As an example, workloads that allocate almost all memory to applications, like memory databases or high performance computing, will run perfectly well with very low percentages of mirrored memory. Conversely, workloads that use large amounts of OS memory to perform basic kernel operations, like you might find on a file server, can suffer from operations failures and poor performance if you do not allocate enough mirrored memory for the kernel to operate normally.

There is a fixed lower bound of around 2% mirrored memory, below which the kernel cannot work at all. (The kernel needs 1.56% mirrored memory just for structures to manage the rest of memory.) The highest possible percentage of mirrored memory is 50%. Remember that the percentage is applied to the total amount of memory before mirroring, so 50% means that we split all memory into the primary and secondary copies of mirrored memory. If you find that your workload needs close to 50% mirrored memory, then you should probably just the *Full CH mirroring* mode in the BIOS setup instead of partial mirroring.

TESTING

We can use the ACPI EINJ mechanism to inject errors into mirrored memory and confirm that they are handled as expected. The full details of this testing is beyond the scope of this document, but the overall flow looks something like this:

- Enable error injection in BIOS settings. (This is under the *EDKII > Advanced > System Event Log > WHEA Settings menu*.)
- Write a test driver that allocates some mirrored kernel memory and publishes the physical address.
- Inject an uncorrected memory error at the physical address using the Linux error injection interface in */sys/kernel/debug/apei/einj*. Refer to *Documentation/acpi/apei/einj.txt* in the Linux kernel source for an explanation of this interface.

primary or secondary copy in physical memory. If the error is in the primary copy, then the error will trigger and the flow will proceed as described in the *Mechanics of mirroring* section. The error gets corrected in the primary copy using the data from the secondary, since EINJ only injects transient errors.

However if the error is in the secondary copy, no error gets triggered because the read will get good data from the primary copy of the mirror. In this case the driver should write new data to the address to clear the error and retry with another address.

CONCLUSION

Partial memory mirroring for kernel allocations can provide increased reliability and uptime for single servers or single point-of-failure systems running Linux kernel version 4.6+. By mirroring only the memory that the kernel needs, partial memory mirroring is also more cost effective for some workloads than mirroring all of a system's memory. You can use the kernel parameters, configuration options, and tools to enable, test, and fine-tune partial memory mirroring to best fit your requirements.

FOOTNOTES

1. https://software.intel.com/en-us/articles/address-range-partial-memory-mirroring