A Linux-based tool for hardware bring up, Linux development, and manufacturing

T. Venton
M. Miller
R. Kalla
A. Blanchard

In this paper we describe Bare Metal Linux (BML), a cut-down version of Linux® 2.6 that requires no firmware, has an in-memory root file system, and runs without a virtualization layer. We designed and implemented BML in order to accelerate the bring up of POWER5™-based systems. The use of BML allows testing and validation of the POWER5-based system to be conducted in parallel with the standard path, which involves the bring up of a hypervisor, the partition firmware, and the operating system. BML, which has fast boot times and can be modified quickly, is used in fault detection during chip manufacturing, POWER5 chip verification, system-board verification, and benchmarking for performance. BML is also used to reproduce and resolve problems in Linux.

Bare Metal Linux (BML), a tool that we implemented to accelerate the bring up of POWER5*[1]-based systems, is described in this paper. The POWER5 processor, released in 2004, is the latest version of the POWER architecture from IBM (POWER is a RISC [reduced instruction set computer] architecture). The POWER5 design implements two-way simultaneous multithreading (SMT) on each of the two processor cores on the chip. SMT combines multithreading, which consists of multiple threads utilizing the same processor in one-at-a-time fashion, with the simultaneous use of the multiple execution units present in a modern processor. In the two-thread SMT architecture of POWER5, the execution units not needed by the first thread are available to the second thread in the same clock cycle.

Non-Uniform Memory Access (NUMA) refers to a computer memory architecture where the memory access time depends on the memory location. Specifically, access to local memory is faster than nonlocal memory. For increased efficiency the operating system must incorporate in its algorithms knowledge about NUMA, such as the ratio of access times to local and remote memories. Although POWER5 systems, which contain multiple memory controllers distributed throughout the machine, are not NUMA in the classical sense (remote memory

latency is very close to local memory latency), they still benefit from NUMA-aware scheduling.

When a new system is designed, it is necessary to put the hardware through a series of tests to verify that it functions as expected. Booting a general-purpose operating system is a complex exercise requiring hardware errors to be addressed, initializations to be set up correctly, and firmware to be functional before operating-system testing can commence. This bring-up process is usually done in stages, incrementally increasing the scope and coverage of the hardware tested.

Typically the bring up of a processor chip begins at wafer test, when test patterns are run on the wafer to detect any circuits that are not working correctly. After good test sites (on the wafer) have been identified, the chips are diced and mounted on substrates to form modules. The bring up then continues on these modules by mounting them in test fixtures, which provide the system environment. At this point the chips execute functional code sequences intended to verify proper instruction execution. These low-level tests consist of the following steps: (1) generate a stream of instructions, initial conditions, and expected results, (2) load and run the generated stream and save the results, and (3) compare these results to the expected results.

After the low-level tests have verified basic processor functions, more complex exercisers are then used to verify functions in the processor and memory subsystems. After this stage is completed, the verification process continues at the operating-system level. Support is provided to execute larger, more complex programs that require a file system for storing code, data, and supporting tools. At this point support for I/O devices is needed. Whereas it is fairly straightforward to develop and employ low-level exercisers for processor core and memory, when I/O is required, then the flexibility of a general-purpose operating system is typically needed.

The POWER5 system predecessor, using POWER4* processors,[2] supported two methods of booting an operating system. In the first method the operating system is booted directly on the hardware by firmware. In the second method the firmware loads a hypervisor and, at the same time, the system resources are allocated to a number of hypervisor-controlled partitions. Each partition behaves as a separate virtual computer, on which an operating system may be loaded.

The POWER5 hypervisor provides additional virtualization capabilities compared to those for POWER4 systems, and in particular a high degree of resiliency to runtime errors. Supporting such advanced functions necessarily involves complexity. Although the architecture of the hypervisor has been designed to support additional virtual resources, these advanced functions were integrated throughout the hypervisor and the supporting firmware. As a result, POWER5 firmware no longer supports booting the operating system directly on the hardware.

This presented a problem during the bring-up phase of system development, when the hardware and the software were brought together. At this stage, the I/O had very limited testing. Without a general-purpose operating system running, the POWER5 bring-up team could not run operating system-based exercisers on the new hardware. Yet, the hypervisor had to be functional before an operating system could be booted. Complex error recovery during early bring up was not desirable because it had the potential to hide errors from the debug engineers. For these reasons relying on the hypervisor for the bring up was ruled out.

Our solution was to create Bare Metal Linux (BML) by modifying the Linux** kernel to run directly on the hardware, leaving out both the hypervisor and the firmware layer. By not including any error recovery and by supporting only simple configurations, we kept the BML code simple. For example, the code that configured the I/O subsystem was only a few pages long. Handling complex configurations was deferred until after the hypervisor became operational on the system. By eliminating many code layers, along with the associated initialization delays, we achieved rapid boot times for BML, an important feature of the tool. ***Figure 1*** illustrates the POWER5 bring-up process using BML.

Related work

There are many firmware solutions targeted at CPU and system bring up. IBM has the PIBS[3] (PowerPC* Initialization and Boot Software) firmware stack, and there are other products that offer similar

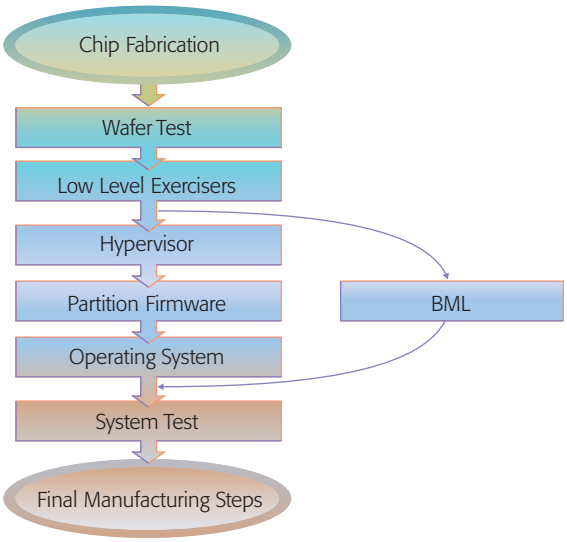

Figure 1
POWER5 bring-up process using BML

functionality.[4] By presenting a common layer between the hardware and the software, they isolate many of the platform details from the operating system. A similar solution is provided by the Linux-based LinuxBIOS,[5] which serves as a firmware stack that initializes the hardware and boots a second-stage operating system.

In addition, many embedded boards are brought up without firmware, and instead a minimal boot loader is used to load a Linux kernel.[6] In this case the boot-loader does some low-level initialization and loads the kernel, which is then responsible for initializing the rest of the system.

Other methods for accelerating system bring up are based on simulation and include the technique known as virtual power on.[7] This technique, used extensively by the IBM zSeries* development team, employs simulation to debug firmware and resolve operating-system bring-up issues before hardware is available.

Firmware solutions and LinuxBIOS still require a kernel to be loaded and control to be transferred to this kernel. This means there are two code bases to work with as well as interface constraints between them. BML on the other hand was developed as a single code base, which made it easy to develop and debug incrementally. The virtual power-on concept is complementary and focused on software.

The rest of the paper is organized as follows. In the next section, "BML design," we describe the key changes we made to the Linux kernel. In the following section, "Experience," we describe how BML performed as a tool for POWER5 bring up. We also describe several additional applications that BML was found to be helpful with, such as chip manufacturing and Linux development. The section "Conclusion" contains some final comments.

BML DESIGN

A number of changes were needed to run the Linux kernel without a firmware stack. We describe some of the key changes we made in order to adapt the Linux kernel to our environment in the following subsections.

Hardware discovery

IBM POWER5 systems use Open Firmware[8] to load the operating system, which then would normally interact with this firmware during early boot and initialization. Among other things, Open Firmware provides a method of device and system discovery in which various system parameters can be determined, such as the amount of system memory, the number of CPUs, and the location of PCI (Peripheral Component Interconnect) host bridges. PCI is laid out in a tree structure with the host bridge at the root. The host bridge is the point at which the CPU interfaces with the PCI subsystem.

In the BML environment there is no Open Firmware code to provide this information. This, however, presents less of a problem than it seems at first. Because Linux is highly portable, architecture-specific system interfaces such as Open Firmware and ACPI[9] (an open industry specification for configuration and power management) cannot be used throughout the operating system. Instead, the architecture-specific system interfaces are isolated in small, easy-to-modify sections of code. A simple interface was provided to accept the critical system information that is normally gathered by means of Open Firmware. This involved loading a number of parameter values into general-purpose registers before starting the kernel initialization thread. The parameters were as follows:

- *Number of CPUs*—POWER5 systems can have up to 128 threads. A 128-bit mask of the available CPU threads was passed in by means of a pair of registers.

- *Memory size*—The memory size in gigabytes was passed in by means of a register. In the interest of simplicity, memory was required to be contiguous starting at address zero.
- *Memory layout*—The POWER5 processor has an on-chip memory controller. On machines with multiple chips this means there are multiple memory controllers with memory distributed among them. Although not required for initial bring up, passing in the memory layout allowed Linux to optimize the use of all memory controllers. This improved test coverage, as more of the system could be exercised. The memory layout was specified as an array of bytes in a single register, each byte containing the size of the memory in gigabytes behind that memory controller.
- *Number and location of I/O bridges*—POWER5 systems can have multiple I/O bridges; the number and location can vary from system to system. A method of specifying which I/O bridges exist in a given system was required. Again a bit mask was used, each bit representing the availability of a possible bridge connection.

In-memory root file system

The Linux 2.6 kernel supports a memory-based file system initialized from a cpio[10] archive. The purpose of this mechanism is to move data files and complex initialization functions out of the kernel and into user space. The mount of the root file system is performed from the memory-based file system, which is populated with only a few device files and directories.

Early in the bring up, there were some programs we needed to run before writing the PCI probe code ("probe" is used as shorthand for "probing for hardware discovery"). By removing the call to mount the root file system device, we were able to populate the entire file system in memory and run the programs from there. This turned out to be very useful, as we were able to load a Debian**[11] distribution-based root file system that, because it was operating without a hard drive, did not risk hard-drive corruption during reboots and made our code easily transportable to different systems in the laboratory. Recent additions to the kernel allow us to load the cpio archive without any changes to the kernel source. These additions also allow the file system to be loaded separately as an initrd[12] image, removing the need to rebuild the kernel image to change the file system.

I/O probe

Most devices on POWER5 systems connect by means of PCI-X**[13] buses. Normally Open Firmware would probe, assign address spaces to, and initialize all PCI and PCI-X devices. This procedure is relatively complex and requires setting up registers in the entire hierarchy of devices, including host bridges, PCI-PCI bridges, and PCI devices.

In Linux 2.4 for 64-bit PowerPC architecture, the PCI code relies on Open Firmware to perform this task. In the BML environment this is not possible because the firmware layer does not exist. It is not necessary to perform this initialization, however, because Linux itself is capable of performing the probe and initialization. This functionality is required because a number of architectures, such as some embedded 32-bit PowerPC platforms, have no firmware layer and therefore have to provide full support for PCI probe in the operating system.

In Linux 2.6 for 64-bit PowerPC architecture, the PCI code was designed to support the generic Linux code to work with both methods, either relying on Open Firmware or allowing the Linux kernel to do the probe and initialization.

Parallelizing the I/O probe

During bring up of both hardware and operating-system software, many reboots are often required. Hardware failures and operating-system bugs can be hard to isolate, and repetitive attempts are often the only method of progress. For example, the Linux out-of-memory (OOM) killer bug[14] (discussed later in this paper) took five reboots before the problem was isolated.

Some problems require many attempts, in the tens or even hundreds. The time it takes to recreate a problem is a critical factor in how fast the bug can be found and fixed. Attempts to recreate the problem often require rebooting the operating system—to achieve repeatability, to cope with the loss of machine (hardware or software) state, or to add instrumentation as new hypotheses are formulated. The time it takes to reboot a machine can be a significant portion of the time it takes to recreate the problem. On machines with large amounts of I/O (e.g., disk and network adapters) reboot times

increase as firmware and operating-system software perform initializations and probe for more devices. During the bring up of the larger POWER5 systems, when it became clear that the lengthy boot times were an impediment to making progress, a study was undertaken to determine where the time was spent. The largest contributor to the boot time on a large POWER5 machine with 500 disks and 360-gigabit network interfaces was found to be the I/O probe code in the Linux kernel.

The Linux 2.6 kernel serializes the probe of all PCI and SCSI[15] (small computer system interface) devices because adding the host to the SCSI subsystem is performed during the adapter probe callback from the PCI subsystem. In this scheme the machine could spend a great deal of time waiting for a single, possibly non-existent hardware device to respond. Because on a large SMP (symmetric multiprocessing) machine only one CPU thread out of a large number is utilized while the remaining resources are effectively idle, parallelizing the probe function was needed.

Because in a parallelized execution of the I/O probe devices can come up in any order, initialization scripts that rely on a specific order may no longer work, and thus persistent device naming is important in this environment. Persistent device naming is an operating-system feature in which devices are identified consistently from boot to boot, even if the configuration is changed. For example, if a network card is moved from one IO slot to another, the operating-system user sees it as the same device. In the Linux 2.6 kernel, the device-naming policy was moved into user space by the udev[16] program using the sysfs file system.

Another difficulty with parallelizing the I/O probe is that there are some I/O devices that must be probed during boot to allow the machine to be usable. This often includes a network device as well as a disk containing the operating system. One possible solution to this problem is to store the location of these devices in NVRAM (non-volatile random access memory) and query it at boot time. The operating system can then skip all I/O devices that are not required.

In a bring-up environment in which hardware is changing often, keeping NVRAM up to date is time consuming and error prone. Instead, a small number of I/O slots are specified to be probed at boot, and the rest of the slots are to be probed under the control of the user.

The existing boot-time I/O probe code in the Linux 2.6 kernel could have been modified to probe all I/O at boot time with multiple threads. However, there are a number of advantages to performing the probe after the boot. After user space is initialized, a debug environment that is much richer in features is available. For example, the user can log in and selectively activate the I/O devices required for the test. Some tests can be initiated immediately, such as CPU core tests, while the I/O is being probed in the background. Also the user can selectively activate the I/O devices required for the test and avoid the time-intensive alternatives of either physically removing these I/O devices from the machine or waiting for them to be probed and initialized. Finally, existing user-space tools can be used to identify failing I/O adapters, such as the ethtool[17] utility that can cause indicator lights to blink on supported adapters.

The POWER5 architecture allows for multiple PCI-X host bridges, and this offers a useful and convenient way to partition the I/O adapters into those that are required at boot and those that are not. This also ties in well with the Linux 2.6 PCI probe architecture, in which the probe iterates over each host bridge, performing an exhaustive search of the buses "below" it (controlled by it).

Consequently, the BML kernel was modified to probe at boot a single host bridge. All I/O devices required for the boot are placed behind this host bridge, and any subsequent I/O operations are spread across the remaining host bridges. During the boot, devices are still probed serially, but because the number of devices involved is small—typically one network device and a few SCSI disks—this step is completed quickly.

In order to support a parallel probe scheme in the BML kernel, we broke the problem into a number of steps. As the first step, when the machine completes booting and the user initiates a parallel probe operation, a kernel thread is created for each group of host bridges. Each thread probes the buses below it in parallel and initiates a built-in self-test (BIST) of all adapters. At this stage no adapters are initialized.

Upon completion of the first step, another kernel thread is created for each host bridge. Its job is to add all the devices found in step 1, using the existing Linux 2.6 hot-plug infrastructure. At this stage the SCSI host is registered with the SCSI subsystem, but the probing of devices is suppressed.

The sysfs file system can selectively probe each SCSI device. The BML parallel probe uses this capability in its final step in discovering SCSI disks.

Compilation environment

One of the major advantages of using the Linux kernel is the full availability of its source code for both inspection and modification. When chasing both hardware and software bugs, rapid turnaround times allow hypotheses to be tested quickly, which ultimately leads to bugs being found faster. Experience shows that a compilation environment that is responsive and easy to use is an important asset. In the course of developing BML we adapted several existing technologies to our compilation environment.

A critical element of the compilation environment that we incorporated is the distcc package, written by Martin Pool.[18] According to its Web site, distcc is a "program to distribute builds of C, C++, Objective C or Objective C++ code across several machines on a network." While the original setup consisted of a single CPU POWER4 partition, it was quickly augmented with a number of retired four-way POWER3* systems. Due to ease of configuration and transparent failover of distcc, all users of the compilation environment were able to take advantage of the extra processing capacity. Furthermore, because the standard BML file system image included the distcc server, as POWER5 systems were brought up in the laboratory, they could be incorporated into the distcc build farm easily.

Although the core BML team was small (three people), there was still a need for a revision control system. The choice of this revision control system was first based on the requirement to track the rapidly changing Linux 2.5 (a development release) kernel tree. This served the double purpose of finding new Linux bugs early (as new kernels were regularly tested in the laboratory) and incorporating new features into BML, as they were merged into the main tree. In addition, there was the requirement to merge changes back into the main Linux kernel tree. Because the team was small, it was important to get the changes merged into the main tree—anything merged into the main tree was no longer a maintenance load.

Traditional tools such as CVS[19] did not fit in well with this development model. Andrew Morton, head of Linux 2.6 kernel development, has developed a simple yet powerful set of scripts[20] that have proven to work very well instead.

The final piece in the environment was the ccache package, written by Andrew Tridgell.[21] Because ccache wraps the invocation of the compiler and intelligently caches compiled objects, the compilation time was accelerated even more.

EXPERIENCE

We describe in this section our experience using BML for POWER5 bring up. We also discuss the role BML played in investigating various problems we encountered and the impact BML had on Linux development.

Figure 2 shows the BML early test machine. The system represents the largest single SMP system in IBM's history, consisting of a 32-CPU POWER5 processor, 128 threads, and 32 I/O drawers (it is a prototype of the IBM eServer* p5 595 system). The setup consists of four racks. The second rack from the right contains the CPU and its power supply. The remaining three racks contain the I/O drawers with disks and network adapters.

POWER5 bring up

We created BML by modifying the Linux kernel to run directly on the hardware, leaving out both the hypervisor and the firmware layer. By leaving out error recovery and by supporting only simple configurations, we kept the BML code simple and achieved rapid boot times. The role BML played in POWER5 bring up went beyond what we initially planned and extended to system-board verification, benchmarking for performance, and chip manufacturing.

Processor verification

BML's flexibility, which enabled us to cope with a multitude of problems, made it the bring-up vehicle of choice throughout the POWER5 program. It was the first operating system booted on each of the

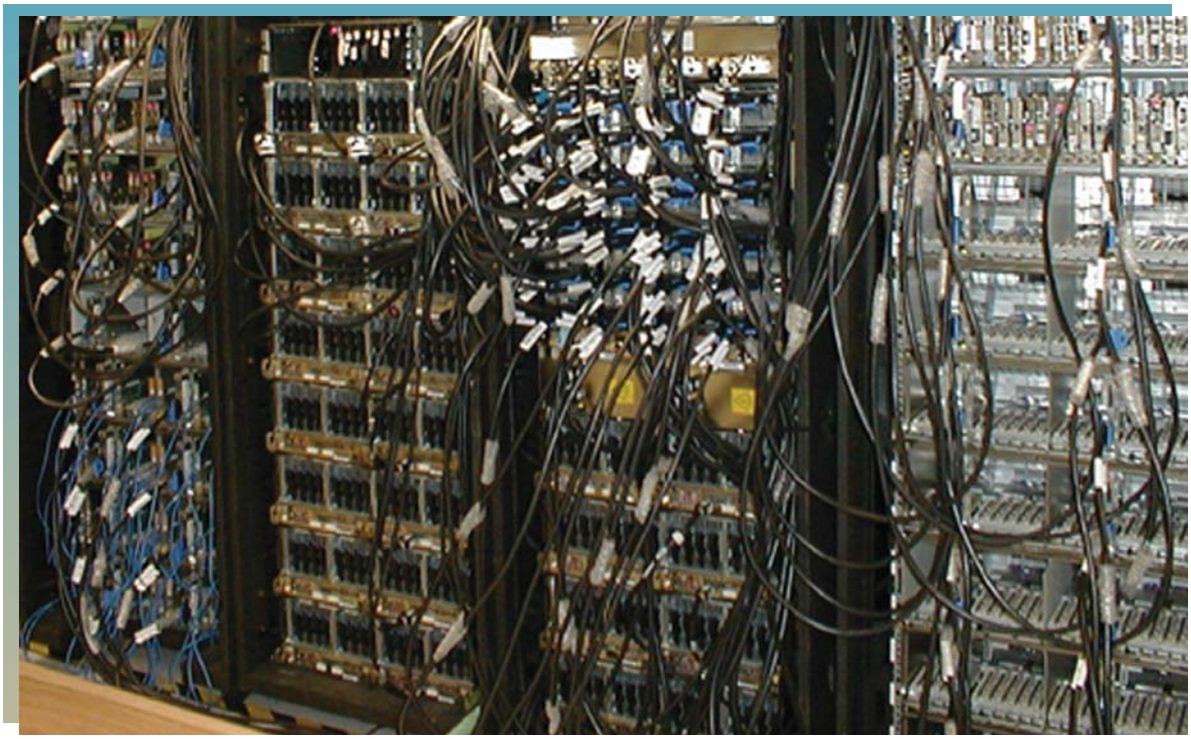

Figure 2
BML early test machine

POWER5-based systems. This was partly due to the amount of work required to bring up the hypervisor, but in many cases there was no immediate need for the hypervisor to be used in system testing. This allowed more hardware resources to be focused on hardware verification. In addition, it allowed the firmware and hypervisor teams time to work through issues encountered in their code while the hardware team could work through hardware testing and problem resolution in parallel. The hypervisor and firmware components, being part of a product shipped to customers, must be maintainable. This means the hardware has to be described to the firmware through configuration data. Because BML was not constrained in the same way, we were able to write temporary code in order to address a system configuration issue without regard to long-term maintainability.

BML is not only simple in design, but it has proven to be very flexible as well. During the POWER5 chip's initial bring up, access to memory was not possible. With several functional processors in hand, a team of engineers developed a method to reroute the local memory requests to a remote system's memory via the I/O interface. This was accomplished by changing the basic memory map layout of the system and the associated chip initializations. In addition, changes to the Linux kernel were needed to ensure all memory accesses were cache-inhibited. This capability alone saved us several weeks in bring up, as it allowed us time to work through several hardware configuration issues and a wide range of system and Linux issues. After access to system memory became available, Linux was booted on the system, and the standard system exercisers were running within a matter of hours, in large part due to the learning that occurred during the configuration using only a remote memory.

By using BML on the POWER5 chip, the developers uncovered several problems. Though these problems could have been discovered with an operating-system-plus-hypervisor configuration, finding them with BML had three significant advantages: (1) With BML these problems were discovered earlier in the bring-up process, thus allowing fixes to be fabricated in silicon sooner; (2) the rapid boot time allowed high debug productivity with BML; and (3)

recreating a problem was often quicker without the virtualization layers of the hypervisor.

Due to BML's fast boot times and quick turnaround time, it quickly became clear that the tool was usable outside the initial bring-up process. With a fully functional operating system at hand that was capable of booting quickly, the additional uses for BML were many.

System-board verification

System-board verification was one of the first tasks BML assisted with after the initial POWER5 system bring up. As each new system model became available to the laboratory teams, the board design and features that were enabled in that system needed to be verified. With a wide assortment of Linux-supported I/O devices, we were able to fully configure most POWER5 systems and verify that a large portion of the board functions were operational. In the cases in which failures were observed, because BML was the only code running on the system, we were able to quickly determine the source of the problem.

Benchmarking for performance

The successful bring up of a POWER5 system results in a stable system. The next major hurdle for the POWER5 team is evaluating the hardware performance through benchmarking. Again BML seemed to be ideally suited for the task. When performance problems were identified, using hardware performance counters, hardware traces, and Linux tools such as Oprofile,[22] most of the problems were quickly corrected by changes to POWER5 initializations or changes to Linux itself. A baseline performance was first obtained by running a suite of benchmarks. These included such standard benchmarks as SDET,[23] SPECjbb**,[24] SPEC CPU2000,[25] STREAM,[26] and Netperf.[27] In addition, several simple tests were written to measure other performance characteristics of the system. At first several of the benchmarks performed more slowly than expected. Using an iterative approach, tuning knobs were adjusted, and the expected performance was eventually achieved. The stress of these benchmarks also uncovered several time-out values that needed adjusting in order to avoid flagging errors during normal operation.

During POWER5 development many workloads were simulated and SMT gains projected. However, it was not until the chips arrived at the laboratory that an accurate performance evaluation through measurements could be performed. Although simple applications can be modified to run without a general-purpose operating system, many applications such as SPECjbb (a Java** application benchmark modeling a three-tier server-side application) are complex and rely on operating-system facilities. Because BML was booting on all POWER5 platforms early, we used it to quantify the SMT gains. To a user-space application the BML environment is almost impossible to distinguish from the standard firmware and hypervisor environment. We found all standard applications would run without changes, including STREAM, SPEC** CPU2000, SPECjbb, and others. BML was used to tune the dynamic resource-balancing logic in the POWER5 processor core. Following this tuning several benchmarks showed significant improvement in performance. *Figure 3* illustrates the improvement in the relative throughput using SMT over a single-thread-per-processor configuration. The results shown were obtained with the SPECjbb benchmark on a 32-CPU IBM eServer p5 595 system. The SPECjbb benchmark uses a "warehouse" as its basic unit of independent work. Throughput rises as warehouses are added until the limit of the machine is reached. With SMT disabled, maximum throughput is reached at 32 warehouses, one warehouse per CPU. When SMT is enabled, throughput continues to increase, although at a slower rate, until all 64 threads are active. The graph shows close to a 40 percent improvement in relative throughput.

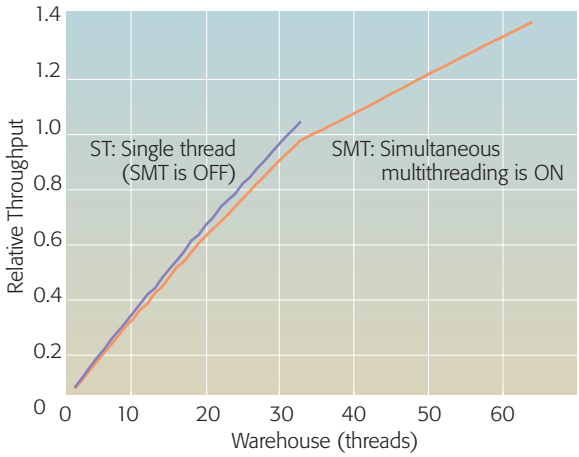

Figure 3
Relative throughput improvement with SMT (SPECjbb benchmark, IBM eServer p5 595)

After most of the tuning task was completed, a large number of "what-if" questions arose. For example, the POWER5 processor has several power-saving features that needed to be tested. Almost all these features were controllable by changing initializations in the POWER5 chip. By comparing results against the baseline measurements, we were able to verify that performance was not negatively impacted by these features.

Chip manufacturing

While bring up and verification of the POWER5 chip was in full swing at various laboratories inside IBM, the manufacturing team was researching how to best screen each chip for faults. This screening is absolutely critical to the success of POWER5 products. In addition to detecting faults, the data collected on each chip during the screening process are used to determine the POWER5 system that the chip is best suited for, based on its power and cooling requirements. BML appeared to be the perfect tool for screening the POWER5 chip. First, the major exerciser suite used in testing the processor had already been ported to Linux and had been in use for several months in the bring-up laboratories (thus no porting of the exerciser code was required). In addition, because BML boot times are low, the screening process is more efficient (any time spent booting directly subtracts from the time available for testing the chip). The manufacturing team quickly adopted BML as their screening tool, which is now used to screen every POWER5 chip produced.

Impact on Linux development

BML was instrumental in identifying and resolving a number of Linux kernel problems.

Bug in the out-of-memory killer

When the machine is completely out of memory, one of the active tasks has to be terminated (killed) in order to free its memory allocation, so that the rest of the system can operate. In Linux this is the responsibility of the OOM killer.

In the past there had been reports of the OOM killer misbehaving and killing the entire system. However, these reports were few, and the problem could not be recreated in the laboratory. Further, there existed code specifically designed to avoid the problem. During a stress test using BML on a 64-CPU machine with only a few gigabytes of memory, we discovered that this failure scenario could be replicated in less than 1 minute.

Armed with an easily reproducible test case, the SMP race (a case in which multiple processors access data without required serialization or locking) in the code was found, and the problem was fixed. Whereas the bug would seldom surface in smaller machines, the large 64-CPU POWER5 machine displayed it reliably.

Random driver scalability

Linux has a random number generator (random driver) whose output is used by various kernel functions. In order to increase the randomness of this component, some random external events are tracked, and this information is stored in a so-called entropy (randomness) pool, for use by the algorithm that produces the random numbers.

During the stress tests for measuring maximum I/O bandwidth during POWER5 bring up, we spotted a serious drop in network I/O performance on our large machine. We traced this performance deterioration to the use of a global lock taken when device interrupts associated with the use of external events by the random number generator occur. We solved the problem by a change to the Linux 2.6 kernel; the random number generation algorithm now recognizes when there is sufficient randomness in the entropy pool, in which case no additional external random events are used, and thus the use of the global lock is decreased.

SCSI-probe race conditions

During the development of the parallel probe feature in BML, we discovered a number of race conditions in the SCSI probe code.[28] In one test, 500 SCSI disks were probed in parallel, which took about 10 seconds to complete. This environment provided a good platform for testing locking performance in the Linux kernel.

SCSI-driver error handling

Device drivers provide an interface for connecting hardware devices to the operating system. Although these device drivers should handle malfunctioning hardware gracefully, often the error paths associated with these malfunctions are not well tested.

A laboratory environment like ours has a mix of hardware components, some of which may not

always function correctly. When large machines are built, it is possible to end up with defective adapters or disk devices. For this reason, it is highly desirable to enable the remaining hardware in the system to be used until the defective hardware can be replaced. We discovered and fixed several deadlocks in the error paths of two SCSI host-adapter drivers.

Linux hot spots

While running various workloads, traces were taken of the system bus and analyzed to find and fix Linux kernel performance issues. For instance, traces were used to locate hot spots where many CPUs were accessing shared cache lines. Armed with this data, changes were made to the Linux kernel to improve the situation.

One example of such a change was a cache line that was being accessed frequently. By looking at a map of the Linux kernel, it was determined that a variable that was modified a lot by one CPU (`tb_last_stamp`) was in the same cache line as some read-only data used by each CPU at every timer interrupt. In the Linux 2.6 kernel release, the timer interrupt runs 1000 times a second, so on a 64-CPU 128-thread machine, the cache line was being fetched 128,000 times a second. After separating `tb_last_stamp` from the read-only data, these 128,000 cache-line transfers each second were removed completely. This performance oversight was corrected in the Linux 2.6.7 kernel.

Detecting interprocess communication errors

Tests designed to find CPU bugs often stress-test the underlying operating system. One such test utilizes the interprocess communication (IPC) mechanism. Repeated and closely spaced starts and stops of this test led to intense activity in the creation and deletion of IPC resources, which in turn exposed deadlocks in the IPC code.

SMT- and NUMA-aware scheduler

While the POWER5 bring-up work was in progress, the Linux community was working on enhancing the scheduler to make it SMT- and NUMA-aware. Nick Piggin was working on an approach he called `sched domains`,[29] which provides support for platforms with SMT and NUMA features. The POWER5 architecture provided a perfect testbed because it has both features.

With BML running on the largest POWER5 machines, the prototypes of `sched domains` were tested. BML enabled us to obtain results quickly and allowed us to test many scheduling options. Based on our testing, the scheduler was accepted in the Linux 2.6.7 kernel. Had we waited until the firmware and the hypervisor stack were operational, we would have been unable to provide timely feedback to the Linux community.

CONCLUSION

During the POWER5 chip bring up, BML proved to be an efficient and flexible tool. Using BML in the manufacturing of POWER5 systems has benefited both the POWER5 bring-up effort and Linux development. BML enabled the operating system-level testing to start early, thus reducing the time to market of POWER5 systems. Stress testing the Linux kernel on a leading-edge system led to the resolution of some bugs and increased its robustness. We expect that BML will continue to be used in the bring up of future systems.

CITED REFERENCES AND NOTES

1. R. Kalla, B. Sinharoy, and J. Tendler, "Simultaneous Multithreading Implementation in POWER5," Presented at Hot Chips 15, A Symposium on High Performance Chips, IEEE Computer Society (2003), http://www.hotchips.org/archive/hc15/pdf/11.ibm.pdf.
2. J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, "POWER4 System Microarchitecture," *IBM Journal of Research and Development* **46**, No. 1, 5–26 (January 2002).
3. *Download the IBM PowerPC 750GX-750FX Evaluation Kit*, developerWorks, IBM Corporation (2004), http://www.ibm.com/developerworks/power/ppc750gxfx/.
4. The Open Firmware Source, FirmWorks, http://www.firmworks.com/.
5. The LinuxBIOS Project, LinuxBIOS, Inc., http://www.linuxbios.org/.
6. A. K. Santhanam and V. Kulkarni, *Linux System Development on an Embedded Device*, developerWorks, IBM Corporation (2002), http://www.ibm.com/developerworks/linux/library/1-embdev.html.
7. K.-D. Schubert, E. C. McCain, H. Pape, K. Rebmann, P. M. West, and R. Winkelmann, "Accelerating System Integration by Enhancing Hardware, Firmware, and Co-Simulation", *IBM Journal of Research and Development* **48**, Nos. 3/4, 569–582 (2004).
8. IEEE 1275 Open Firmware Home Page, IEEE Open Firmware Working Group, http://playground.sun.com/1275/.

9. ACPI—Advanced Configuration and Power Interface, http://www.acpi.info/.
10. *Cpio*, Free Software Foundation, Inc., http://www.gnu.org/software/cpio/cpio.html.
11. *Debian—The Universal Operating System*, Software in the Public Interest, Inc., http://www.debian.org/.
12. Linux-Kernel Archive: initramfs buffer spec—third draft, Linux-Kernel Archive (January 13, 2002), http://www.uwsg.iu.edu/hypermail/linux/kernel/0201.1/1605.html.
13. *PCI-SIG*, PCI-SIG Corporation, http://www.pcisig.com/home.
14. A. Morton, *Re: 2.5.74-mm1*, Note posted on the Linux-Kernel mailing list (July 5, 2003), http://marc.theaimsgroup.com/?l=linux-kernel&m=105736781923269&w=2.
15. *SCSI Storage Interfaces*, T10 Technical Committee on SCSI Storage Interfaces, International Committee on Information Technology Standards, http://www.t10.org/.
16. G. Kroah-Hartman, "udev—A Userspace Implementation of devfs," *Proceedings of the Linux Symposium*, Ottawa, Ontario, Canada (July 23–26, 2003), http://archive.linuxsymposium.org/ols2003/Proceedings/All-Reprints/Reprint-Kroah-Hartman-OLS2003.pdf.
17. *Project Info—gkernel*, Open Source Technology Group, http://sourceforge.net/projects/gkernel/.
18. M. Pool, *distcc: a Fast, Free Distributed C/C++ Compiler*, http://distcc.samba.org/.
19. Concurrent Versions System, https://www.cvshome.org/.
20. Patch scripts are available at http://www.zip.com.au/~akpm/linux/patches/.
21. A. Tridgell, *ccache*, http://ccache.samba.org/.
22. *Oprofile—A System Profiler for Linux*, Open Source Technology Group, http://oprofile.sourceforge.net/.
23. *057.SDET*, SPEC SDM Suite, Standard Performance Evaluation Corporation, http://www.spec.org/sdm91/#sdet.
24. *SPEC JBB2000*, Standard Performance Evaluation Corporation, http://www.spec.org/jbb2000/.
25. *SPEC CPU2000*, Standard Performance Evaluation Corporation, http://www.spec.org/cpu2000/.
26. J. D. McCalpin, STREAM: Sustainable Memory Bandwidth in High Performance Computers, http://www.cs.virginia.edu/stream/.
27. The Public Netperf Homepage, http://www.netperf.org/.
28. "Re: [PATCH] serialize bus scanning," Mailing list ARChives (MARC) (September 15, 2003), http://marc.theaimsgroup.com/?l=linux-scsi&m=106361030428383&w=2.
29. N. Piggin, *Status of Hyperthreading-Aware Scheduler*, Kernel Traffic (November 25, 2003), http://www.kerneltraffic.org/kernel-traffic/kt20031201_243.html#10.

Accepted for publication January 11, 2005.
Published online April 7, 2005.

Todd Venton
IBM Server and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (venton@us.ibm.com). Mr. Venton, an advisory engineer in the Central Electronic Complex Bring Up and Integration area, has worked for the past five years on bringing up and debugging POWER-based servers. He holds a B.Sc. degree in computer engineering from the University of Illinois at Urbana-Champaign. In his free time he enjoys water skiing.

Milton Miller
IBM Server and Technology Group, 11400 Burnet Road, Austin, Texas 78758 ((miltonm@us.ibm.com). Mr. Miller, an advisory engineer in the Global Server Integration area, has been involved for the past 14 years in bringing up and debugging POWER-based servers. He received an Outstanding Technical Achievement award for his contribution to the POWER4 system. He holds a B.Sc. degree in computer and electrical engineering from Purdue University at West Lafayette, Indiana. He holds one patent and has several additional patents pending. In his free time he is involved in the local amateur-radio community.

Ron Kalla
IBM Server and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (rkalla@us.ibm.com). Mr. Kalla is the lead engineer for IBM POWER5 systems specializing in processor core development. He has designed processor cores for S/370™, M68000, AS/400®, and RS/6000® machines and has contributed to many hardware bring-up and verification projects. Mr. Kalla holds 12 U.S. patents, has filed for 15 additional patents, and has published 15 articles in the IBM Technical Disclosure Bulletin. He received an IBM Outstanding Technical Achievement award for his contribution to the POWER4 system.

Anton Blanchard
IBM Linux Technology Center, 8 Brisbane Avenue, CA03 Barton ACT Australia 2600 (anton@au.ibm.com). Mr. Blanchard, one of the PowerPC 64-bit Linux kernel maintainers, has been involved in Linux development for over seven years. He holds a Bachelor of Engineering degree in computer systems from the University of Technology at Sydney. In his free time he enjoys traveling. ■