AMD

OPTERON PROCESSOR AMD

AMD Opteron™ 6200 Series Processors

Linux Tuning Guide

Contents

1.0 Introduction

This guide provides configuration, optimization, and tuning information and recommendations for AMD Opteron™ 6200 Series processors (formerly code-named "Interlagos" and built on AMD's new core architecture) running in a Linux environment. This guide is designed to help users through initial bios and system setup to ensure that a base level of performance is achieved. Once a system is configured to this level, users can reference AMD's Software Optimization Guide for AMD Family 15h Processors to help further optimize workload performance.

1.1 Intended Audience

This document is intended for High Performance Computing (HPC) systems admins, application end-users, and developers on a Linux platform who perform application development, code tuning, optimization, and initial system installation, and base-level system checks. It also provides some basic troubleshooting information to help with performance issues.

The information presented here assumes the reader has a basic understanding of the AMD Opteron™ 6200 Series processors. More information is located at: http://www.amd.com/us/products/server/processors/6000-series-platform/6200/Pages/6200-series-processors.aspx.

For complete information on the AMD Opteron™ processor architecture and instruction set, see the multi-volume *AMD64 Architecture Programmer's Manual* available from http://www.amd.com. The following table provides individual volumes and their order numbers.

Title	Order Number
Volume 1: Application Programming http://support.amd.com/us/Processor_TechDocs/24592_APM_v1.pdf	24592
Volume 2: System Programming http://support.amd.com/us/Processor_TechDocs/24593_APM_v2.pdf	24593
Volume 3: General-Purpose and System Instructions http://support.amd.com/us/Processor_TechDocs/24594_APM_v3.pdf	24594
Volume 4: 128-Bit and 256-Bit Media Instructions http://support.amd.com/us/Processor_TechDocs/26568_APM_v4.pdf	26568
Volume 5: 64-Bit Media and x87 Floating-Point Instructions http://support.amd.com/us/Processor_TechDocs/26569_APM_v5.pdf	26569
Volume 6: 128-Bit and 256-Bit XOP and FMA4 Instructions http://support.amd.com/us/Embedded_TechDocs/43479.pdf	43479

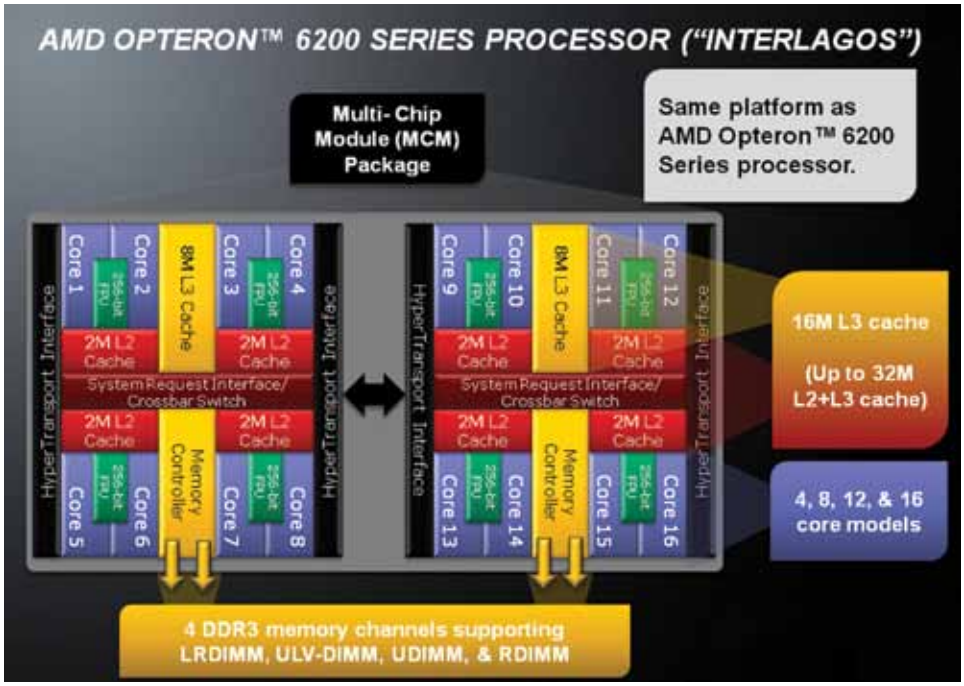
1.2 AMD's New Core Architecture Overview

The AMD Opteron™ 6200 Series processor uses AMD's newly released core architecture. This architecture is based on a building block called a module. Each module has two tightly coupled x86 processing engines that are called cores. The architecture is based on a "share what makes sense" approach, with each core having its own dedicated resources, such as integer scheduler, execution engine, and L1 cache and the two cores of a module sharing some resources, such as instruction fetch, decode, Floating Point Unit, and L2 cache. Each processor for the AMD Opteron™ 6200 Series is a Multi-Chip Module (MCM), meaning that it has two CPUs on a die package.

The AMD Opteron™ 6200 Series processors can boost core frequencies by allocating more power to individual cores, subject to a total power limit and other restrictions. Cores that are idle can "go to sleep" and turn off their power draw, allowing more power to be dedicated to active cores. As a result, a single core job could see a processor running at 3.2Ghz, which all cores active see cores running at 2.3Ghz (for top 16-core standard power AMD Opteron™ 6276 Series processors).

This following block diagram of the G34 socket Interlagos processor shows a 16-core variant of the part. There are a few new key features that have a significant effect on performance. As an introduction to this topic, note that the device is organized as a Multi-Chip Module, which means that there are two CPU dies in the package, interconnected using a HyperTransport link. Each die has its own memory controller interface to allow external connection of up to two memory channels per die.

This memory interface is shared among all the CPU core pairs on each die. Each die is organized as 2-, 3-, or 4-core pairs, and each core pair has two CPU cores, each of which has a set dedicated integer unit and general purpose register files. In addition, there are resources that are shared between the cores in a pair that operate at double rate, including instruction cache, instruction decode logic, Floating Point Unit, and L2 cache. Each core pair appears to the operating system as two completely independent CPU cores. Thus, to the OS, the device shown in the following block diagram appears as 16 CPUs.



Sharing of CPU resources is an engineering design decision intended to provide superior performance for a broad range of applications while maximizing power efficiency. Since most of the shared resources can provide more performance than a single CPU core can take advantage of, it makes good engineering sense to share among multiple cores.

The fact that some resources are shared has implications for performance since, in some cases, the shared resources may not satisfy the full demand from an application running on both CPU cores. Because of this, it is important to understand more about these shared resources.

1.3 Shared Resources

AMD Opteron™ 6200 Series processors shared resources boost single core performance. The processor has numerous features designed to boost the performance of single-core jobs and a job load that does not use all the cores. This is a good thing but, unless understood, could be viewed as poor scaling. The goal is to run single-core jobs faster and, where possible, reallocate resources and power to boost performance. Therefore, you see a more powerful core in many dimensions. The following features can be shared and reallocated to some degree.

- POWER AND FREQUENCY
  - We can boost frequencies by allocating more power to the single core, subject to a total power limit and other restrictions. Core pairs that are idle can "go to sleep" and turn off their power draw, allowing more power to be dedicated to active cores.
  - A single-core job could see a processor running at 3.2Ghz, while with all cores active would run at 2.3Ghz (for top 16-core standard power AMD Opteron™ 6276 Series processors).
- FLOATING POINT UNIT
  - Each core in a core pair shares the floating point unit. The floating point unit has two FMAC units, each able to produce a 128-bit result each cycle.
- MEMORY BANDWIDTH
  - All cores on a die are behind a single memory controller. As more cores are added, the available bandwidth gets shared. Different applications have different requirements for memory bandwidth and will be affected differently by how memory is shared by the cores.
- L2 CACHE
  - Each core in a core pair shares the L2 cache.
- L3 CACHE
  - All cores on a die share the L3 cache. There are two die in a package.
- INSTRUCTION FETCH AND DECODE CIRCUITRY
  - Each core in a core pair shares instruction fetch and decode circuitry. This is generally invisible to program performance.
- I/O AND INTERCONNECT BANDWIDTH
  - At the board level, all cores share the I/O and interconnect bandwidth.

1.4 Dedicated Resources

Inside of each module are three dedicated schedulers, one for each integer core and one to feed the Flex FP. The integer schedulers are 40-entry and the Flex FP scheduler is 60-entry. By having a dedicated scheduler for each integer core, AMD's new core architecture helps ensure that the four integer pipelines are being kept continually filled with instruction for the highest efficiency.

Each integer core has control over its own scheduling so that there is no bottleneck between the two dedicated threads that are executing in the module simultaneously. The Flex FP scheduler is a single entity because in AVX mode it needs to be able to send a single stream of 256-bit AVX operations through the FP pipes. In 128-bit mode, the extra entries in the Flex FP scheduler help ensure that the two 128-bit FMACs are receiving a constant stream of math instructions to execute.

1.5 Floating Point Capabilities

Today's server workloads require a broad mix of processor capabilities, from those using mostly integer operations to those where floating point performance is paramount. The challenge for a general purpose processor is to be fast and power efficient at both of these extremes. The Flex FP is designed to support a wide range of applications that vary greatly in the amount of floating point work needed while reducing the power for those not needing much floating point.

The Flex FP supports the next generation AVX and FMA4 floating point instructions for both 128-bit and 256-bit operands.

The new FMA4 instructions implement A = B * C + D in a single instruction rather than using two instructions (an FMUL then an FADD). The FMA4 produces the result with lower latency than if an FMUL and an FADD were used. For scientific applications, a compiler can replace the majority of FMUL and FADD instructions with a single FMA4 instruction, reducing execution time and code size.

The floating point unit is capable of producing four double-precision FLOPS per cycle per clock cycle simultaneously to *each* core in a pair for a total of eight per core pair per cycle. This is comparable to the floating point performance per core per cycle of an AMD Opteron™ 6100 CPU. But, unlike prior CPUs, when one core is issuing fewer floating point instructions, the other core in the pair can use its four FLOPS/cycle plus any unused by the other core to fully exploit the capacity of the Flex FP. For example, in the extreme case of one core executing no floating point instructions, the other core of the pair could achieve up to 8 double precision floating point operations per cycle.

2.0 Getting Started

Start by ensuring your system is configured properly. Since memory configuration is critical to performance and vulnerable to misconfiguration, you will verify proper memory configuration by inspection, observing Linux's view of the memory configuration, and finally by verifying the memory performance using the STREAM benchmark.

In this section you will:

- Check physical memory configuration.
- Select specific BIOS options.
- Check NUMA configuration.
- Use STREAM to verify configuration.
- Run HPL.

2.1 Physical Memory Configuration Check

Always ensure you are using a balanced and symmetrical memory configuration. Follow the manufacturer's configuration guide, if available. Otherwise,

1. Ensure that all DIMMS installed are identical. Identical means DIMMs from the same manufacturer, model, speed, CAS, ranks, etc.
2. The least error-prone memory configuration is to have all memory sockets occupied with identical DIMMs. But, if one DIMM per memory channel is the desired configuration, be very careful to install a DIMM in the DIMM socket most distant from the CPU and to have exactly one DIMM per memory channel. Be careful not to plug two DIMMs into one channel, leaving an empty channel. In most systems that have two DIMM sockets per memory channel, you will achieve the best memory bandwidth when using Dual Rank DDR3 1600 DIMMs with one or two DIMMs per channel.
3. Consult the loading tables for the platform for supported DIMMs and configuration that achieve the best capacity and performance. Note that not all platforms support 1600 DIMMs. Having three DIMM sockets per channel can provide more capacity but will restrict memory bus frequency and make it more dependent on which DIMMs are supported at full speed. Be sure to check the manufacturer's documentation.

2.2 BIOS Configuration Summary

1. Load BIOS defaults.
2. Set the date and time.
3. Node Interleaving = Disabled.
4. Power Management/C1E = Disabled.
5. Core Performance Boost (CPB) = Disabled.
6. C6-State = Disabled.
7. Downcoring = Disabled.
8. HPC/Optimization Mode (if it exists) = Enabled.

9. Virtualization, AMD V = Disabled.
10. OS Power Management = Disabled.

Note: Always refer to your motherboard or system's owner's manual for further BIOS setting information.

For production HPC system BIOS recommendations, see section 4.0 Configure a Performant Production System below. See also the BIOS and Kernel Developer's Guide (BKDG) at: http://support.amd.com/us/Processor_TechDocs/42301_15h_Mod_00h-0Fh_BKDG.pdf. Always refer to your motherboard or system BIOS info in your owner's manual.

11. Install a new Linux distribution (e.g., SLES11sp1 or 2 or OpenSUSE12.1).

    The most recent version and update of other Linux distributions would also work for these tests (e.g., RHEL6.1); however, the instructions for disabling speed stepping differ among distributions.

12. After installation, disable automatic power state management so that the core frequency will remain at full speed by running the following command as root (for SLES11sp1):

    powersave -f

Next, you can begin to verify the memory performance of your new system.

2.3 Check NUMA Configuration

Run `numactl --hardware` at the command line and check that the size of memory on each node is as expected (e.g., on the 2P system, check that 25% of the total installed memory is on each node). You should also expect that the `free:` on each node to be similar on each node and close to the `size:` on each node (`size:` is shown in `red` and `free:` is shown in `blue` below).

In the following example, up to 3GB is in use on node 2 since something was running at the time `numactl --hardware` was running; however, when run immediately after system boot, the `free:` should be close to `size:`.

```
bash>  numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 16382 MB
node 0 free: 14730 MB
node 1 cpus: 8 9 10 11 12 13 14 15
node 1 size: 16384 MB
node 1 free: 15539 MB
node 2 cpus: 16 17 18 19 20 21 22 23
node 2 size: 16384 MB
node 2 free: 13467 MB
node 3 cpus: 24 25 26 27 28 29 30 31
node 3 size: 16368 MB
node 3 free: 15401 MB
```

```
node distances:
node   0   1   2   3
  0:  10  16  16  16
  1:  16  10  16  16
  2:  16  16  10  16
  3:  16  16  16  10
```

If the `size:` is different on some nodes, then DIMMs are either not identical or not plugged into the right sockets.

If `numactl --hardware` only shows one node, then ACPI is not operating properly in the kernel. This could be the result of ACPI not being enabled in BIOS or a kernel being too old to recognize the NUMA configuration BIOS provides to the kernel during boot.

2.4 STREAM to Verify Configuration

Many performance issues are normally caused by poorly configured memory. Since HPC performance is strongly dependent on memory performance, we next describe how to verify that the machine is configured to achieve the maximum memory performance possible with the AMD Opteron™ 4200/6200 Series processors. We will use the STREAM benchmark to verify memory bandwidth. We will show how to build STREAM with GCC and with the AMD Open64 Compiler Suite for a quick but low-performance test that can reach the potential memory bandwidth performance expected with AMD Opteron™ 4200/6200 Series processors.

- GET AND BUILD STREAM

1. Download STREAM from the University of Virginia at: http://www.cs.virginia.edu/stream/.
2. Download the C source code from: http://www.cs.virginia.edu/stream/FTP/Code/stream.c.
3. Edit stream.c and change the definitions of N, NTIMES, and OFFSET to the following:

   #define N 873800

   #define NTIMES 30

   #define OFFSET 1840

Non-uniform memory access (NUMA) is a system design that became popular on UNIX servers in the mid-1990s. It is based on the concept that access to all bytes in memory need not occur at uniform access rates. Today's non-uniform memory access is a performance-enhancing technique that leverages the extraordinary amount of memory now available on systems. For an additional discussion of NUMA, see http://developer.amd.com/assets/LibNUMA-WP-fv1.pdf.

2.5 Easy STREAM Using GCC Compiler

GCC is included with any Linux distribution. Unfortunately, GCC does not generate efficient code for STREAM. Nonetheless, using STREAM built with GCC can uncover memory performance issues.

- BUILD STREAM WITH GCC VERSION 4.3.4, WHICH IS NATIVE TO SLES 11 SP1, USING THE FOLLOWING FLAGS:

  -O2 -msse -msse2 -msse3 -o stream stream.c -static -fopenmp

- RUN STREAM

  Run STREAM on all 32 cores of a 2P system with AMD Opteron™ 6276 Series processors and 64GB (8x8GB) of 1600Mhz memory as follows:

```
> export OMP_NUM_THREADS=32
> ./stream
   -------------------------------------------------------------
   STREAM version $Revision: 5.9 $
   -------------------------------------------------------------
   This system uses 8 bytes per DOUBLE PRECISION word.
   -------------------------------------------------------------
   Array size = 87380000, Offset = 1840
   Total memory required = 2000.0 MB.
   Each test is run 30 times, but only
   the *best* time for each is used.
-------------------------------------------------------------
       (lines deleted)
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       36916.7828       0.0386       0.0379       0.0390
Scale:      37034.2906       0.0384       0.0378       0.0387
Add:        41602.0300       0.0514       0.0504       0.0518
Triad:      41769.3595       0.0512       0.0502       0.0517
-------------------------------------------------------------
```

  - On all cores of only 1 NUMA node (i.e., 1 CPU die) of a 2P Opteron™ 6276 (32 cores) with 64GB (8x8GB) 1600Mhz memory, we observe about a quarter of the performance than with running on all cores:

```
> export OMP_NUM_THREADS=8
> export GOMP_CPU_AFFINITY=”0 1 2 3 4 5 6 7”
> ./stream
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       10520.9029       0.1371       0.1329       0.1417
Scale:      10583.1833       0.1358       0.1321       0.1382
Add:        12010.5944       0.1806       0.1746       0.1843
Triad:      12062.1800       0.1802       0.1739       0.1839
-------------------------------------------------------------
```

  Note: With only one of the four NUMA nodes running STREAM, the memory bandwidth is about 25% of that when running on all 32 cores above.

  Note: Also, run STREAM on each die to see that STREAM is the same on each. If not, STREAM on all cores is lower than it should be and may result from having a memory channel empty (i.e., plugging in a DIMM in the wrong slot).

At least 20% better STREAM performance can be achieved using GCC 4.6.0 or later, but these later versions are unlikely to be included in Linux distributions today. However, 70% better STREAM can be achieved by using the AMD Open64 compiler.

2.6 High Performance STREAM Using AMD Open64 Compiler

Build STREAM using the Open64 compiler when attempting to measure the best achievable memory performance.

- Download and install the AMD Open64 compiler.
  - Locate the Open64 compiler on http://developer.amd.com/tools/open64/pages/default.aspx and then download and follow the installation instructions.
- AMD Open64 compiler flags.
  - Use compiler AMD Open64 version 4.5.1 or later with the following flags:

```
-march=bdver1 -mp -Ofast -LNO:simd=2 -WOPT:sib=on
-LNO:prefetch=2:pf2=0 -CG:use_prefetchnta=on -LNO:prefetch_ahead=4
-static
```

- Run STREAM with the following expected performance.
  - Run on all cores.

    The following is an example run on 2P AMD Opteron™ 6276 Series processors (32 cores) with 64GB (8x8GB) 1600Mhz memory:

```
> export OMP_NUM_THREADS=32
> ./stream
-------------------------------------------------------------
Function    Rate (MB/s)     Avg time     Min time     Max time
Copy:       67973.1136       0.0208       0.0206       0.0211
Scale:      70406.8166       0.0200       0.0199       0.0201
Add:        65922.3917       0.0319       0.0318       0.0321
Triad:      65656.1828       0.0321       0.0319       0.0324
```

    Note: The AMD Open64 version of STREAM yields 70% better bandwidth than the GCC version when running on all 32 cores.

  - Run on all cores of each NUMA node.

    The following is an example on the same system but running only on the first NUMA node, cores 0-7.

```
> export O64_OMP_AFFINITY=”TRUE”
> export O64_OMP_AFFINITY_MAP=”0,1,2,3,4,5,6,7”
> export OMP_NUM_THREADS=8
> ./stream
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       17311.7443       0.0812       0.0808       0.0828
Scale:      18046.0464       0.0779       0.0775       0.0784
```

```
Add:        16798.1705       0.1253       0.1248       0.1267
Triad:      16724.2629       0.1259       0.1254       0.1279
-------------------------------------------------------------
```

Repeat on each NUMA node. If there is significant performance difference between the bandwidth achieved on each NUMA node, check the memory configuration that corresponds to that NUMA node.

- Peak memory bandwidth is achieved when STREAM is run on three cores of each NUMA node. For example, the following run shows that the same system is capable of achieving STREAM 5% better than when using all cores.

```
> export O64_OMP_AFFINITY=”TRUE”
> export O64_OMP_AFFINITY_MAP=”2,4,6,10,12,14,18,20,22,26,28,30”
> export OMP_NUM_THREADS=12
> ./stream
   -------------------------------------------------------------
   STREAM version $Revision: 5.9 $
   -------------------------------------------------------------
   This system uses 8 bytes per DOUBLE PRECISION word.
   -------------------------------------------------------------
   Array size = 87380000, Offset = 1840
   Total memory required = 2000.0 MB.
   Each test is run 30 times, but only
   the *best* time for each is used.
   -------------------------------------------------------------
   Number of Threads requested = 12
   -------------------------------------------------------------
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   Printing one line per active thread....
   -------------------------------------------------------------
   Your clock granularity/precision appears to be 1 microseconds.
   Each test below will take on the order of 22461 microseconds.
      (= 22461 clock ticks)
```

```
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       73113.2180       0.0194       0.0191       0.0198
Scale:      74003.6161       0.0193       0.0189       0.0197
Add:        68265.6350       0.0310       0.0307       0.0315
Triad:      67651.3341       0.0314       0.0310       0.0317
-------------------------------------------------------------
Solution Validates
-------------------------------------------------------------
```

2.7 Run High Performance Linpack

It is beneficial to run the High Performance Linpak (HPL) benchmark to verify adequate cooling and to burn-in the system. The HPL benchmark is very demanding on the floating point units and consumes almost peak power. In addition, it uses a significant amount of memory and checks the answer it computes, checking for memory errors. Thus, it is an excellent test of system stability. Many HPC centers use HPL to burn-in their new systems.

For instructions on installing and running the HPL benchmark on servers using AMD Opteron™ 6200 Series processors, refer to the whitepaper at http://developer.amd.com/Assets/linpack_wp_bd.pdf.

3.0 Operating System and Software Choices

This section helps you understand the various operating systems, compilers, and libraries you should consider when attempting any optimization or tuning processes.

In this section, you will:

- See the HPC Sample Operating System (OS) & Compiler Configurations.
- See the Commercial Options for Linux.
- Understand Compiler Choices.
- Find Linux Kernel Versions and Distributions.
- Check Configuration After Installing Linux.
- Learn about compiling for FMA4.
- Find available Libraries.

3.1 HPC Example OS & Compiler Configurations

Here are a few example starting point operating system and compiler configurations that can be used for common HPC workloads and development.

	Operating System	Compiler	Library
Recommended for SPECCPU, LINPACK, HPC Challenge	RHEL 6.2 with patches	Open64 4.5	ACML 5.1
Recommended for application development and benchmarks with gcc	RHEL 6.2 with patches	GCC 4.7	ACML 5.1 and/ or libM 3.0
Recommended for HPC application code development	RHEL 6.2 with patches	Open64 4.5 or PGI 11.9	ACML 5.1

3.2 Linux Kernel Versions and Distributions

For best possible performance, you should use an **enabled** Linux kernel version or distribution that fully supports and enables the new features of the AMD Opteron™ 4200/6200 Series processors. The following graphic shows AMD Opteron™ 4200/6200 Series processors OS support options:

Assumes latest updates/patches are installed*

ENABLED Optimized to support some or all of the features of AMD's new core architecture	COMPATIBLE Will boot and run but not take advantage of the features of AMD's new core architecture outside of new instructions	NOT SUPPORTED Will not run on platforms based on AMD's new core architecture and/or will not be supported by OSV
Includes new instruction support: Linux kernel 2.6.37 + , 3.0 + Novell SLES 11 SP2 (includes Xen) RHEL 6.2 with KVM (with latest z-stream patches) Windows Server 2008 R2 SP1 (optional scheduler patch available) Windows Server 2012/Hyper-V (in development) Xen 4.1 + Ubuntu 11.04 (w/ KVM) VMware vSphere 5.0 *Versions in this category also include latest software advances.*	**Includes new instruction support:** Linux kernel 2.6.32 – 2.6.36 Novell SLES 11 SP1 RHEL 6.1 Ubuntu 10.10 **Does not support new instructions for either** AMD or Intel's new core architectures (formerly code-named "Bulldozer" and "Sandy Bridge"): Hyper-V R1 Hyper-V R2, Hyper-V R2 SP1 Novell SLES 10 SP4 and higher RHEL 5.7 (included KVM) Solaris 10u9, 11 VMware vSphere 4.1u2 Windows Server 2003 R2 SP2 Windows Server 2008 R2 Windows Server 2008 SP2 Xen 3.4.2 *Versions in this category also include latest software advances.*	**Linux kernel 2.6.31 or earlier** Novell SLES 10 thru SP3 Novell SLES 11 RHEL 4.x RHEL 5.0 – 5.5 RHEL 5.6 (can run with patches but is not supported by Red Hat) RHEL 6.0 Solaris 10 – 10u8 VMware ESX 3.5 VMware ESX 4.0 – 4.1u1 Windows Server 2003 versions prior to R2 SP2

The following list shows the kernel versions and distributions that take advantage of AMD's new core architecture and instructions and support one or more of these processors' new features.

Enabled Linux kernel releases and distributions include:

- Linux kernel 2.6.32.46 (longterm release).
- Linux kernel 2.6.33.19 (longterm release).
- Linux kernel 2.6.34.10 (longterm release).
- Linux kernel 2.6.35.14 (longterm release).
- Linux kernel 2.6.36.3 (snapshot release).
- Linux kernel 2.6.37.6 and higher
- Linux kernel 3.
- Novell SUSE Linux Enterprise Server (SLES) 11 SP2 (includes Xen 4.1 and KVM).
- Red Hat Enterprise Linux (RHEL) 6.2 (includes KVM) and Red Hat Enterprise Linux derivatives CentOS 6.2 and Scientific Linux 6.2.
- Ubuntu 11.04 (includes KVM).

3.3 Check Configuration After Installing Linux

Run `free:` at the command line and check that the total memory is the expected size, 65958808, in the following example:

```
bash>  free
             total       used       free     shared    buffers     cached
Mem:      65958808    5401568   60557240          0     151892    2483496
-/+ buffers/cache:    2766180   63192628
Swap:      2104472          0    2104472
```

If total memory is only showing about 3GB and you know you have installed more than that, then the most likely cause is a kernel that does not fully support AMD Opteron™ 6200 Series processors.

It would also be useful to check the NUMA configuration; see 2.3 Check Numa Configuration for the details.

3.4 Compiler Choices

The AMD Opteron™ 4200/6200 Series processors include a variety of new instructions that optimize compute-intensive software. This includes AVX, FMA4, and XOP instructions. More information about these instructions can be found in the *AMD64 Architecture Programmer's Manual* at http://support.amd.com/us/Embedded_TechDocs/43479.pdf.

For best possible performance, your code must be compiled with a compiler that is enabled to use these features in AMD Opteron™ 4200/6200 Series processors. This is particularly important for floating point codes. The following chart gives you a list of compilers that offer optimized support for AMD Opteron™ 4200/6200 Series processors.

This table shows the compilers supported for this processor at the time this document was published. Be sure to consult http://developer.amd.com/zones/hpc/Pages/default.aspx for the latest compiler information.

Compiler	Status	SSSE3 SSE4.1-.2 AVX	FMA4 XOP	Auto Generates Code	Comments
GCC 4.6.2 GCC 4.7	Available	✔	✔	✔	GCC 4.4 is included in RHEL 6.0 distribution and should be updated to GCC 4.6.2 or 4.7 for optimized support
Open64 4.5	Available	✔	✔	✔	Provides incremental performance (~2% SPECCPU) and functionality improvements
PGI 11.9	Available	✔	✔	✔	PGI Unified Binary™ technology combines into a single executable or object file code optimized for multiple AMD and Intel processors

- GCC (GNU Compiler Collection) - 4.6.2 (or 4.7 which is a bit faster). These versions can generate FMA4 and AVX instructions.
  - http://gcc.gnu.org/
- X86 Open64 Compiler Suite (4.5.1 or later).
  - http://developer.amd.com/tools/open64/Pages/default.aspx

3.5 Compiling for AMD's New Core Architecture Instructions

The shared floating point unit for the AMD Opteron™ 6200 Series processors features new FMA4 and XOP instructions that can improve floating point throughput for workloads. For more details on the new instructions see the *AMD64 Architecture Programmer's Manual Volume 6: 128-Bit and 256-Bit XOP and FMA4 Instructions* at http://support.amd.com/us/Embedded_TechDocs/43479.pdf. The following graphic shows a bit about the new instructions:

FMA4 Overview (AMD Unique)	XOP Overview (AMD Unique)
Performs fused multiply–add (FMA) operations. The FMA operation has the form d = a + b x c. FMA4 allows a, b, c, and d to be four different registers, providing programming flexibility. • A fast FMA can speed up computations which involve the accumulation of products • FMA capabilities are also available in IBM Power, SPARC, and Itanium CPUs. • Intel is anticipated to introduce FMA3, a more limited implementation of FMA (where d is the same register as either a, b, or c) to Xeon in 2013 timeframe*	Provides three- and four-operand non-destructive destination encoding, an expansive new opcode space, and extension of SIMD floating point operations to 256 bits. • Horizontal integer add/subtract • Integer multiply/accumulate • Shift/rotate with per-element counts • Integer compare • Byte permute • Bit-wise conditional move • Fraction extract • Half-precision convert

The classic example of a floating point intensive code is the DGEMM (Double-precision GEneral Matrix Multiply) routine that is heavily used by the HPL benchmark. FMA4 instructions have a latency of five cycles. Individual SSE and AVX add and multiply instructions also have a latency of five cycles. As a result, binaries compiled to run on previous generations of hardware with the SSE/SSE2 instruction sets will not run as fast on the AMD Opteron™ 6200 Series processors.

Because of this, it is essential to recompile any floating point-intensive application with appropriate FMA4 compiler flags and to link with optimized libraries to ensure that the application can take advantage of the new floating point unit's full capability.

Users should start using flags recommended in the latest *Compiler Options Quick Reference Guide* for the AMD Opteron™ 4200/6200 Series processors based on the new core architecture. The guide can be found at: http://developer.amd.com/Assets/CompilerOptQuickRef-62004200.pdf. The overall recommendation for performance on the 6200 processor is to compile generating both FMA4 and AVX128 instructions (the exact flag depends on the compiler).

If an application binary currently includes the instructions that are common to AMD's new core architecture and to the Intel CPUs (e.g., AVX, SSE3, SSE4.1, SSE4.2, AES-NI), then this code will run well on the AMD Opteron™ 6200 CPU, as long as the binary checks only the ISA feature bits in the CPUID. Unfortunately, much code generated by the Intel compiler and Intel libraries inserts checks for the CPU Vendor to be "GENUINEINTEL" and will thus either fail or execute an inefficient code sequence on AMD processors. Recompile such software.

3.6 Libraries

Special purpose high-performance libraries are another important contributor to application performance. A classic example of this is when a program makes extensive use of BLAS (Basic Linear Algebra Subroutines) and LAPACK (Linear Algebra PACKage) subroutines. In these cases, it is imperative to use an FMA4-enabled floating point library. The AMD Core Math Library has been tuned to use FMA4 instructions, where possible, and will provide a significant performance boost when compared to other libraries that have not been tuned for AMD Opteron™ 4200/6200 Series processors.

- ACML (AMD Core Math Library) provides a free set of thoroughly optimized and threaded math routines for HPC, scientific, engineering, and related compute-intensive applications.
- ACML consists of the following main components:
  - A full implementation of Level 1, 2, and 3 BLAS with key routines optimized for high performance on AMD Opteron™ processors.
  - A full suite of LAPACK routines. As well as taking advantage of the highly tuned BLAS kernels, a key set of LAPACK routines has been further optimized to achieve considerably higher performance than standard LAPACK implementations.
  - A comprehensive suite of Fast Fourier Transforms (FFTs) in both single-, double-, single-complex, and double-complex data types.
  - Random Number Generators in both single- and double-precision.
  - For complete ACML V5.1.0 or later information, data sheet, performance libraries, and download, see http://developer.amd.com/libraries/acml/pages/default.aspx.

3.7 Other Libraries and Tools

Other modules and tools that may assist with building GCC and OpenMPI include:

- OpenMPI: http://www.open-mpi.org/
- Build the OpenMPI to include:
  - Portable Hardware Locality (hwloc): http://www.open-mpi.org/projects/hwloc/
  - KNEM module (Core - Core on-board memory single-copy optimization using this KNEM that builds a kernel module:
    http://runtime.bordeaux.inria.fr/knem/
    http://runtime.bordeaux.inria.fr/knem/doc/
- libnuma-develop (Module with headers and libraries for developing programs using NUMA policies)
  - http://oss.sgi.com/projects/libnuma/
  - http://rpm.pbone.net/index.php3?stat=3&search=libnuma-devel&srodzaj=3
  - http://rpmfind.net/linux/rpm2html/search.php?query=libnuma-devel
- GMP (GNU Multiple Precision Arithmetic Library)
  - http://gmplib.org/
- LIKWID (Lightweight Performance Tools)
  - http://code.google.com/p/likwid/
- Linux Performance Tools (Linux profiling with performance counters)
  - https://perf.wiki.kernel.org/
- AMD CodeAnalyst (Performance Analyzer for Linux)
  - http://developer.amd.com/tools/CodeAnalyst/codeanalystlinux/Pages/default.aspx

4.0 Configure a Performant Production System

This section provides tasks to help you check existing performance and improve future performance on production systems.

4.1 BIOS Configuration Options

The following are the recommended BIOS configurations for HPC.

- APM enabled to enable core frequency boost.
- C6 state enabled to allow boost to Pb0 that can provide up to 1 GHz (OPN dependent) core frequency boost when half of the CUs per die are in halted in C6.

4.2 HPC P-state Mode

Be sure to enable the HPC P-state mode (if available in the BIOS) to prevent APM from decreasing the core frequency below software P0 frequency (i.e., the base frequency for the CPU, e.g., n GHz for an AMD Opteron™ 6276).

4.3 Power Management and Boost

Application Power Management (APM) allows the processor to provide maximum performance while remaining within the specified power delivery and removal envelope. APM extends the normal P-states used by the OS to control core frequency (P0-P6) by adding two boosted P-states, Pb1 and Pb0, of which the OS is unaware. APM dynamically monitors CPU activity and generates a deterministic approximation of power consumption. If power consumption exceeds a defined Thermal Design Power (TDP) for the CPU, APM applies a P-state limit to reduce power consumption. However, if the CPU is using less than TDP, APM will shift to a boosted state (either Pb1 or Pb0) with an increased core frequency to apply this unused power to improve performance.

APM ensures that average power consumption over a thermally significant time period remains at or below the TDP for the CPU mode being used.

Two levels of boosted P-states are supported. APM can place compute units in the first level of boosted Pstates (Pb1) if the OS kernel requests the highest performance P-state available (e.g., P0) and processor power consumption remains within the TDP limit. The second level of boosted P-states can only be achieved if

- a subset (usually half) of compute units on each die are halted in CC6,
- software requests P0, the highest performance P-state available, and
- the processor power consumption remains within the TDP limit.

Here is an example of P-states for a Opteron™ 6276 with HPC P-state mode enabled in BIOS.

Pb0 := Freq: 3200 MHz

Pb1 := Freq: 2600 MHz

P0 := Freq: 2300 MHz

P1 := Freq: 2300 MHz

P2 := Freq: 2300 MHz

P3 := Freq: 2300 MHz

P4 := Freq: 1400 MHz

In this example, P0 is the highest non-boosted P state in normal operation. Pb1 is the first boosted frequency. Having HPC P-state mode enabled reconfigures P1 through P3 to be set to the same frequency as P0. This prevents slower performance as long as the system TDP design allows continuous operation at these frequencies.

4.3.1 Check OS Power Management Default

Linux in general (upstream) uses 'cpufreq' subsystem for power management. The power management modes are **ondemand, performance, conservative, powersave,** and **userspace**. In the performance mode, the kernel keeps the cores in P-state P0. By default on servers, **ondemand** policy is used, which scales up frequency to the highest software P-state possible when there are tasks to run. When there is nothing in the run queue for a core to do, in **ondemand**, the kernel will move to a lower performance P-state (e.g., P1 – P6).

An Opteron™ 6200 or 4200 CPU has two levels of boost: PB0 and PB1. Depending on the C-states of the sibling cores and the TDP utilization, the processor will transparently boost to PB0 (highest) when possible or PB1. This happens transparently to the kernel. All of the power management modes except **powersave** mode allow CPUs to boost transparently.

The following are Linux Distribution/Version defaults:

- Red Hat RHEL 6.1 and 6.2 by default uses **ondemand** mode (governor). 6.1/6.2 has cpuidle support that can also put cores into C6 states. Hence, highest boost levels can be achieved transparently by default.
- Red Hat RHEL 5.7/5.8 by default uses **ondemand** mode, but RHEL 5.7/5.8 do not support deep C-states; therefore, they will not enable the processor to do "big" boost (Pb0).
- Red Hat RHEL 5.x line does not support deep C-states. Hence, although boost happens transparently, the processor cannot boost to the highest boost P-state (Pb0).
- SUSE Linux Enterprise Server (SLES) 11 SP2 is similar to mainline. **ondemand** is the default mode. The OS supports cpuidle and highest boost states can be achieved transparently.

Note: The frequency in /proc/cpuinfo does not show any effect of boost. The turbostat utility, if in your distribution, can be used to actually read the boosted frequency of CPUs.

A useful public tool package for monitoring CPU boost is the cpufreq-utils package that can be used to display useful information about the current state of the CPUs with the cpufreq-info tool and allows you to watch the current speed of the CPUs and identify if and when they boost. This handy package is located at http://rpmfind.net/linux/rpm2html/search.php?query=cpufrequtils.

4.3.2 Trade-offs and Consequences

No discussion of changing processor clocking modes and power management features would be complete without a discussion of the consequences and trade-offs. The default settings should be suitable for many applications and certainly best for machines being deployed for general purpose computing with a wide variety of applications. There are some special circumstances where it provides a benefit to make these changes, for example, in some benchmarking scenarios. Keep the following points in mind when deciding when to use APM, HPC P-state mode, and other modifications to default power management settings.

- If performance is paramount, enable C6, APM, HPC P-state Mode, and put Linux governor in performance mode. This will run at frequencies between software P0 and Pb0 but will never run at the lower frequencies associated with software P1, P2, etc. This will allow the processor to run at the fastest possible clock speeds but will increase the amount of power drawn by the system.
- If stable performance is paramount (i.e., minimize jitter in the frequencies at which all of the cores are running), then disable APM and put Linux governor in performance mode. This will run at software P0 frequency only. This will enable more consistent results for benchmarks and comparison purposes. When power management is enabled, it can be difficult to determine if performance differences are due to clock speed changes or other factors.
- If power performance is paramount, enable C6 and APM, disable HPC P-state mode, and put Linux governor in on-demand mode. This will run extreme HPC workloads (e.g., HPL and even parts of SPEC CFP2006rate) at slower P-state frequencies than software P0, spending significant time in P1 and maybe even P2. Overall performance may be lower, but this mode should provide enough power consumption benefits, improving the ratio of performance to power consumption of the system.

4.4 Thread to Core Assignment Considerations

For a 1-socket IL-16 part with 16 cores running just two one-threaded jobs, the best allocation would be bind to core 0 and core 8 to allocate one job per die. You could order allocation to maximize the shared resources (typical round-robin is beneficial going first over die [eight cores per die-sharing memory and L3] then over core pairs [2 cores per pair sharing L2, FPU, and fetch decode]).

Note that for some applications you may want to minimize power use rather than maximize performance, so filling the jobs from core 0 in a linear fashion could be the best strategy. This will allow the unused core pairs to enter the C6 sleep mode saving considerable power.

The term *job* means a single-threaded unit of code such as an MPI rank or a single thread of an OpenMP application.

The interesting cases are:

- No cores active.
- 1 core active.
- All cores active.
- Some number of cores active between 1 and all.

Next, decide what you want to achieve in terms of power usage and resource allocation. Your choices are:

- Allocate jobs to minimize the total power used and allow most resources to be turned off.
  - Allocate from core 0 in a linear fashion until all cores are active.
- Allocate jobs to maximize performance by enabling the maximum resources to be turned on.
- Allocate round-robin over each of the following in turn until that resource is filled, then go round-robin to the next level.
  - Over sockets, skip by 16.
  - Over Die / NUMA memory domains, skip by 8.
  - Over Core pairs, skip by 2.

For a 2-socket node with 16-core parts, the highest performance order of core allocation is:

- Two jobs - Use both sockets and use both L3 caches: 0, 16.
- Four jobs - Use all 4 die and all memory controllers: 0, 8, 16, 24.

- 16 jobs - Use all L2 caches and shared floating point units: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30.
- 32 jobs - Use all L2 caches and shared floating point units: 0,1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31.
  - The *best* resource hogging allocation would maximize the area under the graph if plotting total node performance for each number of cores active going from 0 to 32 cores.
  - Sharing is only an issue when a particular program or workload mix on a node hits a limit.
  - Jobs can be bound to cores with the numactl utility. For example, the following runs a job on core 8 allocating memory local to that core.
  - `numactl  --physcpubind 8 –localalloc job_name`

5.0 Known Issues

5.1 Address Space Layout Randomization (ASLR)

The AMD Opteron™ 4200/6200 Series processors have a shared Level-1 instruction-cache that, under specific circumstances, leads to different performance characteristics to previous processor generations.

Running specific scenarios under Linux on Family 15h-based systems can cause a rapid increase of cache cross-invalidations leading to diminished system performance due to system configuration.

To learn how to avoid the ASLR issue and more about how this shared L1 cache micro-architecture works, refer to AMD's whitepaper *Shared Level-1 instruction-cache performance on AMD family 15h CPUs* located at: http://developer.amd.com/Assets/SharedL1InstructionCacheonAMD15hCPU.pdf.

6.0 Useful Tools

This section lists some tools you may find helpful when analyzing performance and measuring power states.

- **CodeAnalyst**
  - Use CodeAnalyst for application profiling. For details, see http://developer.amd.com/tools/CodeAnalyst/codeanalystlinux.
- **PAPI**
  - Performance Application Programming Interface (PAPI) enables a host of other university performance tools using performance counters. The document defining the performance counters is the Bios and Kernel Developer's Guide (BKDG) for the specific CPU family. The BKDG is located at http://developer.amd.com.
- **cpufreq-utils**
  - A handy public utility for monitoring CPUs and can be found at: http://rpmfind.net/linux/rpm2html/search.php?query=cpufrequtils.
  - The cpufreq-aperf tool displays the current speed of the CPUs and shows if and when they boost.
  - Instructions for using this tool are located at: http://www.thinkwiki.org/wiki/How_to_use_cpufrequtils.
- **turbostat**
  - reports processor topology, frequency, and idle power state statistics on modern X86 processors. Either command is forked and statistics are printed upon its completion, or statistics are printed periodically.
  - requires that the processor support an "invariant" TSC, plus the APERF and MPERF MSRs. **turbostat** will report idle CPU power state residency on processors that additionally support C-state residency counters.
  - The source code is located at: http://lxr.free-electrons.com/source/tools/power/x86/turbostat/turbostat.c.
- **PowerTOP**
  - PowerTOP is a software utility designed to measure, explain, and minimize a computer's electrical power consumption. It was released in 2007 under the GPLv2 license. It works for Intel, AMD, ARM, and UltraSPARC processors.
  - PowerTOP analyzes the programs, device drivers, and kernel options running on a computer based on the Linux and Solaris operating systems and estimates the power consumption resulting from their use. This information can be used to pinpoint software that results in excessive power use.
  - PowerTOP's minimum kernel requirement is now Linux version 2.6.36, which includes the infrastructure that is needed for performance; some of the features even require kernel version 2.6.37.
  - For all PowerTOP information, see: http://www.h-online.com/open/features/Powertop-2-0-saving-power-under-Linux-1257057.html.

7.0 AMD Reference Material

- X86 Open64 Compilers Suite: http://developer.amd.com/tools/open64/
- Using the x86 Open64 Compiler Suite: http://developer.amd.com/tools/open64/Documents/open64.html
- X86 Open64 4.2.5.1 Release Notes: http://developer.amd.com/tools/open64/assets/ReleaseNotes.txt
- AMD Developer Tools: http://developer.amd.com/tools/
- AMD Libraries (ACML, LibM, etc.): http://developer.amd.com/libraries/
- AMD Opteron™ 4200/6200 Series processors Compiler Options Quick Guide: http://developer.amd.com/Assets/CompilerOptQuickRef-62004200.pdf
- AMD OpenCL™ Zone: http://developer.amd.com/zones/OpenCLZone/
- AMD HPC: http://www.amd.com/hpc
- The Software Optimization Guide for Family 15h provides an excellent description of the microarchitecture followed by hundreds of pages on code generation cases: http://support.amd.com/us/Processor_TechDocs/47414_15h_sw_opt_guide.pdf
- AMD APP SDK Documentation: http://developer.amd.com/sdks/AMDAPPSDK/documentation/Pages/default.aspx
- ACML Information: http://developer.amd.com/libraries/acml/features/pages/default.aspx
- AMD64 Architecture Programmer's Manual Volume 6: 128-Bit and 256-Bit XOP and FMA4 Instructions: http://support.amd.com/us/Embedded_TechDocs/43479.pdf
- Shared Level-1 instruction-cache performance on AMD family 15h CPUs whitepaper: http://developer.amd.com/Assets/SharedL1InstructionCacheonAMD15hCPU.pdf
- 2- and 4-socket results for the AMD Opteron™ 6276 Series processors (16-core, 2.3Ghz). The SPEC runs used the X86 Open64 Compiler Suite: http://www.spec.org/cpu2006/results/res2011q4/cpu2006-20111025-18742.pdf http://www.spec.org/cpu2006/results/res2011q4/cpu2006-20111025-18748.pdf