Technical White Paper
DATA CENTER

www.novell.com

Optimizing Linux* for Dual-Core AMD Opteron™ Processors

Novell

Table of Contents:

SUSE® Linux Enterprise and the AMD Opteron™ Platform

Dual Core technologies are the latest evolution in processor design. Novell® enterprise products are ready to take advantage of the state of the art AMD64 dual-core technology. Read this white paper to learn how to optimize your Linux applications with SUSE® Linux on the AMD Opteron™ platform.

Dual-processor, Dual-core and Multi-core Definitions

Dual-processor (DP) systems are those that contain two separate physical computer processors in the same chassis. In dual-processor systems, the two processors can either be located on the same motherboard or on separate boards.

Native dual-core technology refers to a **CPU** (central processing unit) that includes two complete execution cores per physical processor. It combines two processors and their **caches** and cache **controllers** onto a single **integrated circuit** (silicon chip). It is basically two processors residing side by side on the same die.

A dual-core processor has many advantages, especially for those looking to boost their system's multitasking computing power. Since each core has its own cache, the operating system has sufficient resources to handle intensive tasks in parallel, which noticeably improves multitasking. Dual-core technology is quite often referred to as multi-core technology, yet **multi-core** can also mean more than two separate processors (i.e., quad-core).

Dual-core Background

As said before, a dual-core processor is comprised of two independent execution cores. From a software perspective, a dual-core processor appears much the same as a traditional multiprocessor system with two single core chips (symmetric multiprocessing [SMP]), and all the basic multiprocessor optimization techniques apply. For the most part, single-core SMP-optimized applications will also be optimized for dual-core.

Dual-core differs from single-core SMP because memory access occurs per chip. This means that dual-core systems will have lower memory latency but also lower memory bandwidth. Each core on a dual-core AMD Opteron processor will have a 128 KB L1 cache (256 KB/chip) and a 1 MB L2 cache (2 MB/chip), but future versions may also provide a shared L3 cache, further improving performance. As the technology matures, operating systems might adopt process-scheduling algorithms that can take advantage of these differences to improve performance for each multiple-processor model.

Complete optimization for the dual-core processor requires both the operating system and applications running on the computer to support a technology called thread-level parallelism (TLP). TLP is the part of the OS or application that runs multiple threads simultaneously, where threads refer to the part of a program that can execute independently of other parts.

AMD Specific

Things are never quite as simple as they seem. As a first order approximation, a system with two cores on one die will behave similarly to a system with two separate processors.

However, to get the most performance out of the hardware, software might have to be aware of the the location of each core.

AMD Opteron processors with direct connect architecture, being a Non-Uniform Memory Access, or **NUMA**, system architecture, have the concept of a **node**, and **local** and **remote** memory. Each AMD Opteron processor is a node, and has local memory that it can access with the greatest speed. The local memory of one node is termed remote memory when accessed from another node. NUMA architectures allow software to transparently access local and remote memory; however, performance will be greatest when remote memory accesses are minimized.

The direct connect architecture of the AMD Opteron processor implements an independent memory controller on each node (chip), whether it be single- or dual-core. This means that both cores of a dual-core chip share memory resources and have the same local memory, while two single-core AMD Opteron processors each have their own memory resources.

NUMA Background

NUMA technology is a scalable memory architecture for multiprocessor systems that has long been used by the most powerful servers and supercomputers. A **NUMA API for Linux** (www.novell.com/collateral/4621437/4621437.pdf) has a good introduction to NUMA technology and terminology. This paper also details the interfaces and methods used to fine tune NUMA memory control on Linux, and will be referenced in regard to this subject later in this document.

NUMA Memory Access and Allocation in Linux Basics

Memory management of NUMA systems on Linux comprises a number of policies.

The default policy for program heap memory (also called anonymous memory: e.g., **malloc()** memory) is so called **first touch**.

First touch means memory is allocated from the local memory of the CPU that first touched that memory.

A thread which calls **malloc()** to allocate memory will not actually cause physical memory pages to be allocated, only reserved. Physical memory is only allocated when the program reads from or writes to that newly **malloc()**-ed data. This may occur from a different thread or CPU than the one that called **malloc()**!

Once memory is allocated to a node, it will not move even if the program changes to another CPU and all subsequent accesses to that memory are **'remote'**. For this reason it can be important for performance for thread and process migration between nodes to be minimized, and for memory placement to be carefully considered and tuned.

What Are Characteristics of Workloads that Would Be Helped by Using Dual-core CPUs?

For a workload to benefit from using dual-core CPUs, it must first and foremost be able to run in parallel (parallelizable). This is a fundamental requirement of any multiprocessor system.

Any workload can basically be divided into a serial component and a parallel component such that the parallel component may be executed and run by any number of processors while the serial component can only be run on a single processor. A workload is said to be able to run in parallel (parallelizable) if the parallel component dominates. A workload that can only run one processor has a zero-sized parallel component and is called **serial**.

A workload for a given system might be parallelized in one of two ways: either it is running an application that is specifically written to take advantage of a multiprocessor system, or it is running multiple independent/

semi-independent applications that may or may not be individually parallelized.

What Class of Parallel Workloads Can Specifically Take Advantage of Dual-core Systems?

Dual-core systems increase some specific types of computing resources in a system versus single-core chips. Workloads where the computing resource causing a bottleneck is one of those increased by dual-core chips will benefit from dual-core systems.

Remember that both cores of a Dual-Core AMD Opteron processor share the same memory controller. Dual-core chips will double the basic CPU processing capability of the system, and also double the cache size. They do not introduce more memory subsystem resources (memory bandwidth). Thus, the main applications that will benefit from dual-core processors are those that are compute bound and have spare memory bandwidth available for the second core.

How Can I Tell If My Workload or Application Would Benefit?

As outlined above, a good first-order estimation of the performance characteristics of moving to dual-core chips is simply doubling the number of single-core chips. If your workload is already taking good advantage of multiple processors, it will likely benefit from dual-core chips. However if the workload is very memory intensive, this test won't be very indicative.

If your application or workload has inherent limits in its parallelization (for example, if only four processes are run at once or if there is a hot lock hindering scalability), then adding more cores may not help even if your application is not at all memory intensive.

What Tools Exist to Tell Me Whether My Application Would Benefit?

On Linux, some basic monitoring programs like **vmstat** and **top** can be used to see whether your system is compute bound. If **idle** plus **iowait** combined (or just **idle**, if there is no **iowait** field) is approaching 0 for some or most of the time, then the system could be compute bound.'

These programs can also indicate the number of runnable (executable) processes in the system. For multiple cores to be of use, there need to be processes available to run on them. The caveat here is that some algorithms and applications tune themselves automatically to utilize the available CPUs. This is common in, but not limited to, high-performance computing (HPC) applications.

```
`vmstat 1` gives the following output after 3 seconds[*]:

procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 4  0      4  35552   2640  79708    0    0     0     0 1021   159 100  0  0  0
 4  0      4  35552   2640  79708    0    0     0     0 1005   113 100  0  0  0
 4  0      4  35552   2640  79708    0    0     0     0 1002    89 100  0  0  0
```

The first field indicates there were an average of four processes running, and the system is completely compute bound. This gives a preliminary indication that such a workload could utilize up to four cores (e.g., two dual-core processors), but will not benefit from more cores.

Using **OProfile** (http://oprofile.sourceforge.net/news/) to monitor memory controller activity can give an indication of whether or not an application is memory bound. The performance counters **CPU_CLK_UNHALTED** and **MEM_PAGE_ACCESS** can be used together to assess the amount of memory accesses per clock that the system is not idle.

Oprofile yields the following output when profiling a kernel compile on a two-way AMD Opteron processor-based system with single-core CPUs:

```
CPU: AMD64 processors, speed 2194.13 MHz (estimated)

Counted CPU_CLK_UNHALTED events (Cycles outside of halt state) with a unit mask of
0x00 (No unit mask) count 10000

Counted MEM_PAGE_ACCESS events (Memory controller page access) with a unit mask of
0x07 (multiple flags) count 1000

CPU_CLK_UNHALT...|MEM_PAGE_ACCES...|

  samples|      %|  samples|      %|

-----------------------------------

 20538586 64.3466    387282 14.5290 cc1

  5986767 18.7563   2141124 80.3248 vmlinux

  2574275  8.0651     65516  2.4578 libc-2.3.5.so

  ...      ...        ...    ...    ...
```

Note that **MEM_PAGE_ACCESS** events are in units of 1,000, while **CPU_CLK_UNHALTED** events are in units of 10,000, and the numbers are cumulative for all cores.

After doing some calculations, we can derive that this workload is causing one memory access per 119 work cycles. We know that compilation with gcc tends not to be memory bound, with lots of work being done from the cache, so we estimate that 119 is indicative of a workload that is not memory bound on this platform.

Next, a memory intensive workload, STREAM (www.cs.virginia.edu/stream/), is profiled:

```
CPU: AMD64 processors, speed 2194.13 MHz (estimated)

Counted CPU_CLK_UNHALTED events (Cycles outside of halt state) with a unit mask of
0x00 (No unit mask) count 10000

Counted MEM_PAGE_ACCESS events (Memory controller page access) with a unit mask of
0x07 (multiple flags) count 1000

CPU_CLK_UNHALT...|MEM_PAGE_ACCES...|

  samples|      %|  samples|      %|

-----------------------------------

   813682 75.7735    180562 23.9087 stream_amd

   219432 20.4344    574490 76.0696 vmlinux

   ...      ...        ...    ...    ...
```

STREAM is memory bound by design, and we can see it causing one memory access per 14 cycles. That is almost 10x more memory intensive than the kernel compile. We can estimate that 14 is indicative of a memory-bound application on this platform.

How well does dual-core performance correlate with the numbers we have?

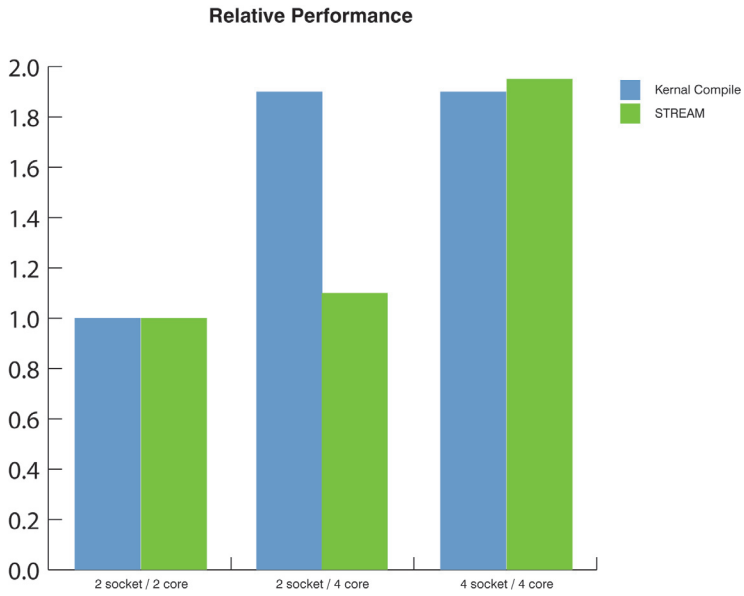

Figure 1. *Relative Dual-core Performance*

So we can see that moving to two dual-core CPUs has improved kernel compilation times by a factor of 1.91, which is as good as moving to four single-core CPUs. On the other hand, **STREAM** performance was basically unchanged when moving to dual-core CPUs, in contrast to a very significant gain with the move to four single-core CPUs (due to their greater memory resources).

The numbers '119' and '14' will vary from system to system, so a good strategy is to get two baseline results from known workloads such as **STREAM** and **gcc** against which you can compare your workload.

How Can Applications Be Tuned for Dual-core?

Many SMP-capable applications should see a performance improvement when moving from single- to dual-core processors, without any specific tuning.

The most basic thing to keep in mind is to ensure good data access patterns and thereby reduce the memory controller load. This kind of optimization is usually applicable to all software.

Specifically, tuning an application for dual-core systems will require an understanding of the performance differences involved with running tasks on the same core, the other core, or a different physical chip.

Tasks that are independent of one another in terms of what memory they operate on and their level of synchronization or communication are ideal candidates to place on different physical chips. This will maximize utilization of memory resources.

Tasks that are closely coupled, operating on the same memory or doing a lot of synchronization or communication should be placed together, either on each core of the same chip or both on the same core.

Putting two or more closely coupled tasks on the same core should be carefully weighed against the possible negative impact of allowing other cores to go idle. If there are two pairs of such tasks running on a four-core system, the best performance will usually be with each of the four tasks running on a single core. In the case of four pairs of tasks on the same four-core system, having each pair of tasks together on their own core will usually be best.

Sub-problems which weren't worth parallelizing on multiprocessor AMD Opteron processor-based systems, due to the cost of synchronization or remote memory accesses, may be worth parallelizing if threads are bound to the cores of a single physical chip, due to the lower latency of local memory access from both cores on a chip.

Explicitly binding processes or threads to CPUs and memory to nodes might be useful in getting the full potential from a dual-core system.

What Tuning Is Possible to Control Performance on a System Using Dual-core CPUs?

In Linux, the CPU scheduler behavior itself is not tunable at runtime. The philosophy is that performance should be good enough in all cases that specific tuning isn't needed. That said, there are cases where the scheduler won't make optimal choices. These can be handled by explicit binding of tasks to CPUs.

Explicit CPU or memory binding may help, although this reduces the effectiveness of the CPU scheduler's multiprocessor balancer. It would be easy to reduce performance unless the system behavior is very well understood.

How Does the Linux 2.6 CPU Scheduler Handle Dual-core CPUs?

Exact behavior differs depending on the version of Linux used. Generally, behavior is better in more recent kernels.

There are a number of "behavioral goals" shared by all multiprocessor systems, including those with dual-core chips (even a system with one dual-core chip is a multiprocessor):

Utilize the Maximum Amount of CPU Resources Available

In a dual-core system it is possible for one core to be completely idle while the other core has two processes running on it. Actually, only one is running at any given time, with one "runnable" (executable).

Linux's scheduler attempts to keep all CPUs busy by moving runnable processes to idle CPUs and keeping the number of running tasks on each core in reasonable balance.

Minimize the Amount of Process Movement Between CPUs

When a process runs on a CPU, it builds up a footprint in cache of its most frequently used memory. If a task has to be moved to another CPU, it loses the benefit of this cache and must refill from memory.

On NUMA systems like AMD Opteron processor-based systems with more than one physical chip, moving a process to another node means that all the local memory it had allocated while running on the original node is now effectively remote memory.

So, moving processes between CPUs is always costly, but ranges from hugely expensive when moving between NUMA nodes to a fairly small cost when moving between the cores of a dual-core chip.

Minimizing process movement directly conflicts with the goal of maximizing CPU utilization, so a balance must be struck. The Linux scheduler is able to recognize the topology of a system (cores, chips, nodes, etc.) and allows tasks to move more freely between cores than it does between nodes, thus maximizing CPU utilization without moving tasks away from their local memory.

What User Mechanisms, Commands or System Calls (e.g., taskset, numactl, etc.) Are Available to Allow Application Control on a System with Dual-core CPUs?

CPU Binding

Processes can be bound to any subset of the CPUs in the system by modifying their CPU affinity mask. A child process inherits the CPU affinity of its parent. In this way, one application (all its processes and threads) can be made to run on a specific set of CPUs, and another application in the same system can run on another (possibly intersecting) set of CPUs.

The command **'taskset'** controls which CPUs an application may run on. **taskset** can modify the CPU affinity of a currently running process, or run a new command with a given CPU affinity. For full details, view the **man page**.

taskset example:

'taskset -c 0-3,7 startserver'

This invocation will run the command **'startserver'** bound to the given set of CPUs (0, 1, 2, 3, 7).

The **'sched_setaffinity'** and **'sched_getaffinity'** system calls are the programmatic interface to set task CPU affinities. They are the interfaces **taskset** uses, and can perform any of the same functions. The man page has full API details.

Problem with CPU Affinity

The main problem of CPU binding with affinities is that it can cause Linux's multiprocessor CPU scheduler to become inefficient and possibly stop working completely.

This problem will become noticeable if there are many processes bound to a small set of CPUs, causing those CPUs to be much more highly loaded than other CPUs in the system. This situation will cause load balancing between the less loaded CPUs to stop functioning or function less efficiently.

This problem is not an issue if all processes in the system are carefully bound; however, it is not a good tool for more ad hoc administration. For example, assigning all low priority batch tasks to a single CPU (**'cpusets'**) can be a better option.

cpusets

cpusets allow a machine to be partitioned into subsets. These sets may overlap, and in that case they suffer from the same problem as CPU affinities. However, when the **cpusets** do not overlap, they can be switched to "exclusive" mode, which effectively causes the scheduler to behave as though they are two entirely different systems. As such, scheduling behavior is very good; however, an idle **cpuset** will remain idle even while others are very loaded.

cpuset can also be used to bind groups of tasks to specific memory nodes.

cpuset documentation can be found in **Documentation/cpusets.txt** in the Linux kernel source.

Memory Binding

Linux supports explicit control and binding of tasks to NUMA memory nodes in a similar manner to CPU binding. *A NUMA API for Linux* (www.novell.com/collateral/4621437/4621437.pdf), referenced earlier, provides a good introduction to memory binding and control facilities in Linux.

How Does Dual-core Interact with AMD PowerNow! Technology?

In a dual-core system, both cores of each chip will always run at the same frequency. This means frequency will only be lowered when both cores are idle. If power saving is a priority on a multiprocessor system, then both cores of a chip should be highly utilized before any load is moved to another chip.

The Linux kernel scheduler currently will not schedule tasks with power saving as a goal. In fact, Linux tries to make use of the AMD Opteron processor's memory resources where possible, which can be in direct conflict with a power-saving scheduling strategy. For example, when a new process is forked and calls **exec()**, Linux will try to move the process to an idle chip even if the other core of the current chip is idle. Such a strategy is usually the right one for best performance, but not for power saving.

Note: AMD Opteron processor-based systems with one dual-core chip are not affected by this trade-off.

Explicit CPU affinities may be required in order to achieve the best power usage for a given workload. Generally, this won't be a concern for servers and workstations; however, for specialized applications such as embedded systems or clusters, power usage can be a factor.

As described above, it is difficult for applications to control CPU affinities in the presence of changing demand or CPU usage patterns. Careful analysis of the workload and CPU usage patterns will be required, and such a task may simply be infeasible.

How Does Dual-core Interact with I/O and Async I/O?

I/O and async I/O are typically very kernel-intensive activities. All recent versions of the Linux kernel (including SUSE Linux Enterprise Server 9 from Novell, and the upcoming SUSE Linux Enterprise Server 10) are generally very well parallelized and NUMA aware. As such, the I/O capability of a system is likely to be improved with dual-core systems, though exact behavior is always workload dependent.

Use of NUMA and multiprocessor tuning techniques in the application, as described above, will often assist the kernel in taking advantage of dual-cores (and multiprocessor and NUMA systems in general).

Should Hyper-threading Be Disabled on Dual-core?

The question if hyper-threading should be disabled on dual-core is difficult to answer.

It will be very dependent on the workload and the CPU and system architectures in question. The POWER5 chip is dual-core and has multi-threading (similar to hyperthreading). IBM often submits the commercial performance results of its POWER5-based systems with their version of multi-threading turned on.

On Intel chips with hyperthreading, performance gains are often reported on workloads where there is a significant floating-point element.

Similarly, for single-core systems, the best advice is to test both hyperthreading enabled and disabled to see which performs better for your specific workload.

Conclusion

The introduction of x86 dual- or multi-core technology will change commercial and consumer computing while offering new opportunities for software developers. The evolution to multi-core processors is an exciting technological advancement that will play a central role in driving relevant advancements, providing greater security, resource utilization and value for businesses and consumers. The client and consumer markets will have access to superior performance and efficiency compared to single-core processors, as the next generation of software applications are developed to take advantage of multi-core processor technology.

Corporate IT systems currently optimized for SMP multi-threaded applications should see significant performance increases by using dual- or multi-core processors. This logical performance boost will take place within current, available hardware and socket designs, enabling corporate IT managers to add more sophisticated system layers, like virtualization and security, without significant disruption to legacy systems. Other key benefits are simplified manageability, lower total cost of ownership (TCO), and maximum processor performance. The AMD Opteron processor with Direct Connect Architecture enables one platform to meet the needs of multi-tasking environments, providing platform longevity.

Dual or multi-core processors can immediately benefit businesses and general consumers by providing the capability to run multimedia and security applications with increased performance. With multi-core processors, a new era of true multitasking emerges. AMD Athlon™ 64 X2 Dual-Core processors will take computing to a new level by enabling people to simultaneously burn a CD, check e-mail, edit digital photos and run virus protection software—all with increased performance. Multi-core processors also will enable the growth of the digital home: a centralized PC will be able to serve multiple rooms and people in the home.

Software professionals regularly push the limits of current processor capacity. Multi-core processors will solve many of the challenges currently facing software designers by delivering significant performance increases at a time when they need it most. Multi-core processors, in combination with new compiler optimizers, will reduce compiling times by as much as 50 percent, giving developers a critical advantage in meeting time-to-market demands. Software vendors also can use more multi-threaded design methods for delivering enhanced features. With the advent of multi-core processors and adoption of multi-core computer platforms by businesses and consumers, software vendors will have a much larger marketplace in which to distribute new and improved applications.

Novell has optimized its SUSE Linux enterprise products for AMD64 dual-core technology. Your customers will be able to extract the most in performance and reliability when they run your applications on SUSE Linux and the AMD Opteron platform.

Glossary

Key Terms To Understanding Dual-core:

Cache

A cache is a pool of entries or collection of data duplicating original values stored elsewhere or computed earlier.

Chip

The package that contains one or more cores.

Compute bound

If the performance of an application is limited by a system's CPU resources, it is called compute bound.

Core

This is the basic unit of execution.

CPU

CPU is the abbreviation of "central processing unit," and is pronounced as separate letters. The CPU is the brains of the computer. Sometimes referred to simply as the processor or central processor, the CPU is where most calculations take place. The term can reference a chip or a core (e.g., Linux calls each core a CPU).

Dual-core

Dual-core refers to a CPU that includes two complete execution cores per physical processor.

Integrated circuit

Another name for a chip, an integrated circuit (IC) is a small electronic device made out of a semiconductor material.

Memory bound

If performance is limited by a system's memory bandwidth, it is called memory bound. (Memory bound often refers to applications with a working set too big to fit in physical memory, but that is not the case here.)

Node

A node is a basic unit used to build data structures, such as linked lists and tree data structures.

NUMA

Non-Uniform Memory Access (NUMA) is a computer memory design used in multiprocessors, where the memory access time depends on the memory location relative to a processor. Under NUMA, a processor can access its own local memory faster than non-local memory, i.e., memory that is local to another processor or shared between processors.

processor

"Processor" is the shortened term for microprocessor or central processing unit (CPU).

SMP

Symmetric multiprocessing (SMP) is the processing of program code by multiple processors that share a common operating system and memory subsystem.

www.novell.com

Contact your local Novell office,
or call Novell at:

1 888 321 4272 U.S./Canada
1 801 861 4272 Worldwide
1 801 861 8473 Facsimile

Novell, Inc.
404 Wyman Street
Waltham, MA 02451 USA

Novell would like to thank AMD for their contributions to and their support of the creation of this document.

Novell