Evaluation of ESX Server Under CPU Intensive Workloads

Terry Wilcox Phil Windley, PhD

{terryw, windley}@cs.byu.edu

Computer Science Department, Brigham Young University

Executive Summary

Virtual machines are used by IT departments to provide better hardware utilization and to isolate users and programs from each other. We explore how configuration changes affect the throughput of virtual machines hosted on ESX Server during CPU intensive workloads. In particular we explore how Hyper-Threading, Virtual SMP, and the amount of RAM allocated affect throughput.

Hyper-Threading is Intel's implementation of simultaneous multithreading technology and was introduced with the Xeon processor. Virtual SMP allows ESX Server to host virtual machines with two virtual processors.

Our tests were run on two Dell 6650 quad processor 2.20 GHz Xeon servers with 16GB of RAM, a Gigabit Ethernet NIC, and a 1.3 TB SAN. Our benchmarks were run on Linux kernel 2.4.21-15.ELsmp and compiled by gcc 3.2.3 unless specifically stated otherwise.

We used SPEC's cpu2000 benchmark suite to obtain our results. No attempt was made to optimize the benchmarks for the virtual machines and default settings were used in all cases. Unused virtual machines were suspended before each test. Each benchmark was run five times to ensure that the results were consistent and repeatable. We used the **RDTSC** instruction to time the benchmarks.

The table below summarizes our benchmark test cases.

Test Cases	Description
Scalability	
Hyper-Threading, 1 CPU	This compares throughput of one CPU virtual machines with Hyper-Threading both enabled and disabled.
Hyper-Threading, 2 CPUs	This compares the throughput of two CPU virtual machines with Hyper-Threading both enabled and disabled.
1 CPU vs. 2 CPUs	This compares the throughput of one and two CPU virtual machines.
Resource Allocation	
Memory Subsystem	This tracks throughput as the amount of RAM allocated to a virtual machine increases.

Table 1 - Summary of benchmark test cases.

Hyper-Threading significantly increases throughput for CPU intensive workloads. When twelve virtual CPUs were benchmarked and Hyper-Threading was enabled the throughput of single CPU virtual machines increased 21% and the throughput of two

CPU virtual machines increased 23% compared to when Hyper-Threading was disabled. Since Hyper-Threading is most effective with large numbers of virtual machines a host that is Hyper-Threading enabled should have at least two virtual CPUs in aggregate for every physical CPU. Using fewer virtual CPUs reduces the benefits of Hyper-Threading.

Virtual machines should only be allocated a single virtual CPU. From empirical results it appears that Virtual SMP lowers throughput by 10%. Virtual SMP should not be used unless the problem domain requires it.

Allocating excessive memory did not increase the performance of virtual machines. When a benchmark isolated the memory subsystem and was run on a virtual machine running Linux allocating too much memory lowered throughput between 6%-12%. In contrast allocating excessive memory to virtual machines running Windows XP SP2 did not affect performance for any benchmark we ran. Virtual machines should not be allocated more resources than they are expected to use for their current task because it will not increase performance and there may be hidden costs for managing increased amounts of resources.

In summary:

- Single CPU virtual machines scale better than virtual machines using Virtual SMP.
- Hyper-Threading increases throughput if there are a large number of virtual CPUs, but makes no difference if the number of virtual CPUs is less than or equal to the number of physical CPUs.
- Do not allocate excessive resources to virtual machines. Additional resources may hurt performance.

Abstract

We present a summary of our evaluation of VMWare ESX Server 2.5.2. In particular we confirm and work around known timing issues with guest operating systems running on ESX server. Our work validates and adds to the work of other groups modeling the behavior of ESX Server during CPU intensive workloads by exploring in more detail the effects of Hyper-Threading and the overhead of Virtual SMP. We report and measure a previously unknown performance penalty for allocating too much RAM in virtual machines with *Linux* as the guest operating system. This paper also describes the testbed we used to manage and run our tests including a virtualization test management system we developed to run the tests we performed. We describe timing issues that affect performance testing on ESX Server and a method for measuring runtimes that gives accurate results.

Introduction

A *virtual machine monitor* (VMM) is a piece of software that provides an environment that attempts to closely simulate physical hardware. A *virtual machine* (VM) is the environment created by a VMM. With the exception of timing dependencies any program run inside a VM should exhibit the same behavior as the program would exhibit if it were run on physical hardware. Some reasons why virtualization is used by corporate IT departments include:

- Increased hardware utilization.
- Increased hardware scalability.
- Better hardware fault containment.
- Better isolation of users, programs, and physical resources.
- Allowing the transparent migration of a server between physical hosts.

In this paper we explore how configuration changes affect the throughput of virtual machines during CPU intensive workloads. In particular we explore how Hyper-Threading, Virtual SMP, and the amount of RAM allocated affect throughput.

In our work we will use VMware ESX Server 2.5.2 as the VMM.

Background

Hyper-Threading is Intel’s implementation of simultaneous multithreading technology and was first introduced with Intel’s Xeon processor. Hyper-Threading allows the processor to use execution units that are normally unused (such was when the processor is waiting because of a cache miss). The actual performance improvement is application dependent (Intel).

ESX Server is a *native VM system*. A native VM system is one where the VMM is the only software on the machine that runs in the highest privilege level of the host machine. In contrast a VMM that is installed on a host that runs an operating system independent of the VMM is called a *hosted VM system*. If the VMM on a hosted system runs in a privilege level below the host’s operating system it is called a *user-mode hosted VM system*. VMware Server and Microsoft Virtual PC are two examples of user-mode hosted VM systems. VMware claims that since ESX Server runs directly on the hardware I/O performance is significantly higher on ESX Server than on user-mode hosted VM systems (Waldspurger 2002; Smith, 2005).

VMware runs unmodified guest operating systems. Paravirtulization achieves higher performance than traditional VM systems by presenting an interface that is similar, but not identical to the underlying hardware. The changes are intended to make virtualization more efficient, but they also require that guest operating systems be rewritten to only use the new interface. Xen and Denali are two well-known paravirtaulization systems (Barham 2003; Whitaker 2002).

Virtual SMP is an extension of ESX Server created by VMware that allows ESX Server to create and host virtual machines that are allocated two processors. If Virtual SMP is installed ESX Server allows guest virtual machines to be allocated two virtual CPUs.

Related Work

VMware published a white paper that explores the behavior of ESX Server under loads similar to what we consider here (VMware 2005A). Results from the white paper include:

- Guaranteed CPU resource minimums and maximums work as advertised.
- Page sharing has negligible CPU overhead and can result in significant memory savings.
- Hyper-Threading increases throughput of single CPU virtual machines under CPU intensive workloads.
- A native operating system has throughput approximately 12%-14% higher than the same operating system running inside a virtual machine hosted by ESX Server.

VMware and a group from IBM reported on the suitability of using ESX Server to host IBM's WebSphere application server (High-Volume Web Site 2004). The group's results include the following:

- ESX Server 2 effectively allocates CPU shares according to priorities set by users.
- The overhead of ESX Server compared to an operating system running directly on hardware is between 12%-15%.

Hardware and System Setup

Hardware

The machines we used to run most of our tests were two Dell 6650 quad processor 2.20GHz Xeon servers with 16GB of RAM, a Broadcom NetXtreme BCM5700 Gigabit Ethernet NIC, and an EMC CLARiiON CX 300 storage area network with over 1.3 TB of storage. The virtualization testbed is shown in Figure 1. A diagram showing the interconnections is shown in Figure 2.

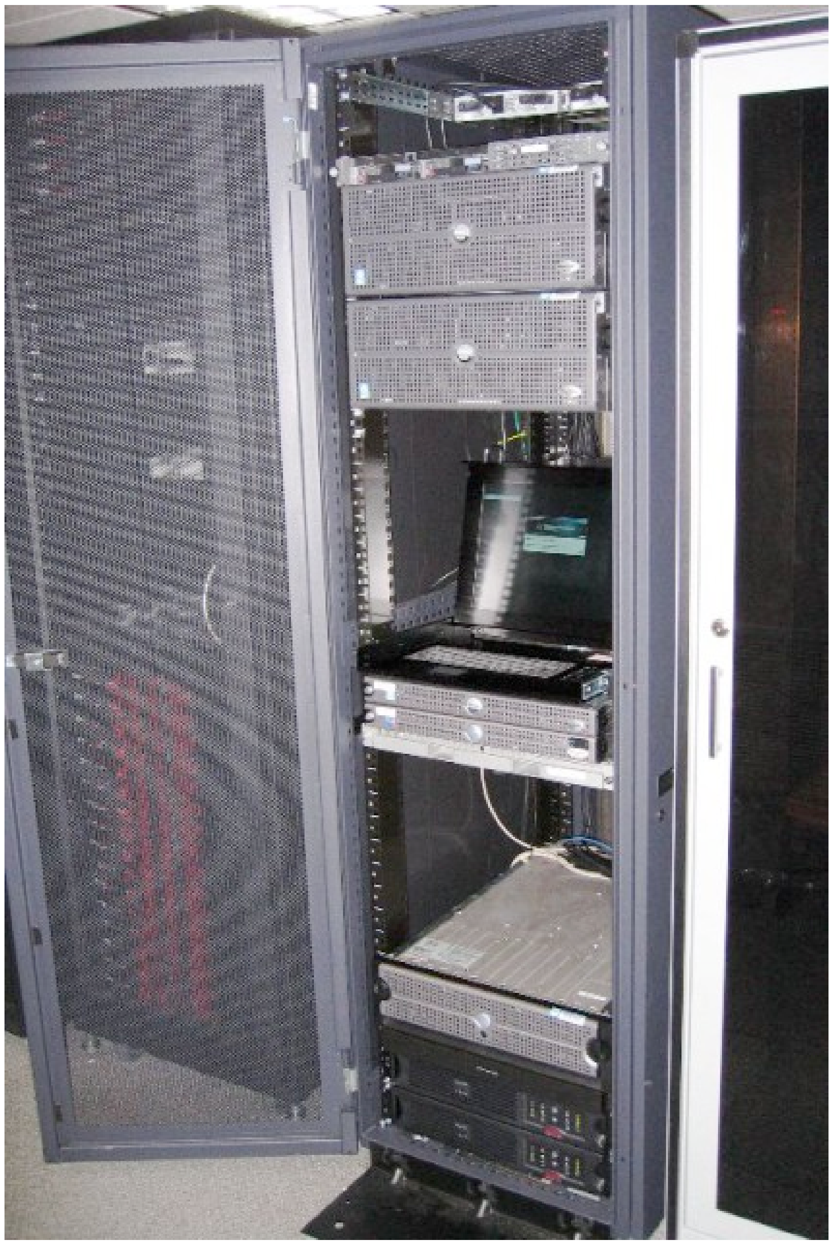


Figure 1 – The Enterprise Computing Laboratory Virtualization Testbed

A small number of tests were run on a ProLiant BL25p with two Opteron Processors 250 which ran at 2406 MHz. This machine had 16GB of RAM.

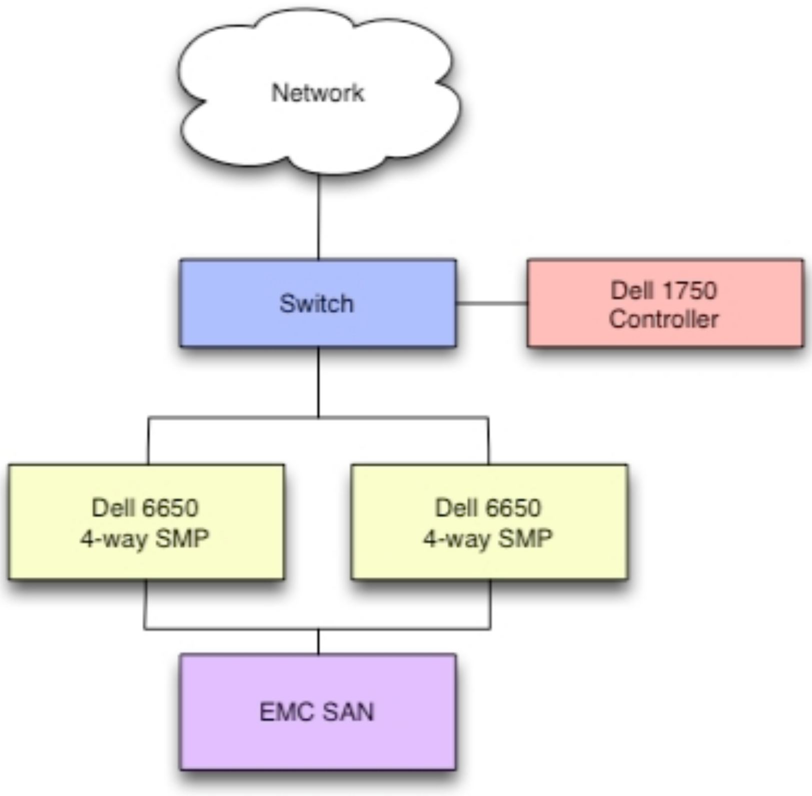


Figure 2 – The environment the majority of our tests were run involved two Dell 6650's connected to a SAN and controlled through a Dell 1750.

Operating Systems

We ran our tests with various guest operating systems, but the results presented in this paper used Linux kernel 2.4.21-15.ELsmp (hereafter referred to as *Linux*) and compiled using gcc 3.2.3 unless specifically stated otherwise. This operating system was selected because it is supported by Virtual SMP allowing us to allocate one or two CPUs to a single virtual machine.

Benchmark Methodology

The performance issues we explore in this paper focus on CPU resources and so the benchmarks are CPU intensive. We present here results from using the following benchmark suites:

- FreeBench
- Spec cpu2000

FreeBench was used to help gather initial results due to its short runtimes. Spec cpu2000 was used to gather most of the results presented here (FreeBench; Henning 2000).

No attempt was made to optimize the benchmarks for the virtual machines and default settings were used in all cases. Unused virtual machines were suspended before each test so that they wouldn't affect CPU usage. Each benchmark was run five times to ensure that the results from the benchmarks were consistent and repeatable. The reported results are the average of the test runs.

More detailed information about the benchmarks we used can be found in Appendix A.

Timing Methodology

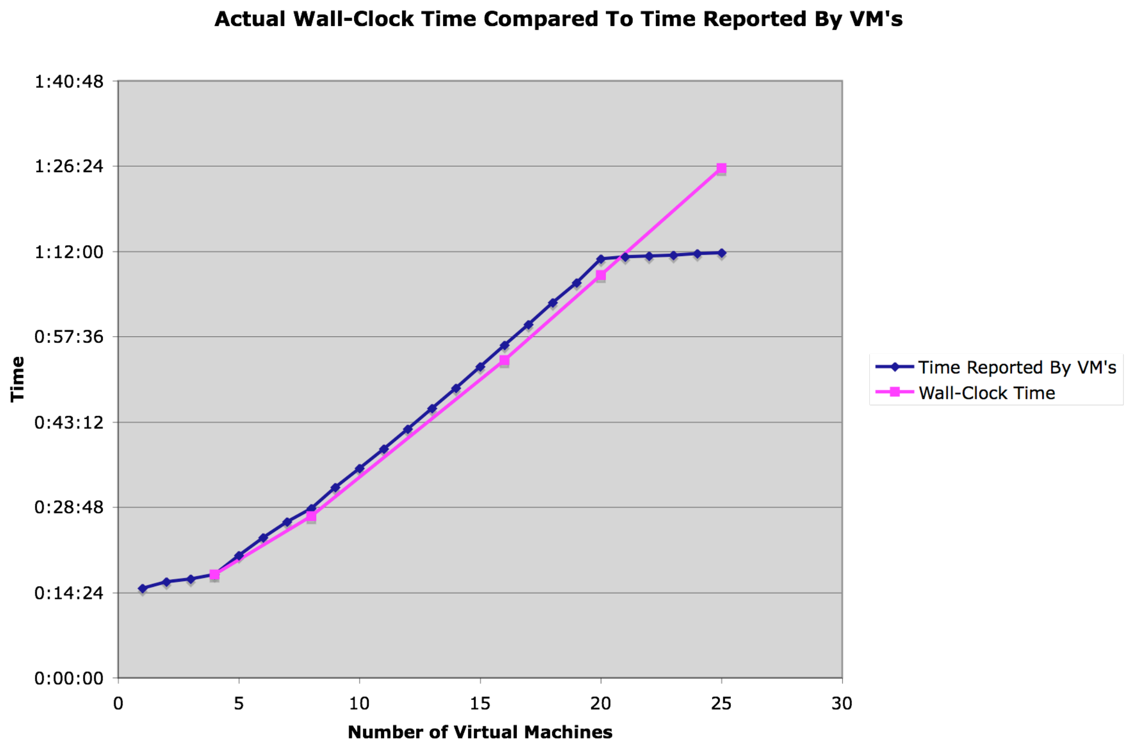


Figure 3 – Comparison of time reported by virtual machines after a benchmark run compared with the actual amount of time that had passed. The guest operating system was *Linux* running the *FreeBench* benchmark.

Initial benchmark runs demonstrated abnormal results under certain conditions. When a small number of virtual machines were running the results of the benchmarks were consistent with timers external to the virtual machines. For large numbers of virtual machines, however, the times reported by the virtual machines did not agree with our external timers. Figure 3 illustrates the difference between reported time and wall-clock time for a benchmark run using *Linux.* Notice that reported time and wall-clock time are nearly identical until 20 virtual machines are running simultaneously and when more than

20 virtual machines are running simultaneously the time reported by the virtual machines stops increasing.

This is a known issue with some operating systems running on ESX Server (VMware 2005B). The affected operating systems are not processing their timing interrupts as fast as ESX server sends them causing those interrupts to be lost. VMware provides a set of tools that periodically synchronizes the guest operating systems time with the host machine. Rerunning the benchmarks with this tool showed the same pattern, as the tool did not synchronize the time sufficiently frequently to affect our benchmarks.

We timed our benchmarks using a method introduced in a VMware white paper (VMware 2005A). We used the **RDTSC** instruction to time workloads. **RDTSC** is a benchmarking instruction that returns the value of a 64-bit register (the **TSC**) that is incremented every clock cycle. We also used the **RDTSC** instruction and modified the virtual machines to use the host machine's **TSC** so that our benchmark results would be independent of timing issues on all operating systems.

The **TSC** was read at the beginning and end of each benchmark run. The difference between the two values is used to determine how long the benchmark ran.

We use CPU throughput of the system to report our results. We have the virtual machines perform a workload and throughput is measured as the number of times the workload is completed by the system for a given time period.

Benchmark Test Cases

The test cases covered in this paper are summarized in the table below.

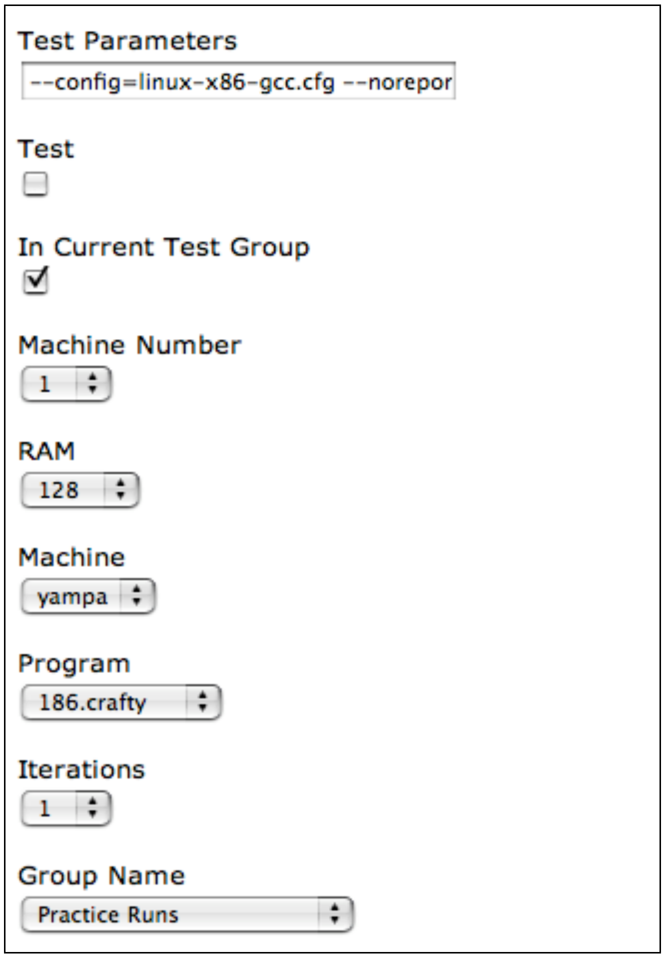
Test Cases	Description
Scalability	
Hyper-Threading, 1 CPU	This compares throughput of one CPU virtual machines with Hyper-Threading both enabled and disabled.
Hyper-Threading, 2 CPU	This compares the throughput of two CPU virtual machines with Hyper-Threading both enabled and disabled.
1 CPU vs. 2 CPUs	This compares the throughput of one and two CPU virtual machines.
Resource Allocation	
Memory Subsystem	This tracks throughput as the amount of RAM allocated to a virtual machine increases.

Table 2 - Summary of benchmark test cases.

The scalability tests measure throughput as the number of active virtual machines increases. The resource allocation test measures throughput as the amount of resources (memory) allocated changes.

Testing Framework

We managed our tests using a distributed application created using Ruby on Rails. Figure 4 shows a screenshot of the test management software.



Test Parameters
--config=linux-x86-gcc.cfg --norepor

Test
☐

In Current Test Group
☑

Machine Number
1

RAM
128

Machine
yampa

Program
186.crafty

Iterations
1

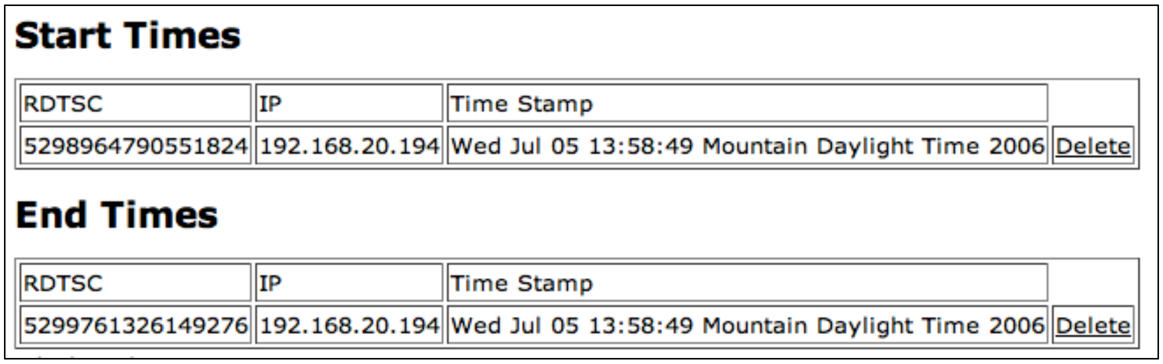
Group Name
Practice Runs

Figure 4 – Screenshot of the program we used to manage our tests. The virtual machines contact the Rails application to get test parameters and to report results.

Tests are setup by entering the test parameters into the application. The parameters include information about the environment (such as how much RAM is allocated) and the test itself (such as which benchmark to run).

A script is run on each virtual machine involved in the test. The virtual machine contacts the test management system and gets the information about which benchmarks to run.

Our test management system greatly reduced the time required to setup and collect the results from our tests. Starting each test by hand was prone to error because the commands to start the tests are long and must be typed in perfectly for each virtual machine involved in the test. Collecting results by hand was error prone and time consuming. When large numbers of virtual machines were involved in the test collecting the output files and entering them into a spreadsheet often took more than thirty minutes for each test. With our test management system each test took only five minutes to setup and collect the results regardless of the number of virtual machines involved in the test.



Start Times

RDTSC	IP	Time Stamp	
5298964790551824	192.168.20.194	Wed Jul 05 13:58:49 Mountain Daylight Time 2006	Delete

End Times

RDTSC	IP	Time Stamp	
5299761326149276	192.168.20.194	Wed Jul 05 13:58:49 Mountain Daylight Time 2006	Delete

Figure 5 - The results of a test are along with the IP number of the virtual machine to identify which virtual machine submitted the results for tests involving many virtual machines.

The results are reported back to the application at the end of the test run as shown in Figure 5. The IP address of the reporting virtual machine is used to uniquely identify which virtual machine submitted the results for tests that use multiple virtual machines.

The test management application groups the results based on the type of test that was run. The results were then transferred to a spreadsheet for further processing.

CPU Intensive Workloads

This section reports our results for CPU intensive workloads. First we compare the throughput of single CPU virtual machines with Hyper-Threading enabled and disabled. Next we compare the throughput of two CPU virtual machines with Hyper-Threading enabled and disabled. Finally we compare the throughput of single CPU and two CPU virtual machines. The results shown in the graphs and tables below are for the *gzip* benchmark in the cpu2000 suite. We only present the results for *gzip* here because the other results are similar.

The *gzip* benchmark is based on the popular data compression program gzip (GNU zip). It uses Lempel-Ziv coding (LZ77) as its compression algorithm. All work is done in memory to help isolate the CPU and memory subsystem.

The single CPU virtual machines were allocated 512MB of RAM and the two CPU virtual machines were allocated 1024MB of RAM. All other virtual machine resources were left at the default values. Results are reported relative to the throughput of a single virtual machine running with Hyper-Threading disabled. The hardware used to run these tests had four CPUs.

Hyper-Threading Comparison With One Virtual CPU

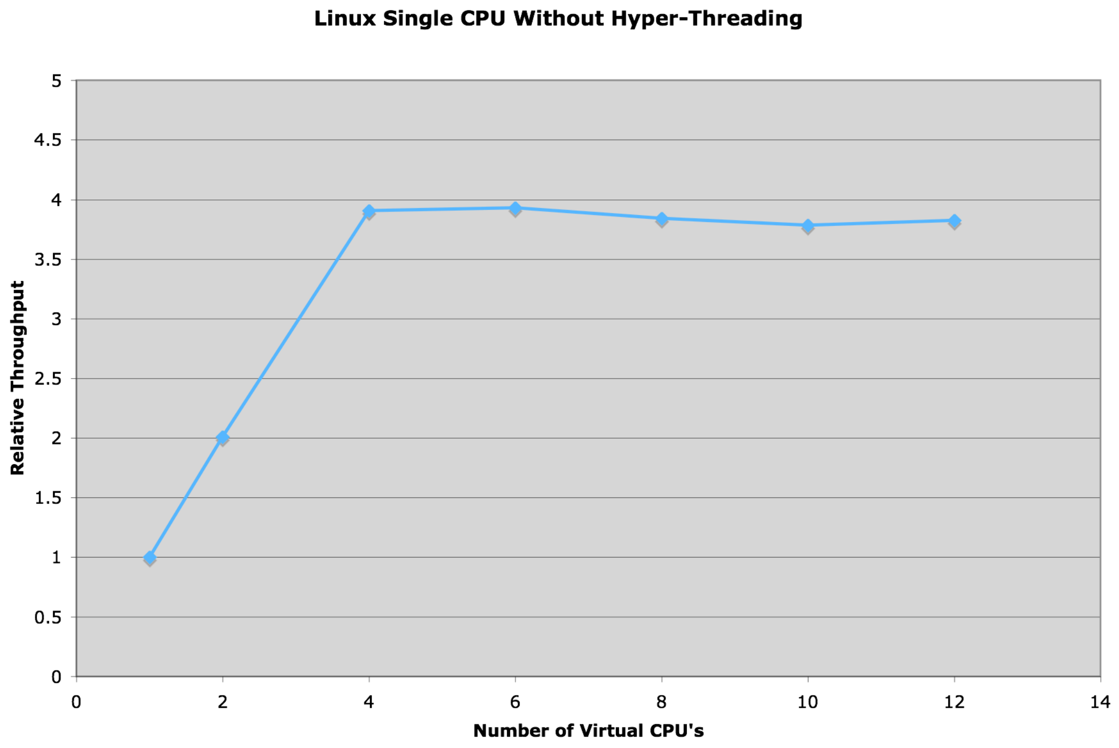


Figure 6 – CPU throughput of virtual machines running Linux and allocated a single CPU with Hyper-Threading disabled for the *gzip* benchmark.

Figure 6 shows the CPU throughput of virtual machines with a single virtual CPU when Hyper-Threading is disabled. With Hyper-Threading disabled throughput increased linearly until four virtual CPUs were running simultaneously. When more than four virtual CPUs were running at the same time throughput decreased marginally. From four virtual CPUs to twelve virtual CPUs throughput decreased 2%. Since each virtual machine is allocated a single virtual CPU the number of virtual machines is equal to the number of virtual CPUs.

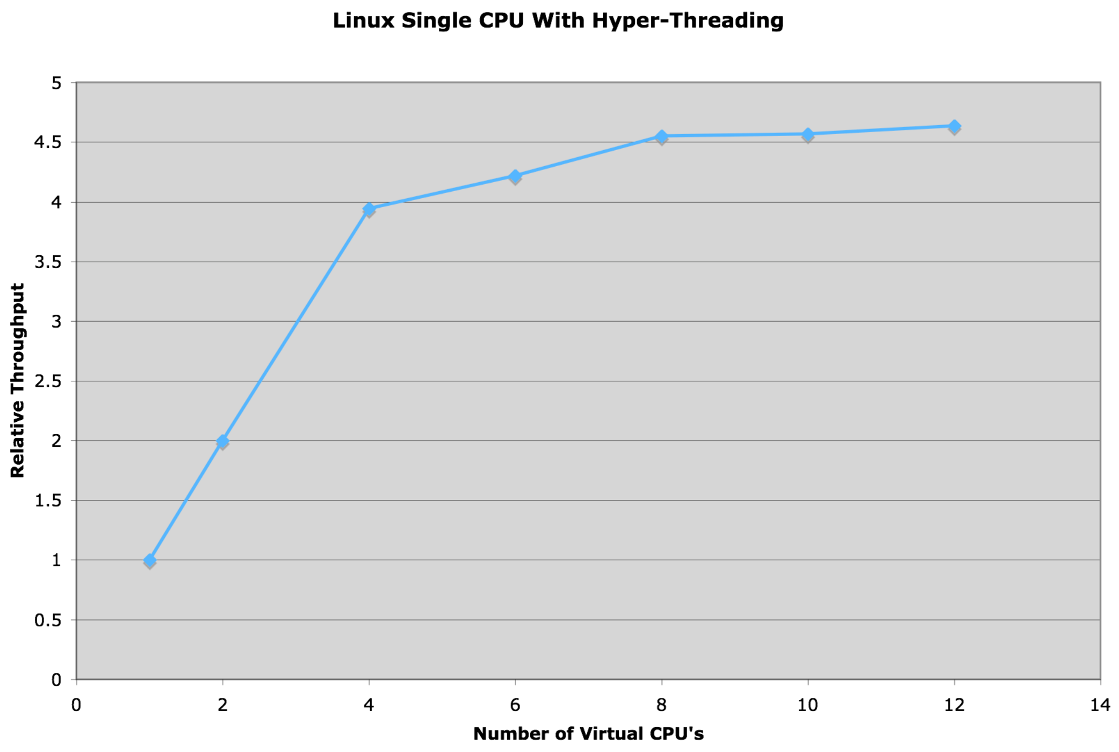


Figure 7 – CPU throughput of virtual machines running Linux and allocated a single virtual CPU with Hyper-Threading enabled for the *gzip* benchmark.

Figure 7 shows the CPU throughput of virtual machines with a single CPU when Hyper-Threading is enabled. With Hyper-Threading enabled throughput increased linearly from one to four and from four to eight virtual CPUs. The increase from four to eight virtual CPUs was more modest than the initial increase. When more than eight virtual CPUs are running simultaneously throughput increases marginally. Throughput increased 17% from four virtual CPUs to twelve virtual CPUs. Since each virtual machine is allocated a single virtual CPU the number of virtual machines is equal to the number of virtual CPUs.

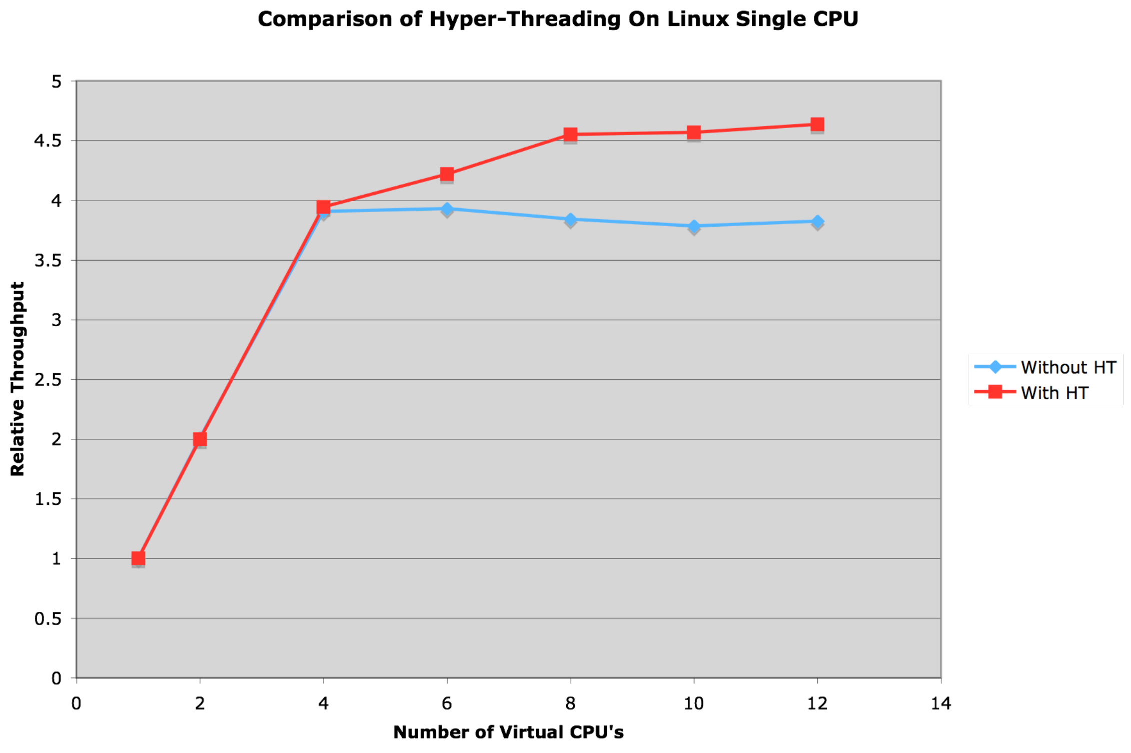


Figure 8 – CPU throughput comparison of virtual machines running Linux and allocated a single virtual CPU with Hyper-Threading enabled and disabled for the *gzip* benchmark.

Figure 8 compares the CPU throughputs of virtual machines allocated a single virtual CPU when Hyper-Threading is enabled and disabled. Notice that from one virtual CPU to four virtual CPUs Hyper-Threading has no impact on throughput. When more than four virtual CPUs are running simultaneously Hyper-Threading enabled virtual machines have higher throughput than Hyper-Threading disabled virtual machines. The throughput of Hyper-Threading enabled virtual machines is 21% higher than the throughput of virtual machines without Hyper-Threading when twelve virtual CPUs are running simultaneously. Since each virtual machine is allocated a single virtual CPU the number of virtual machines is equal to the number of virtual CPUs.

Virtual CPUs	HT Enabled	HT Disabled	Ratio
1	1.00	1.00	1.00
2	2.00	2.01	1.00
4	3.94	3.91	1.01
6	4.22	3.93	1.07
8	4.55	3.84	1.18
10	4.57	3.79	1.21
12	4.64	3.83	1.21

Table 3 – Normalized results for *Linux* running *gzip* with single CPU virtual machines running *Linux* for the *gzip* benchmark.

The throughput increases linearly to four virtual CPUs because the hardware used to run these tests had four CPUs. The throughput at four virtual CPUs was not quite four times the throughput of one virtual CPU because of the overhead of running ESX Server. For large numbers of virtual machines Hyper-Threading enabled has significantly higher throughput for CPU intensive workloads. When Hyper-Threading is enabled having two single CPU virtual machines for every physical CPU in the host machine maximizes throughput. When less virtual machines are running on the host some of the potential benefits of Hyper-Threading are lost.

Hyper-Threading Comparison With Two Virtual CPUs

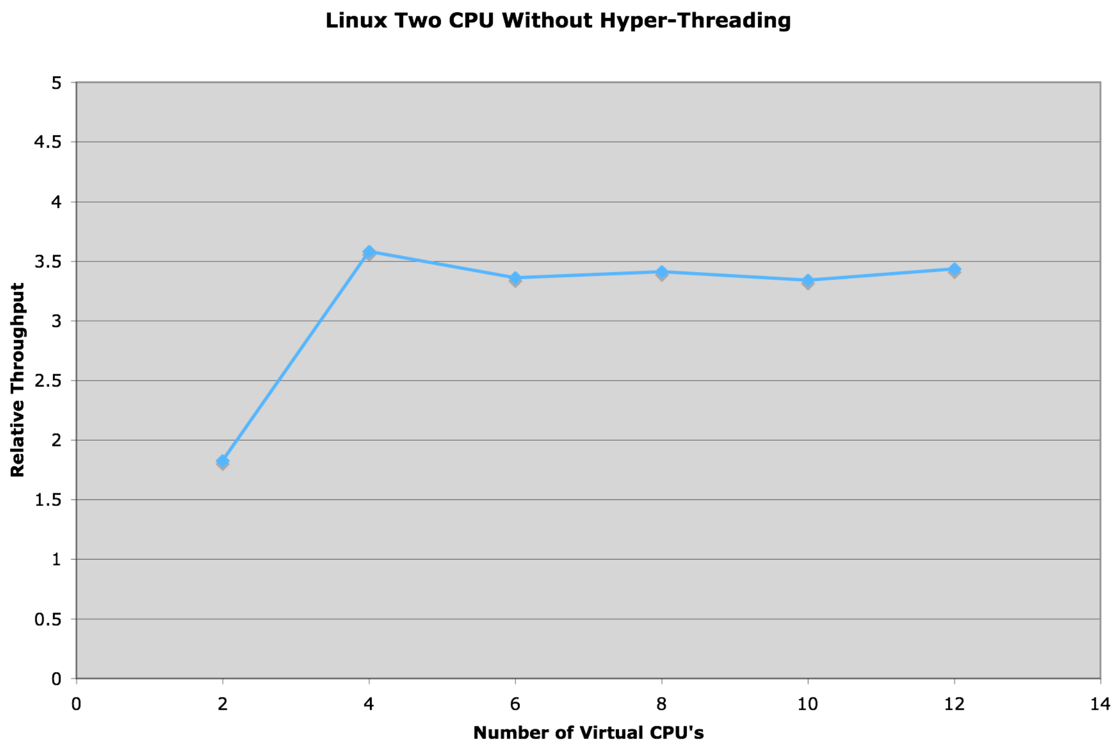


Figure 9 - CPU throughput of virtual machines running Linux and allocated two virtual CPUs with Hyper-Threading disabled for the *gzip* benchmark.

Figure 9 shows CPU throughput of virtual machines with two virtual CPUs when Hyper-Threading is disabled. With Hyper-Threading is disabled throughput increased linearly until four virtual CPUs were running simultaneously. When more than four virtual CPUs were running at the same time throughput decreased slightly. From four virtual CPUs to twelve virtual CPUs throughput decreased 4%. Since each virtual machine was allocated two virtual CPUs the number of virtual machines involved in a test was half the number of virtual CPUs used in that test.

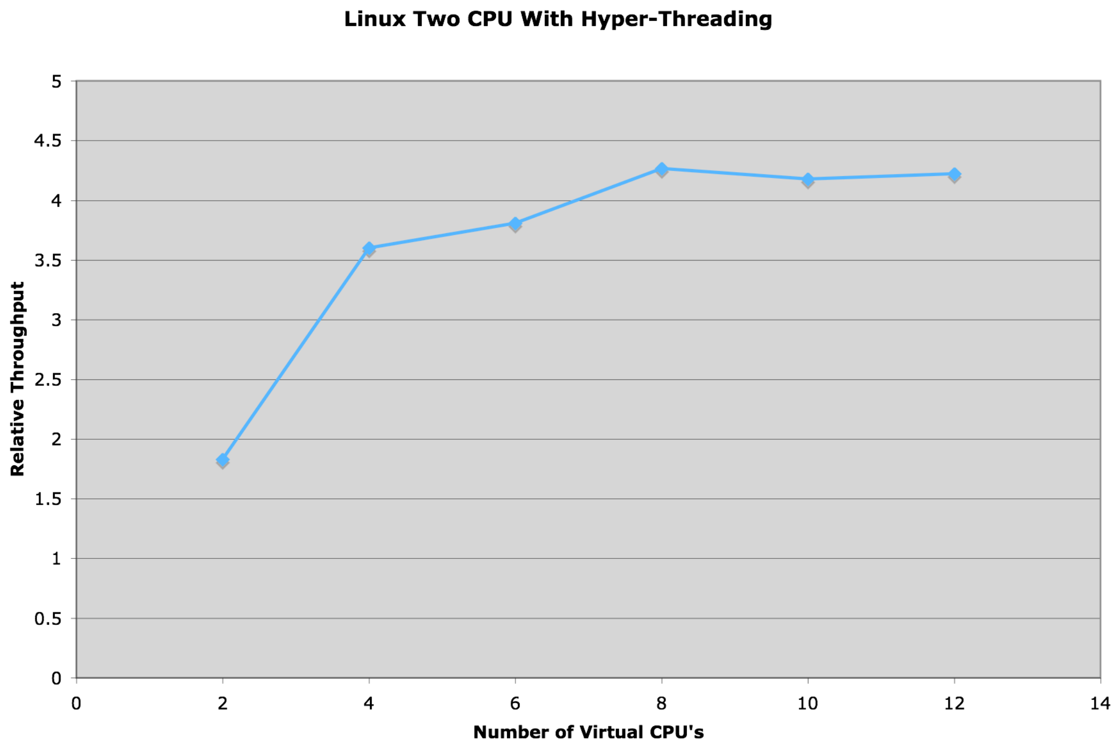


Figure 10 - CPU throughput of virtual machines running Linux and allocated two virtual CPUs with Hyper-Threading enabled for the *gzip* benchmark.

Figure 10 shows the CPU throughput of virtual machines allocated two virtual CPUs when Hyper-Threading is enabled. With Hyper-Threading enabled throughput doubled from two to four virtual CPUs and throughput continued to increase from four to eight virtual CPUs. After eight virtual CPUs were running throughput did not change significantly. Throughput increased 17% from four virtual CPUs to twelve virtual CPUs. Since each virtual machine was allocated two virtual CPUs the number of virtual machines involved in a test was half the number of virtual CPUs used in that test.

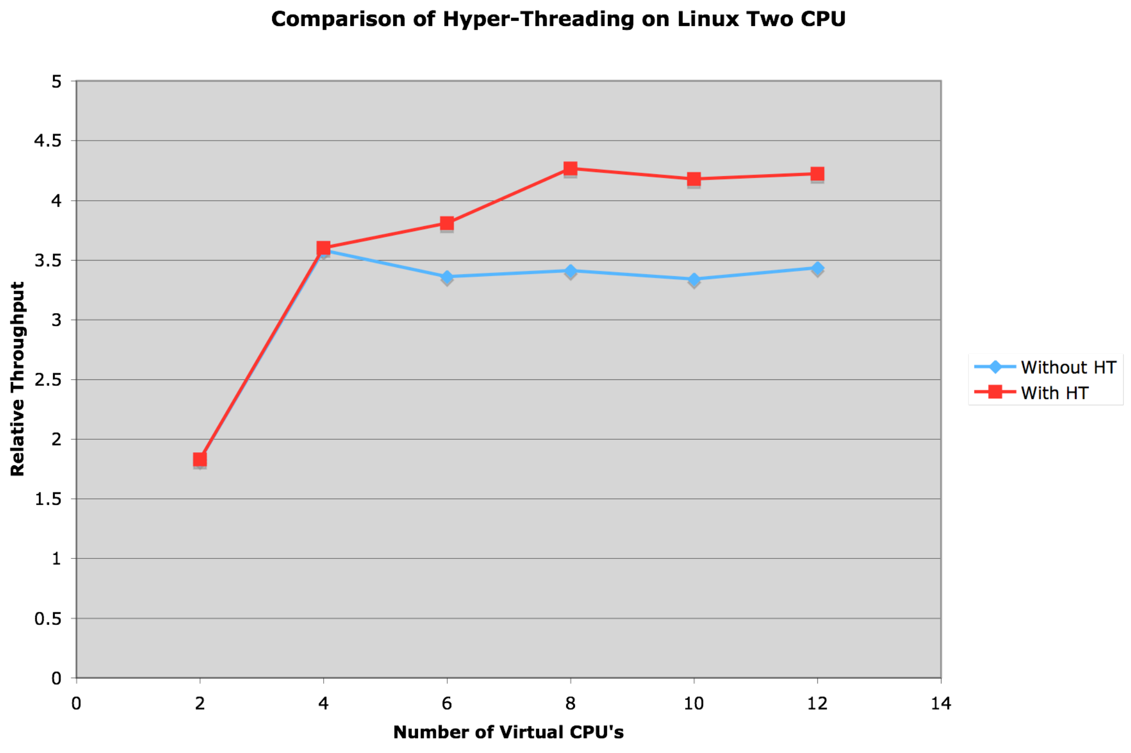


Figure 11 - CPU throughput comparison of virtual machines running Linux and allocated a single virtual CPU with Hyper-Threading enabled and disabled for the *gzip* benchmark.

Figure 11 compares the CPU throughputs of virtual machines allocated two virtual CPUs when Hyper-Threading is enabled and disabled. Notice that when two virtual CPUs or four virtual CPUs are running simultaneously Hyper-Threading has no impact on throughput. When more than four virtual CPUs are running simultaneously Hyper-Threading enabled has higher throughput than Hyper-Threading disabled. The throughput of Hyper-Threading enabled virtual machines is 23% higher than the throughput of virtual machines without Hyper-Threading when twelve virtual CPUs are running simultaneously. Since each virtual machine was allocated two virtual CPUs the number of virtual machines involved in a test was half the number of virtual CPUs used in that test.

Virtual CPUs	HT Enabled	HT Disabled	Ratio
2	1.83	1.83	1.00
4	3.60	3.58	1.01
6	3.81	3.36	1.13
8	4.27	3.41	1.25
10	4.18	3.34	1.25
12	4.22	3.44	1.23

Table 4 - Normalized results for *Linux* running *gzip* with each virtual machine allocated 1024MB of RAM and two virtual CPUs.

The throughput nearly doubles from two virtual CPUs to four virtual CPUs because the hardware used to run these tests had four CPUs. The throughput at four virtual CPUs was not quite double the throughput of two virtual CPUs because of the overhead of running ESX Server. For large numbers of virtual machines Hyper-Threading enabled has significantly higher throughput for CPU intensive workloads. When Hyper-Threading is enabled having one virtual machine with two CPUs for every physical CPU in the host machine maximizes throughput. When less virtual machines are running on the host some of the potential benefits of Hyper-Threading are lost.

Virtual SMP

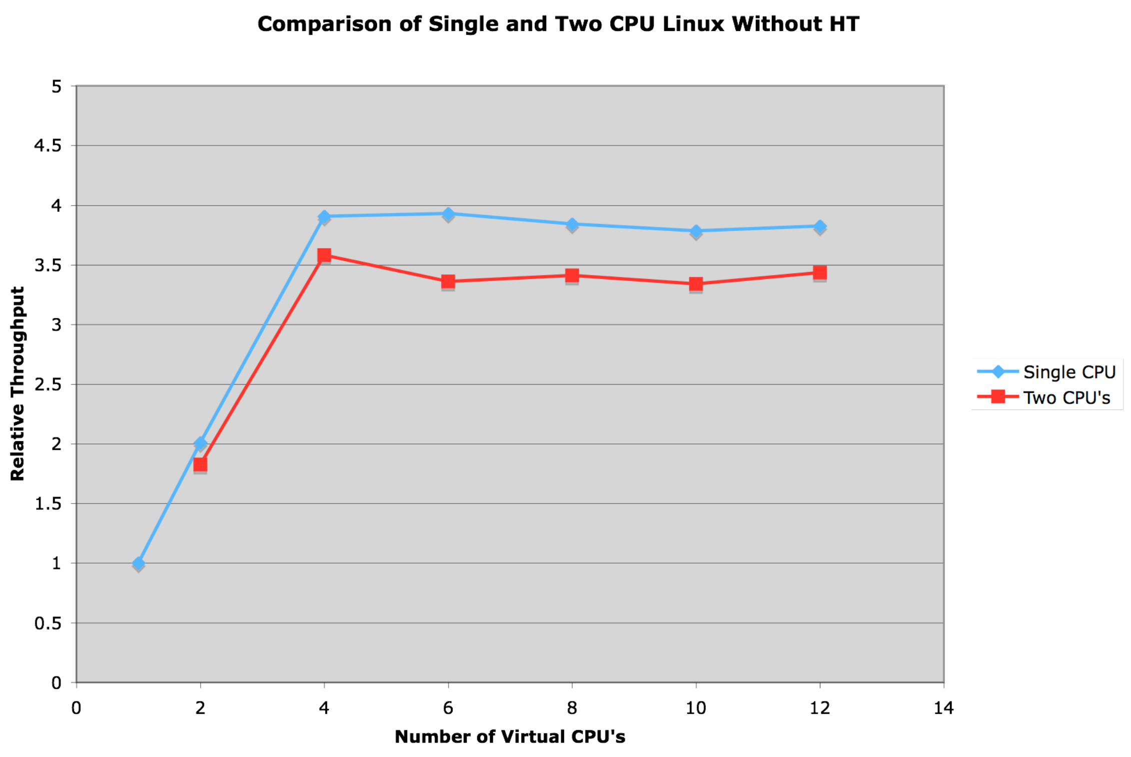


Figure 12 – CPU throughput comparison of single CPU virtual machines and two CPU virtual machines with *Linux* running *gzip*.

Figure 12 compares the CPU throughputs of single CPU virtual machines and virtual machines allocated two virtual CPUs when Hyper-Threading is disabled. As the number of virtual CPUs running simultaneously increase throughput is consistently higher for single CPU virtual machines than for two CPU virtual machines. The throughput difference ranges between 9%-17% where most values are between 10%-13%. The number of single CPU virtual machines involved in a test is equal to the number of virtual CPUs. The number of two CPU virtual machines involved in a test is half of the number of virtual CPUs.

Virtual CPUs	Single CPU	Two CPUs	Ratio
1	1.00	--	--
2	2.01	1.83	1.10
4	3.91	3.58	1.09
6	3.93	3.36	1.17
8	3.84	3.41	1.13
10	3.79	3.34	1.13
12	3.83	3.44	1.11

Table 5 - Normalized results for CPU throughput of single CPU virtual machines and two CPU virtual machines when Hyper-Threading is disabled with *Linux* as the guest OS running *gzip*.

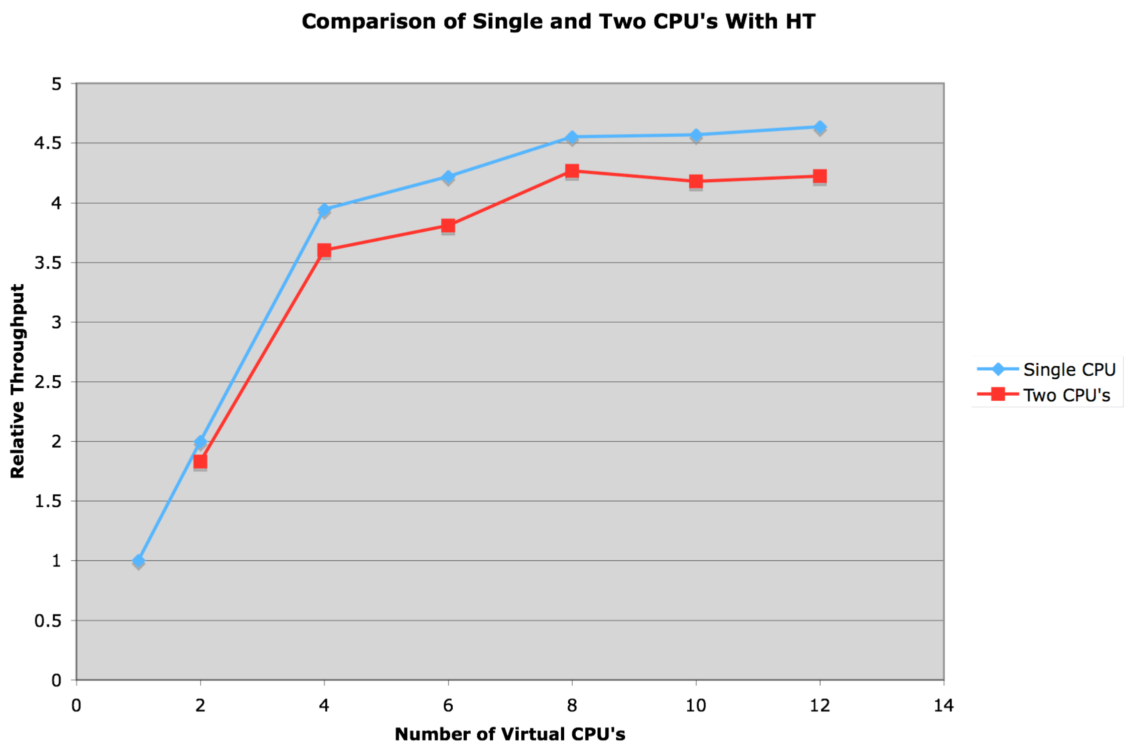


Figure 13 - Comparison of one and two CPU virtual machines when Hyper-Threading is enabled. Each single CPU virtual machine was allocated 512MB of RAM and each two CPU virtual machine was allocated 1024MB of RAM. The guest operating system was *Linux* and the benchmark was *gzip*.

Figure 13 compares the throughput of virtual machines allocated a single virtual CPU and virtual machines allocated two virtual CPUs when Hyper-Threading is enabled. As the number of virtual CPUs running simultaneously increase the gap between the throughput for one CPU virtual machines and two CPU virtual machines goes from 7% to 11%.

Virtual CPUs	Single CPU	Two CPUs	Ratio
1	1.00	--	--
2	2.00	1.83	1.09
4	3.94	3.60	1.10
6	4.22	3.81	1.11
8	4.55	4.27	1.07

10	4.57	4.18	1.09
12	4.64	4.22	1.10

Table 6 - Normalized results for CPU throughput for single CPU virtual machines and two CPU virtual machines when Hyper-Threading is enabled and *Linux* is the guest OS running *gzip*.

Single CPU virtual machines scale better than virtual machines using Virtual SMP. Virtual SMP should not be used unless the problem domain requires the virtual machine to have two processors.

Conclusions from CPU Intensive Workloads

Hyper-Threading significantly increases throughput for CPU intensive workloads. When twelve virtual CPUs were benchmarked and Hyper-Threading was enabled the throughput of single CPU virtual machines increased 21% and the throughput of two CPU virtual machines increased 23% compared to when Hyper-Threading was disabled. Since Hyper-Threading is most effective with large numbers of virtual machines a host that is Hyper-Threading enabled should have at least two virtual CPUs for every physical CPU. Using fewer virtual CPUs reduces the benefits of Hyper-Threading.

Virtual CPUs	Improvement for Single CPU Virtual Machines	Improvement for Two CPU Virtual Machines
1	1.00	--
2	1.00	1.00
4	1.01	1.01
6	1.07	1.13
8	1.18	1.25
10	1.21	1.25
12	1.21	1.23

Table 7 – The improvement of CPU throughput when Hyper-Threading is enabled compared to when Hyper-Threading is disabled with *Linux* as the guest operating system running the *gzip* benchmark.

Virtual machines should only be allocated one CPU. From empirical results it appears that Virtual SMP lowers throughput by 10%. Virtual SMP should not be used unless the problem domain requires it.

Virtual CPUs	Improvement With Hyper-Threading Disabled	Improvement With Hyper-Threading Enabled
1	--	--
2	1.10	1.09
4	1.09	1.10
6	1.17	1.11
8	1.13	1.07
10	1.13	1.09
12	1.11	1.10

Table 8 - The ratio of CPU throughput of single CPU virtual machines compared to the CPU throughput of virtual machines with two virtual CPUs.

Memory

This section reports our results for the memory subsystem workloads. We ran the SPEC cpu2000 benchmarks on virtual machines hosting *Linux* and Windows XP SP2 to compare throughput as a function of the amount of memory allocated to the virtual machine. Hyper-Threading had negligible effect on these tests. The virtual machines used in these tests were only allocated one virtual CPU.

Linux Memory Subsystem Test

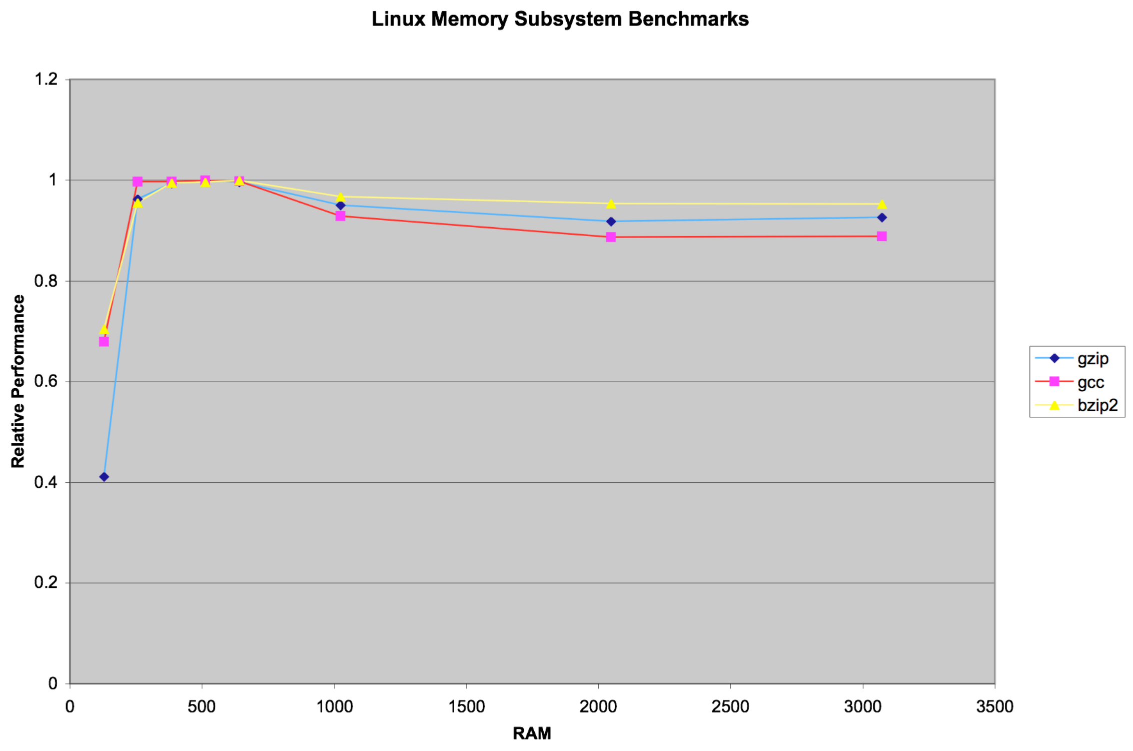


Figure 14 – Comparison of throughput of virtual machines running *Linux* as a function of RAM.

Figure 14 shows the CPU throughput as a function of RAM of the three benchmarks in the cpu2000 suite that best isolate the memory subsystem with *Linux* as the guest operating system. Notice that they had noticeable drops in throughput as the amount of RAM allocated to the virtual machine increased. The throughput of the three benchmarks that are described as exercising the memory subsystem in the cpu2000 suite dropped between 5%-11%.

RAM	gzip	gcc	bzip2
128	0.411	0.680	0.704
256	0.963	0.997	0.955
384	0.994	0.997	0.995
512	1.000	1.000	0.996
640	0.996	0.998	1.000

1024	0.951	0.929	0.967
2048	0.918	0.887	0.954
3072	0.926	0.889	0.953

Table 9 – Normalized results for *Linux* as a function of RAM. The benchmarks shown here are the ones from the integer component of the cpu2000 suite that best isolate the memory subsystem.

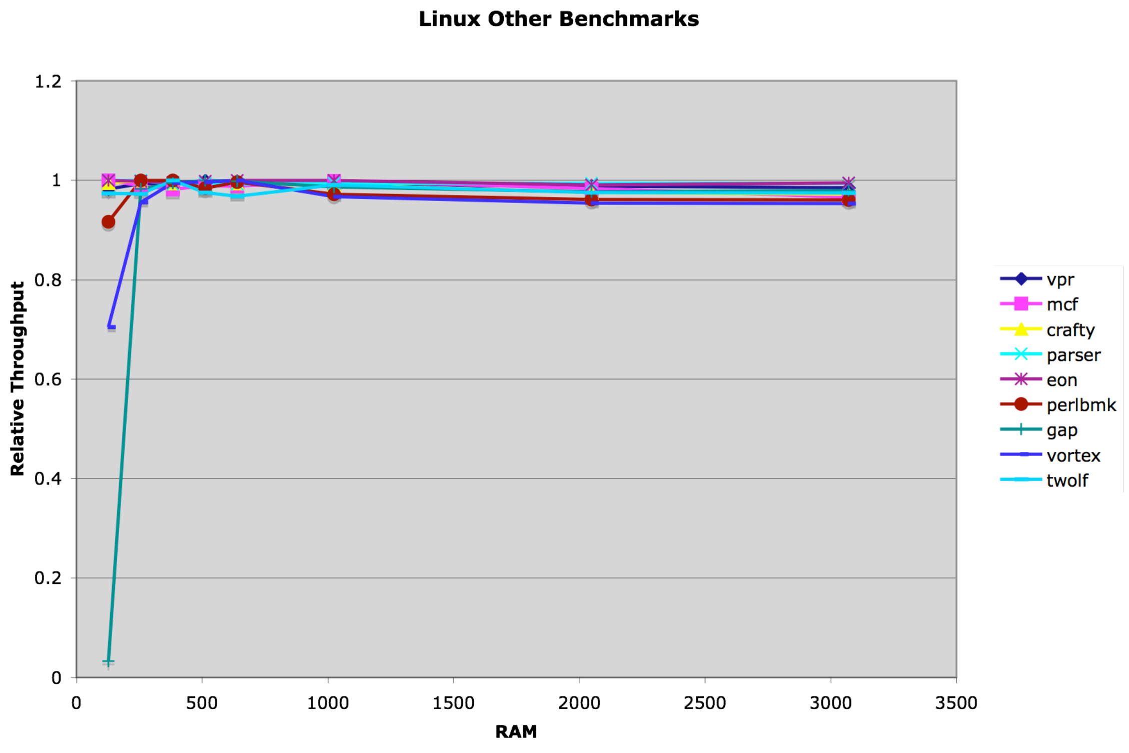


Figure 15 - Comparison of throughput of virtual machines running *Linux* as a function of RAM.

The other benchmarks in the cpu2000 suite are shown in figure 15. As before the guest operating system is *Linux*. The benchmarks that do not isolate the memory subsystem only have a mild drop in performance. The biggest performance drop reported among the group is 4% with the average only being 2%.

RAM	vpr	Mcf	crafty	parser	eon	perlbmk	gap	vortex	twolf
128	0.982	1.000	0.995	1.000	1.000	0.916	0.033	0.989	0.973
256	0.993	0.989	1.000	1.000	0.998	1.000	0.983	1.000	0.973
384	0.992	0.982	0.994	0.992	0.996	1.000	0.994	0.999	1.000
512	1.000	0.989	0.993	0.995	0.998	0.984	1.000	0.993	0.975
640	0.995	0.988	0.994	0.992	1.000	0.996	0.998	0.996	0.968
1024	0.990	0.999	0.992	0.993	1.000	0.972	0.987	0.974	0.991
2048	0.988	0.983	0.975	0.994	0.991	0.961	0.977	0.967	0.976
3072	0.985	0.966	0.972	0.994	0.994	0.961	0.979	0.964	0.975

Table 10 - Normalized results for *Linux* as a function of RAM. The benchmarks shown here are the ones from the integer component of the cpu2000 suite that best isolate the memory subsystem.

Windows XP Memory Subsystem Test

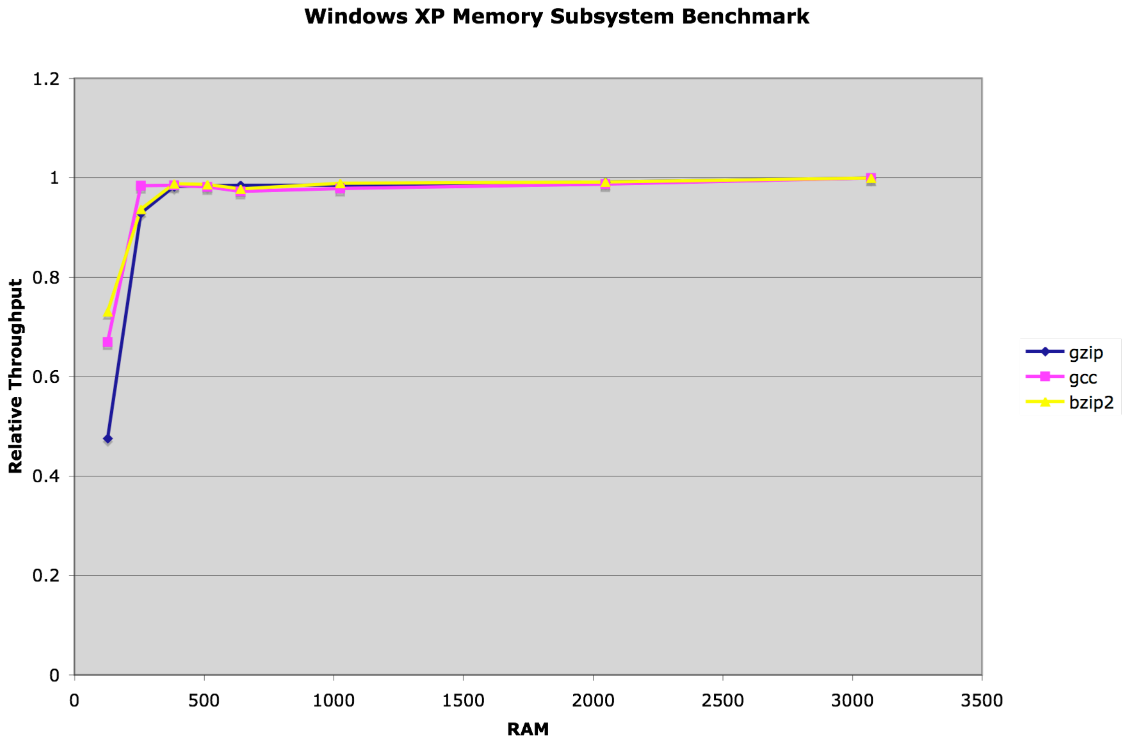


Figure 16 - Comparison of throughput of virtual machines running Windows XP SP2 as a function of RAM.

Figure 16 shows the CPU throughput of three benchmarks in the cpu2000 suite which best isolate the memory subsystem on Windows XP SP2. In contrast to virtual machines running *Linux,* virtual machines running Windows XP SP2 suffered no lose in throughput as the amount of RAM allocated to the virtual machines increased for benchmarks that isolate the memory subsystem.

RAM	gzip	gcc	bzip2
128	0.476	0.669	0.731
256	0.929	0.984	0.937
384	0.983	0.985	0.988
512	0.984	0.982	0.987
640	0.985	0.972	0.977
1024	0.985	0.979	0.989
2048	0.992	0.988	0.992
3072	1.000	1.000	1.000

Table 11 - Normalized results for Windows XP Service Pack 2 as a function of RAM. The benchmarks shown here are the ones from the integer component of the cpu2000 suite that best isolate the memory subsystem.

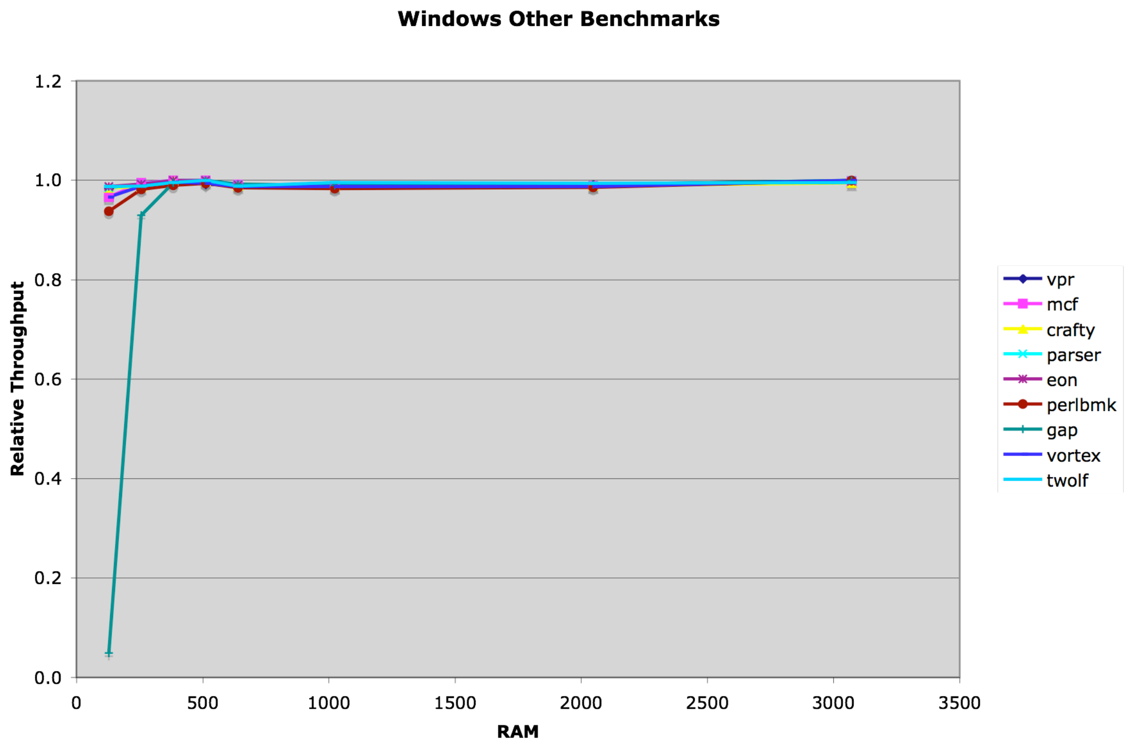


Figure 17 - Comparison of throughput of virtual machines running *Linux* as a function of RAM.

The rest of the benchmarks in the cpu2000 suite with Windows XP SP2 as the guest operating system are shown in Figure 17. Like the benchmarks that isolate the memory subsystem, none of the other benchmarks in the *cpu2000* suite suffered a drop in throughput as the amount of RAM allocated to the virtual machines increased.

RAM	vpr	mcf	crafty	parser	eon	perlbmk	gap	vortex	twolf
128	0.984	0.966	0.985	0.985	0.987	0.937	0.049	0.966	0.988
256	0.991	0.995	0.987	0.993	0.992	0.982	0.929	0.988	0.988
384	1.000	1.000	1.000	1.000	1.000	0.990	0.995	0.996	0.995
512	0.992	1.000	0.998	1.000	1.000	0.994	1.000	0.994	1.000
640	0.989	0.991	0.989	0.991	0.991	0.985	0.992	0.988	0.988
1024	0.986	0.988	0.988	0.990	0.989	0.983	0.990	0.987	0.995
2048	0.991	0.989	0.989	0.990	0.989	0.986	0.991	0.987	0.994
3072	1.000	0.999	0.995	1.000	0.998	1.000	1.000	1.000	0.995

Table 12 - Normalized results for Windows XP SP2 as a function of RAM. The benchmarks shown here are the ones from the integer component of the cpu2000 suite that best isolate the memory subsystem.

Allocating excessive memory did not increase the performance of virtual machines. When the benchmark isolated the memory subsystem and was run on a virtual machine running Linux allocating too much memory lowered throughput between 6%-11%. In contrast allocating excessive amounts of memory to virtual machines running Windows XP SP2 did not affect performance for any benchmark we ran. Virtual machines should not be allocated more resources than they are expected to use for their current task because it will not increase performance and there may be hidden costs for managing increased amounts of resources.

Conclusions

Hyper-Threading significantly increases throughput for CPU intensive workloads. When twelve virtual CPUs were benchmarked and Hyper-Threading was enabled the throughput of single CPU virtual machines increased 21% and the throughput of two CPU virtual machines increased 23% compared to when Hyper-Threading was disabled. Since Hyper-Threading is most effective with large numbers of virtual machines a host that is Hyper-Threading enabled should have at least two virtual CPUs in aggregate for every physical CPU. Using fewer virtual CPUs reduces the benefits of Hyper-Threading.

Virtual machines should only be allocated one CPU. From empirical results it appears that Virtual SMP lowers throughput by 10%. Virtual SMP should not be used unless the problem domain requires it.

Allocating excessive amounts of memory did not increase the performance of virtual machines. When the benchmark isolated the memory subsystem and was run on a virtual machine running Linux allocating too much memory lowered throughput between 6%-12%. In contrast allocating excessive amounts of memory to virtual machines running Windows XP SP2 did not affect performance for any benchmark we ran. Virtual machines should not be allocated more resources than they are expected to use for their current task because it will not increase performance and there may be hidden costs for managing increased amounts of resources.

In summary:

- Single CPU guest machines scale better than guest machines using virtual SMP.
- Hyper-Threading increases throughput if there are a large number of virtual CPUs, but makes no difference if the number of virtual CPUs is less than or equal to the number of physical CPUs.
- Do not allocate excessive resources to virtual machines. The additional resources may hurt performance.

References

Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. 2003. "Xen and the Art of Virtualization," *In Proceedings of the ACM Symposium on Operating Systems Priciples*, pp. 164-177.

FreeBench. *FreeBench.org*, http://www.freebench.org/

Henning, John. 2000. "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *COMPUTER* (May), pp 28-35.

High-Volume Web Site, and VMware. 2004. *VMware ESX Server 2: Performance and Scalability Evaluation,* ftp://ftp.software.ibm.com/software/dw/wes/hvws/esxserver_evaluation.pdf.

Intel. *Hyper-Threading Technology,* http://www.intel.com/technology/hyperthread/

Smith, J. E., and R. Nair. 2005. *Virtual Machines: Versatile Platforms for Systems and Processes,* Moran Kaufmann Publishers, CA.

VMware. 2005A. *ESX Server Performance and Resource Management for CPU-Intensive Workloads*, http://www.vmware.com/pdf/ESX2_CPU_Performance.pdf.

VMware. 2005B. *TimeKeeping in VMware Virtual Machines,* http://www.vmware.com/pdf/vmware_timekeeping.pdf.

Waldspurger, Carl. 2002. Memory Resource Management in VMware ESX Server, *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*.

Whitaker, A., M. Shaw, and S. Gribble. 2002. "Denali: Lightweight Virtual Machines for Distributed and Networked Applications," *In Proceedings of the USENIX Annual Technical Conference*.

Appendix A: Benchmarks

FreeBench

Overview

FreeBench is an open-source benchmark. It is totally free and it works on UNIX variants and Windows systems. The FreeBench project tries to address three issues:

- Openness
- Good balance
- Platform independence

Openness is achieved by the benchmark being open-source. Platform independence is somewhat achieved because the benchmark will run without modification on most Windows and Linux systems.

FreeBench consists of four integer programs and three floating point programs. The programs are a good mix of CPU intensive and memory intensive programs. The integer programs are: Analyzer, FourInARow, Mason, and pCompress2. The floating point programs are: PiFFT, DistRay, and Neural.

Analyzer (Integer)

The program was made to analyze memory traces for data dependence. This program is mainly limited by memory system performance, but that can be hidden by large caches, smart compilers, and out-of-order processors.

FourInARow (Integer)

This program plays the game "four in a row" with itself and it is memory limited by the memory system.

Mason (Integer)

This program solves a puzzle and is limited by clock frequency.

pCompress2 (Integer)

This program compresses a file in three stages: Burrows Wheeler blocksorting, run length encoding, and arithmetic coding. The program is memory intensive, but is also dependent on the efficiency of the C library used by the compiler.

PiFFT (Floating Point)

This program calculates PI to four million decimal points. It is limited by the efficiency of floating point calculations and memory intensive.

DistRay (Floating Point)

This program is a ray tracer using random ray distribution. Most of the time of this program is spent in recursive loops do floating-point arithmetic so this program should be CPU bound.

Neural (Floating Point)

This program trains a neural network to do character recognition. It is memory intensive so having quick memory accesses is important for good performance on this benchmark.

SPEC cpu2000

Overview

This is the current gold standard by which other CPU benchmarks are compared. The Standard Performance Evaluation Corporation (SPEC) has a long history of creating quality benchmarks. The current version of SPEC's CPU benchmark uses only programs that were developed from real user applications and can be used to measure the performance of the processor, memory and compiler on the tested system.

The CPU2000 benchmark contains two suites and a total of 26 programs. All of our tests used the integer suite (as opposed to the floating point suite) mainly because of the time required to run the tests.

The programs in the integer suite are: 164.gzip, 175.vpr, 176.gcc, 181.mcf, 186.crafty, 197.parser, 252.eon, 253.perlbmk, 254.gap, 255.vortex, 256.bzip2, 300.twolf.

164.gzip

This program is the popular data compression program gzip (GNU zip). It uses Lempel-Ziv coding (LZ77) as its compression algorithm. All work is done in memory to help isolate the CPU and memory subsystem.

175.vpr (Versatile Place and Route)

This program performs placement and routing in Field-Programmable Gate Arrays. It starts with random initial positions for the gate arrays and then tries to improve on those positions through small perturbations.

176.gcc

This benchmark uses gcc Version 2.7.2.2 to create assembly code files for a Motorola 88100. The inlining heuristics have been altered slightly to cause the program to spend more time doing analysis of the source code.

181.mcf

This program solves a combinatorial optimization problem. Namely, it solves scheduling problems for a single-depot vehicle in public mass transportation. The main work of th program is integer and pointer arithmetic.

186.crafty

This is a high performance Computer Chess program that is designed around 64-bit words. It has a significant number of logical operations and is a good program for comparing integer/branch prediction/pipe-lining facilities of a processor.

197.parser

This is a parser of English grammar that uses link grammar, an original theory of English syntax. The program creates a syntactic structure which links pairs of words when given a sentence.

252.eon

This program is a probabilistic ray tracer based on Kajiya's 1986 SIGGRAPH conference paper. It sends a number of 3D lines (rays) into a 3D polygonal model. This program has similar computational demands compared to a traditional deterministic ray tracer but with less memory coherence.

253.perlbmk

This program is a cut-down version of Perl V5.005_03, but with most OS-specific features removed. For the workload perlbmk runs several scripts that do various tasks such as converting email to HTML and finding perfect number.

254.gap (Groups, Algorithms, and Programming)

This program is a group theory interpreter. It implements a language and library designed mostly for computing in groups.

255.vortex

This program is a single-user object-oriented database transaction benchmark which exercises a system kernel coded in integer C. This program is a derivative of a full OODBMS that has been customized to conform to SPEC CINT2000 guidelines.

256.bzip2

This program is based on Julian Seward's bzip2 version 0.1. The only difference is that this program performs all compression and decompression in memory to help isolate the CPU and memory subsystem.

300.twolf

This program is for determining the placement and global connections for groups of transistors needed for creating the lithography artwork needed for the production of microchips. This version of the program has been modified to capture the flavor of many implementations of simulating annealing. Most execution time is spent in the inner loop calculations and so the program traverses memory often creating cache misses.