An Investigation of Xen and PTLsim for Exploring Latency Constraints of Co-Processing Units

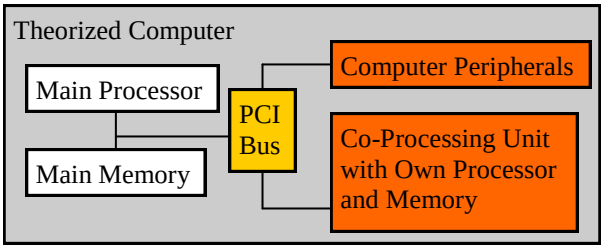
Grant Jenks - jenks@cs.ucla.edu
University of California, Los Angeles

Abstract

This paper explores the capacity of co-processing units to aid in general computations carried out by the CPU. While much work has already explored the effectiveness of specialized units, like graphics cards, this research attempts to model and consider the effect of adding general processing cards on the PCI bus. The hardware is simulated using the Xen 3.0.2 virtualization platform and PTLsim rev. 219 hardware simulation architecture. Much detail is described in the methodology section about how to set up the research platform and run the tests. To some extent, this part may act as a guide that can jump start future researchers using this platform. Five hardware models are simulated and analyzed. The models are approximations of the AMD K8 processor architecture and vary in the number of threads which are simultaneously run. The conclusion focuses both on the results of the hardware simulations and on the use of the PTLsim/Xen hardware simulation tool.

Introduction

Most computers do not fully utilize all the PCI card slots that are available on the motherboard. This under-utilization provides an opportunity for improvement. The goal of this project will be to investigate the potential contribution of a co-processor unit which would communicate on the PCI bus. This co-processor could handle some of the workload of the main CPU. Envisioned is a PCI card with appropriate features to offer the computer more parallel processing power and more memory. Some specific research has already been explored regarding the use of graphics cards as specialized processing units. The goal of this project is to understand what obstacles face a general processing unit.



The research uses virtualized machines as a platform for the hardware simulator. To facilitate the use of virtual machines, a processor with Intel Virtualization Technology is used. The virtual machine platform is from Xen, and the hardware simulation platform from PTLsim. By utilizing the specialized hardware, Xen can run unmodified guests in virtual machine spaces. This virtual machine normally runs natively on the machine hardware. PTLsim allows the machine hardware to be further virtualized so that cycle accurate measurements can be carried out. With the environment properly set up, the analysis is carried out using the PTLsim architecture for hardware simulation which has previously demonstrated accurate measurements.

The research is pursued in three steps. The first step involves demonstrating some need for off-loading execution on a co-processing unit. Once this is demonstrated, the execution profiles of a variety of benchmark programs will be gathered when run alone on the ideal simulated hardware. These profiles will be compared with subsequent profiles that are determined by running the benchmarks in an operating system environment with three different hardware models. The comparison will yield results that can determine the overall usefulness of implementing the co-processing units that are modeled.

Motivation

PCI card slots allow a plethora of peripherals to be added to machines without modification or upgrade of the motherboard. The number of PCI slots that are generally provided on a motherboard are often greater than the actual number of peripherals that are used. The remaining peripheral spots may be used for various redundant or specialized applications. No known unit however exists for general co-processing. Some designs currently devote these peripheral slots to graphics or networking but these applications are specific and the boards are tailored to their purpose. The real desire in this research is to have boards that can serve to offload the processing done by the main processor in any setting. A specific example of this involves offloading the cycles that run for programs in the system tray of Windows clients so that the start up

of programs may not be slowed by background tasks. By utilizing the peripheral slots, there is also less need to upgrade an entire computer which may need a newer motherboard with support for some improved processor. The proposed system my therefore extend the useful life of motherboards.

Prior Work

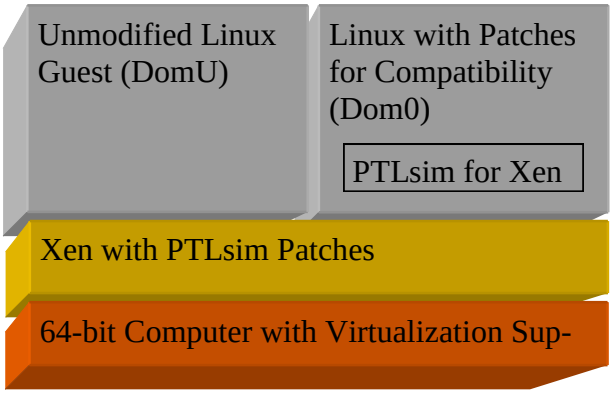
Much prior work has focused on using the peripherals attached to the motherboard for specialized processing techniques. A lot of papers have focused on ways to use the graphics cards in computers for computationally intense problems including many problems in linear algebra. A current company named Ageia has also produced a specialized card which is designed for physics processing of objects in games. This company provides an SDK with their hardware which may be used by software developers in any way. These technologies are closely related to the exploration considered in this research. Many of the papers have also been quite successful in demonstrating not only the usefulness of applying specialized hardware to various problems but also the incredible efficiency. The purpose of this paper is different from other papers in this way. It is not intended for any specialized hardware to be included in the co-processing unit. While graphics cards have an enhanced capacity to do floating point calculations and other boards may specifically be optimized for power, the approach here is fundamentally different. To a certain degree, the idea presented here involves putting a computer within a computer. The added co-processing unit would have a processor, memory, and access to disks. The drawback to this design is the same as that for any computer in which the ability to do calculations for a wide range of problems involves sacrificing the efficiency of solving specific, well understood problems. The reason for this design is so that any program could be run on the co-processing system.

Approach

As the research is broken into three parts, each part has its own approach. The first part involves demonstrating some need for additional processing resources within a computer. This is done using Intel's Vtune program to gather runtime profiles of the computer under different loads. The computer which is used for this purpose in the research has a processor with Intel Hyperthreading (HT) technology. Therefore, runtime profiles with Intel Hyperthreading on and off are gathered and the results are described.

The second part of the research determines specific runtime profiles of benchmark programs run within the PTLsim simulator. This is carried out using PTLsim-Classic. The classic simulator gathers cycle accurate data pertaining to only a single program. It cannot model a multiprocessing environment that would reflect programs run in an operating system. Since the program runs in isolation, this is an ideal way to determine standard program execution.

The final and by far most complicated part of the research involves setting up PTLsim with Xen so that the hardware may be simulated in a cycle-accurate environment. This simulation will allow runtime profiles to be developed which can be compared to the previous runtime profiles gathered in PTLsim-Classic. This approach requires four key things. The first requirement is relatively new hardware. PTLsim requires a 64-bit processor and Xen requires a processor with virtualization technology support in order to run guest virtual machines unmodified. On top of the hardware, a modified version of Xen must be run which has the appropriate patches for PTLsim to interface with it. In order to start virtual machines, a Xen compatible Linux kernel machine must run on top of the Xen platform as domain 0. Once this is set up, PTLsim must be built appropriately so that advantage can be taken of the underlying Xen technology. The methodology section will speak at length about the many different configurations that were tried before a working environment could be set up.



Methodology

The process of preparing the environments for each of the three stages of research has ranged from simple to very challenging. Each phase is described here with particular attention paid to the third stage was has the greatest associated difficulty and most useful capacity. Recall, the initial phase of the research sought to demonstrate a need for more processing power in today's computers.

The runtime profiles of the test machine used in this research were collected using Intel's Vtune

software. Setup was easy and consisted of simply installing the program with defaults set. Learning to use the program is aided with a detailed walk through guide that is provided with the software. The processor loads were constructed in various ways. In each setup the computer was rebooted before any measurements were made. After the system completely booted, the background programs were started and each was given the necessary time to fully load and reach their stable state. After this, the Vtune performance monitor was started. At this point, the foreground programs were started and put into action. This might involve playing music or a video for a few seconds before running the Vtune program for one minute. The Vtune program was set to monitor the entire system, not a single program. For each different setup, the Vtune performance recommendations were recorded and averaged over five trials.

Having acquired the runtime profiles of the system under different loads, the second part of the research involved collecting specific profiles of programs when run on the PTLsim-Classic hardware simulator. The steps involved in this are not much more challenging than before except that the sources for PTLsim were downloaded from the online site and compiled and installed. With the necessary tools already installed, this requires only typing "make" in the shell. Once PTLsim was installed in the default manner, the hardware simulations were carried out in a number of steps. For each benchmark program, a ".conf" configuration file was created in a directory with path "/root/.ptlsim/[path from root to binary]/[binary name].conf." This file specified general log file and statistics information. The statistics information was configured to take snapshots every ten million instructions and to execute no more than fifty million instructions. The core used in this simulation was the out-of-order core that is provided with the standard PTLsim archive file. Because of the highly deterministic nature of this simulator, only three runs were performed on each benchmark program. The recorded data included all summary statistics as well as the overall branch miss-prediction rate, data cache load miss rate, and instruction cache load miss rate.

Implementing the third stage of the research was considerably challenging. These challenges are each described here. The first challenge is acquiring hardware that will be compatible with the simulation environment. PTLsim specifies that it only needs a 64-bit computer to meet its hardware requirements. Xen is more demanding and requires a processor with virtualization technology in order to run unmodified guests under full virtualization. PTLsim does not work with fully virtualized guests but Xen will generally warn during an installation if the computer lacks virtualization support. Two machines were tried for the Xen environment. First a computer with an AMD 64-bit processor was tested without much success. Though Xen would install and a Dom0 guest could be run on top of it, the system was buggy and limited in that Xen could not load unmodified systems and so could not be used for PTLsim. The second machine used an Intel 5300 series 64-bit processor with Virtualization Technology. This machine worked very well with the newest version of Xen but warned when older versions of Xen were installed that the CPU microcode was not recognizable. This problem may be attributed simply to the hardware being ahead of the software. By using online updates to patch this problem, the issue was overcome.

It is the recommendation of PTLsim to use an operating system from one of the major distributors that has a plethora of features. The one that they presently use is OpenSuse 10.2. They use the kernel for this operating system in both Dom0 and DomU. In my attempts I have used OpenSuse 10.2 and 10.1, Fedora 6, Xen 3.0, CentOS 5, and Ubuntu 6. The operating system that finally worked for the environment was OpenSuse 10.2 and I would recommend it just as much as PTLsim does.

When I began, I installed OpenSuse 10.2 on the machine without virtualization technology. OpenSuse has a wonderful operating system installer called Yast that made it very easy to set up the operating system with Xen installed. After several installations it became clear to me that a simple partition table is much more easily managed in these situations than a normal one. Dividing each main folder onto its own partition is a standard Linux method for securing data but in this environment I've generally found it makes things needlessly complicated.

Another word about partition setups before continuing. Use ext2 or ext3 and remember to be sure that support for that file system is in whatever kernel is loaded at boot. This must be true of both the Domain 0 virtual machine that is run and all guest virtual machines. Configuring guests properly can also be tricky with file systems. Xen tries hard to prevent the user from doing something foolish but is not always successful. A number of times, I have accidentally tried to boot Domain 0 from its disk in a virtual machine as DomU. This is an almost guaranteed way to corrupt data on the disk. An all too frustrating thing occurs when after installing an operating system and all needed patches, the disk is corrupted in a few minutes and the process has to be repeated.

Installing the patched version of the Xen kernel and tools in OpenSuse required only following the instructions provided in the PTLsim manual. A very challenging error occurred however when trying to get

the PTLsim demo machine to run. The error was a kernel panic error in which the booting virtual machine could not mount the disks to load the virtual file system. This error is difficult since the abstraction of the disks should be handled by Xen and the needed files for Xen were certainly available on the machine. While struggling with this error for a while, I learned the location of two important folders. The first location is "/etc/xen". This folder contains Xen configuration files, most notably "xensys.config" and virtual machine configuration scripts. The second folder is "/var/xen/log". This folder contains Xen and Qemu log files which can aid considerably in determining problems. Before I could resolve the kernel panic error, I got better hardware and decided to implement the environment on that hardware instead.

The second machine used in the research was a server that had two processors with four cores each, two gigabit Ethernet ports, and a raid controller. In describing the process of getting the environment to work, we will see how each of these aspects complicated installation. I must also admit that I had never worked with a server before and so have learned a number of system administration tasks. Now, a few notes about the server before continuing. Make sure the BIOS and other firmware in the machine is up to date. A couple of errors that I have seen involved invalid microcode being read by the kernel. I don't exactly know how this affects an operating system but having everything up to date is especially important to Xen. Another mistake would be to implement a raid. It's not that it's not possible. Rather, it is best to become fluent in installing the environment before trying to speed everything up. It is noteworthy however to say that a raid may be particularly useful in some environments since Xen can become overly burdened and even crash in very I/O intensive tasks. This will effect the results in your environment since Xen/PTLsim will not abstract properties of the disk that may be enhanced.

After updating and tuning the server as necessary, I tried installing OpenSuse. Installation failed repeatedly for me. While investigating this I determined the problem was similar to another person's who had recognized the problem to be a segmentation fault that occurred when reading the disk meta-data that was written by the raid controller. Unfortunately, no one determined a solution for several weeks and so I meanwhile decided to try other operating systems. The first operating system that I tried was OpenSuse 10.1. I do not suggest using the older operating system for virtualization. Kernel improvements have been extremely important and so recent changes must be included in the distribution that is used. OpenSuse 10.1 had the same problems as 10.2 and never got past the installation phase.

The second distribution I tried was Fedora Core 6. This distribution is fundamentally different in that it derives from the Red Hat code base. Installation of Fedora is relatively simple and facilitated by its Anaconda installer. Installing Fedora with Xen automatically installs all the Xen libraries which include modified glib libraries. PTLsim has made further modifications to these libraries for its purposes which is why the patched Xen kernel and tools must be installed. The Fedora installation worked fine and was capable of creating virtual machine guests. It's always a good test to be able to create virtual machines on the unmodified platform before making changes and creating virtual machines under hardware simulation. By testing this process first, later issues can be nailed down more quickly.

In addition to trying Fedora, I installed CentOS which also derives significantly from the Fedora code base. Installation for CentOs was the same as that for Fedora. As far as I can tell from installation, the distributions are very similar. After following the instructions in the PTLsim manual with the exception of a few name changes, the environment was set up for simulation. It's important to note that the kernel that is built and patched by the operating system installer and update manager was the only compatible kernel that worked with Xen. When trying to run the sample domain that is provided, the same error as before was generated. This error was the kernel panic error which results from the VFS failing at boot up. This error also often occurred only seconds before the machine would freeze and need to be forcibly rebooted. Since hard reboots were often required, I recommend using a file system with journaling so as to avoid corrupting blocks in these systems.

Given the similarity between Fedora and CentOS, it is not surprising that they both failed. I decided after trying these systems to implement Ubuntu. Working with both the desktop and server version of the operating system resulted in the same outcome. Ubuntu will install and configure Xen during installation of the operating system. This makes it, like the others, very easy to install Xen. Making the changes to Xen however, proved to be too challenging for me in my narrowing time window. The Ubuntu installation with Xen includes the Xen modified glib libraries. These libraries are not compatible with the package manager which can install gcc. The GNU C compiler is absolutely necessary for both building the revised Xen kernel and the PTLsim executable. Furthermore, installing both sets of glib libraries was not acceptable to Ubuntu and led me to give up and try OpenSuse again, this time with success.

When I came back around to OpenSuse, con-

siderable work had been done on the bug that I had earlier discovered. The solution to the bug was to zero out all the disks on the drive which would effectively erase the meta-data that caused the segmentation fault. After several attempts at this, I discovered that disk meta-data is stored at the very end of a disk and so skipping to the end and then writing zeros is a very efficient way of preparing the disk for installation. With this change in effect, the installation of OpenSuse 10.2 succeeded exactly as it should.

Following the installation, the steps in the PTLsim manual were followed as closely as possible. The kernel that is provided with PTLsim was installed to the necessary directory but failed to boot. Constructing a ram disk for the kernel and providing that as a parameter to the kernel in Grub also did not help. Furthermore, the kernel that is installed with Yast is not compatible with the changes that are made to Xen. The problem observed was that the Yast configuration tool for virtual machines constantly reported no memory available in the computer and virtual machines that were started would silently fail in a process that degraded the stability of the machine. The kernel that finally did work was custom built from the patched sources that are provided on the PTLsim web site. The configuration file for building the kernel was based off the previous configuration file that is created during the initial installation. Additionally, conservative options were marked when configuring modules that were not already accounted for in the previous kernel configuration. The main goal of the final environment is stability and so including nothing experimental and everything recommended worked as it should. An initial ram disk for this kernel also had to be created using the "mkinitrd" command.

Booting the PTLsim patched Xen kernel with the modified Xen tools and custom built Linux kernel with ram disk worked successfully. The resulting machine had all the features of OpenSuse 10.2 including the ability to manage software repositories with Yast. The only failed module at boot up was the AppArmor module which is not included in the kernel that is provided on the PTLsim web site. This failure did not affect the system. The process of starting virtual machines requires a lot of command line work. I tried repeatedly to port virtual machines that I had developed in other environments to this test machine unsuccessfully. For some reason, both fully virtualized and paravirtualized machines would fail silently. It is important to note that at the time of this writing, the PTLsim patches do not support fully virtualized machines and so trying to do so will result in a machine that freezes during boot. This also implies that PTLsim cannot be used to simulate Windows environments which would require full virtualization. The only effective way that was found to create paravirtualized machines was to do it the basic way. This involved creating an image file in Domain 0 and then copying over the disk from a working bootable system. Oddly, the kernel that was used to boot Domain 0 was not able to boot the paravirtualized machine. Instead, the kernel that was provided on the PTLsim site along with the ram disk that was made for it were used.

With the capacity to boot guest domains at the command line, I now tried to get a graphics system to work. The two methods for graphics in Xen are VNC and tunneling X over ssh. In order to get VNC to work, a VNC server must be started in the guest domain which can be connected to from domain 0. Unfortunately, both these systems failed because networking would not work in the guest domain. This is strange since networking did work in domain 0. Since networking was working in domain 0, the network bridge that Xen sets up was working. I confirmed this repeatedly. Regardless, the network bridge would not connect any guest domains that I created. Without networking, the VNC system could not connect because the server and client in different domains could not identify each other and initialize the connection. This is unfortunate since VNC should not require networking to work. Possibly, a more expert person in VNC would have succeeded in this problem. It should also be clear that without networking, tunneling X over ssh also would not work. Even if this did work, there are issues involved with this communication since time is treated differently in the different domains. This makes a lot of the timeouts, inherent to the TCP protocol, work improperly.

The two methods of implementing graphics did not work in the PTLsim modified Xen domains. Furthermore, these methods do not utilize a graphics

Operating System	Main Success	Main Failure	Comment
OpenSuse 10.2	Boots to working environment.	Raid does not work.	This is the best choice.
OpenSuse 10.1	Xen started properly.	Could not start virtual machines.	Use the newer version.
Fedora Core 6	Could start virtual machines.	Froze when changes were applied.	Good back up.
Ubuntu 6	Xen started properly.	Could not get necessary tools.	Good for learning Xen.
CentOS 5	Could start virtual machines.	Silently failed after changes.	Resilient to failure.

card because they do not work under Xen's full virtualization. This means that the simulator will not properly account for a graphics card as graphics systems are implemented by communicating the graphics information to domain 0 generally via the network card. Because the network card is used for communication and not the bus, the simulated machine does not properly utilize a graphics card. This is an inherent limitation in using PTLsim.

When porting my unmodified guest domains to Xen, I had originally specified that there should be two virtual CPUs. This caused an internal error in PTLsim. A nearly identical error is documented on the PTLsim website with a solution which suggests adding "-reservemem 256" to the command line. There were two issues with this solution. First, this shouldn't fix the problem since it resolves the problem of too much machine memory which shouldn't happen in a machine with four gigabytes of memory. Second, the parameter was unrecognized on the command line and resulted in PTLsim erring and ignoring the parameter. Because the command line parameter was not recognized and the error is not clearly a result of memory issues, this problem could not be resolved. Without being able to specify a number greater than one for the virtual CPUs parameter, cooperative multi-processing systems could not be modeled.

Without being able to simulate cooperative multi-processing systems, other ways to simulate parallel processing were explored. The clearest way to do this was to modify the number of threads which could be processed in parallel in a symmetric multi-threading processor. Five levels of symmetric processing were chosen. These levels modeled processors with ability to process one, two, four, six and eight threads in parallel. This is rough approximation of a CMP system for two reasons. The first issue is that an SMT system is symmetric and therefore no heterogeneity in the processing units exists. The second issue is that the same data and instruction caches are shared amongst all the processing unit. In order to better approximate the originally sought after system, attempts were made to scale the caches with the number of threads processed in parallel. A couple of methods were tried in which the cache would provide multiplicatively more resources based on the processor with which it was paired. No structure was found however which would scale the resources in the same way across all the simulated platforms. Generally, PTLsim would fail at runtime saying that an error had occurred in the simulated platform. This error made the platform unreliable and impractical for research.

The final simulation platform allowed SMT processors which varied in the number of threads processed in parallel to be simulated. These systems all had the same amount of memory and cache subsystems. The systems varied only in the number of threads processed in parallel. This change can easily be made by enabling support for symmetric multi-threading and changing the count of how many threads are processed in parallel in the "smtcore.h" file. By modifying these constants and rebuilding PTLsim after each change, the five different machine models could be simulated.

On the five different machine models, three different tests were run. Without graphics support which used the graphics card, the original goal of evaluating how programs impact each other in different machine models can not be realized as robustly as originally intended with standard, every day, programs such as mail clients, web browsers, and games. The three setups used were, the Tar compression program, a Perl script which generated and printed random numbers, and a combination of these two programs. The Tar compression program was run precisely using "tar zc /usr /lib | tar ztv > /tmp/allfiles.txt" at the command line. The Perl script was run using "perl makerand.pl" at the command line. These commands were executed together with a shell script which would run both commands in the shell as background processes.

The results finally gathered were averaged over five trials which were executed in the following way. First, the guest domain was started with PTLsim. Then, the programs were started in the guest domain and PTLsim was sent the commands: use the modified SMT processor for fifty million instructions, log the use, and switch back to the native execution platform. This method has an issue in that the time between when the program starts and when the simulator starts simulated execution is not instantaneous and is instead based on how long it took me to switch windows and run the necessary commands. I tried to minimize this latency as much as possible with scripts and careful structuring but still could not achieve a negligible latency. The PTLsim system has a program which should overcome this problem by communicating between the domains. Unfortunately, this program did not work in the unmodified guest that I used for this research. This program could not be built in the guest domain because of missing C library dependencies. This system also, unfortunately suffers from not being able to "warm up" the simulation platform before logging and therefore will inevitably include a few thousand cycles of inaccurate results. This bad data should be overcome by running the simulation for fifty million instructions.

Having created the three necessary environments for the research the tests were carried out successfully and have produced the results described

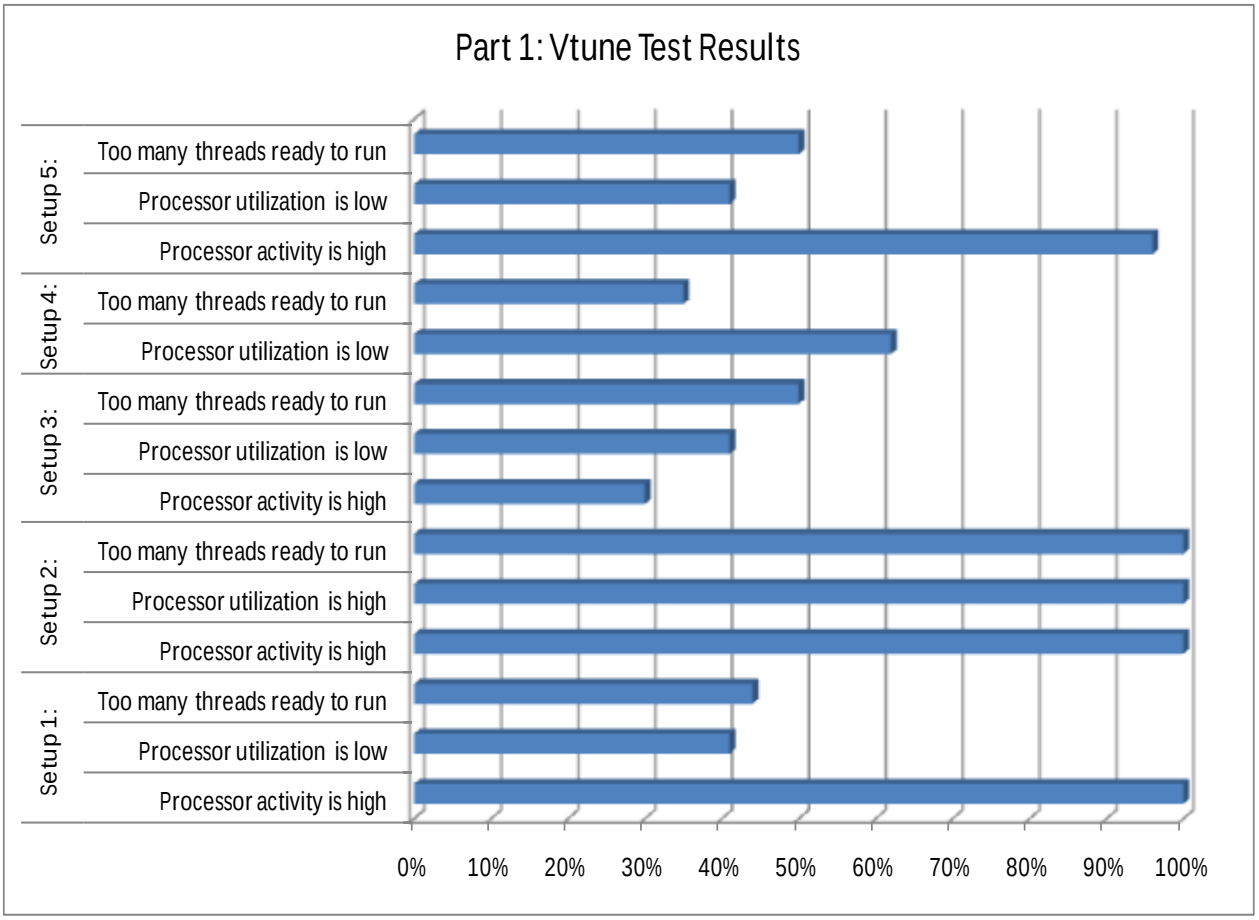
next.

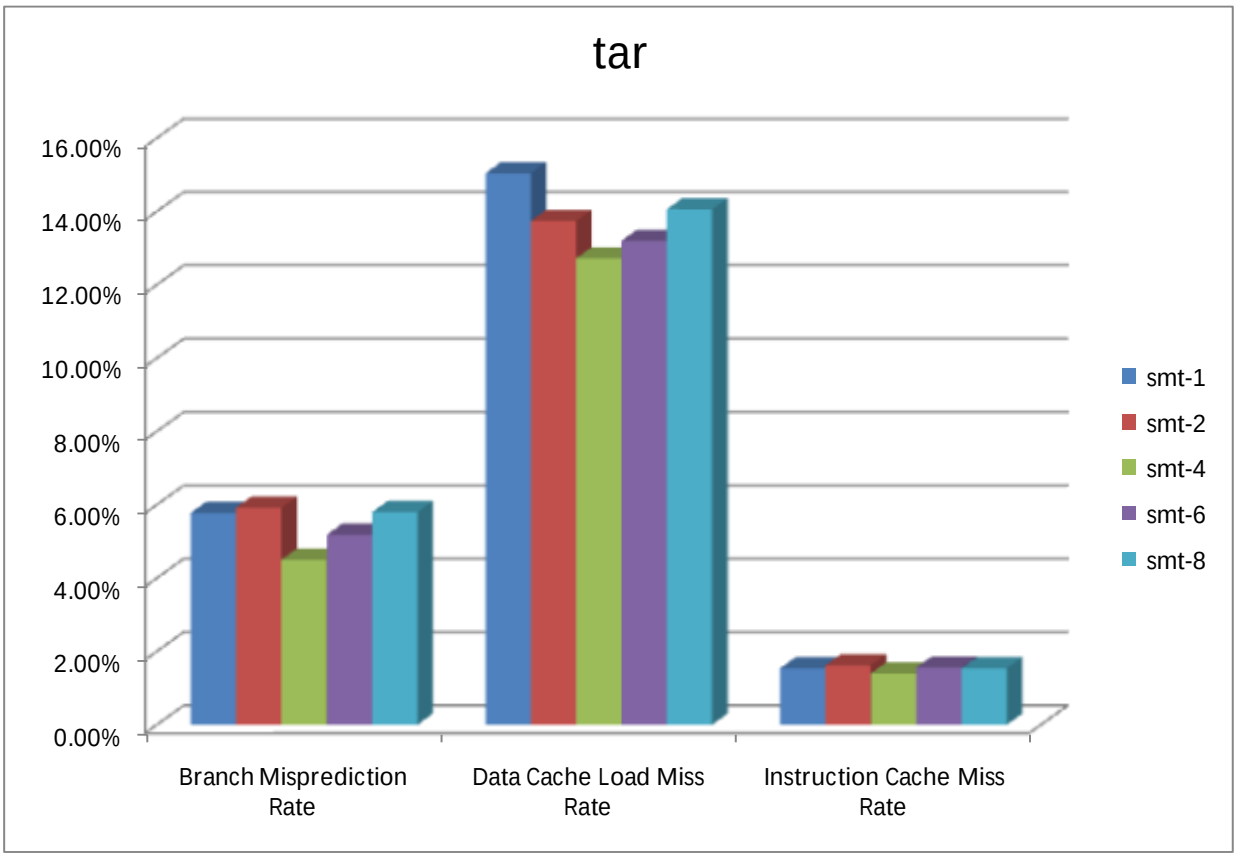
Results

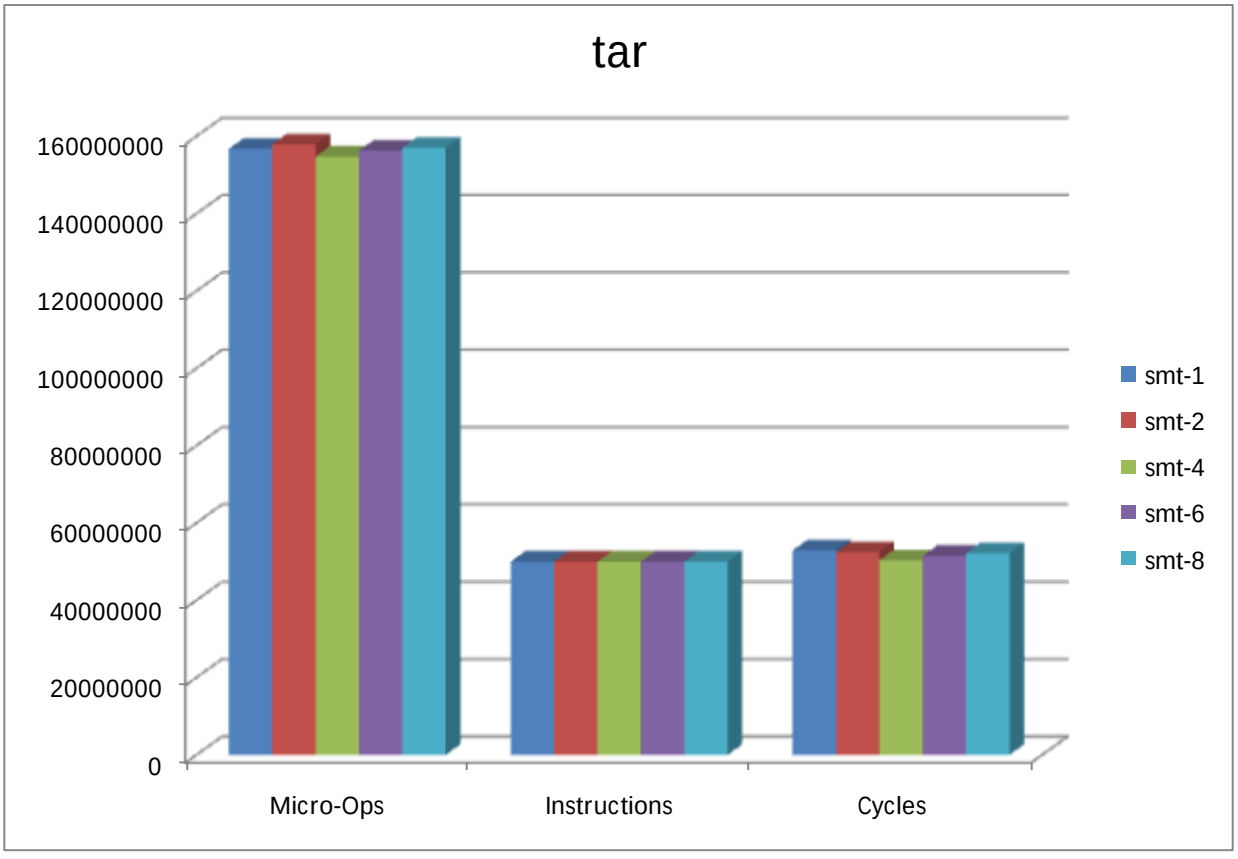
The data for the Vtune test results is based on averages over five trials and displays the recommendations of the Vtune analyzer. The percentages reflect the extent to which the system is affected by the problem. The descriptions of the setups are explained in the table on the right. In the table, "numerous background programs" refers specifically to the following: Microsoft Outlook (mail client), Bit Comet (torrent downloader), Trillian (instant messager), ATI Display Controls (display controller), Microsoft Indexer (file indexer), Natural Colors (monitor controller), Sata Link (disk controller), C-Media Mixer (sound controller), Belkin Bulldog Plus (UPS controller), Avast VFBD (antivirus database), and Avast Scanner (antivirus scanner). The other programs used were: Windows Media Player for playing movies, Windows XP for networked file transfer, WinRAR for file compression, Oblivion for game playing, and iTunes for music playing. These programs were run as described previously in the simulation environment.

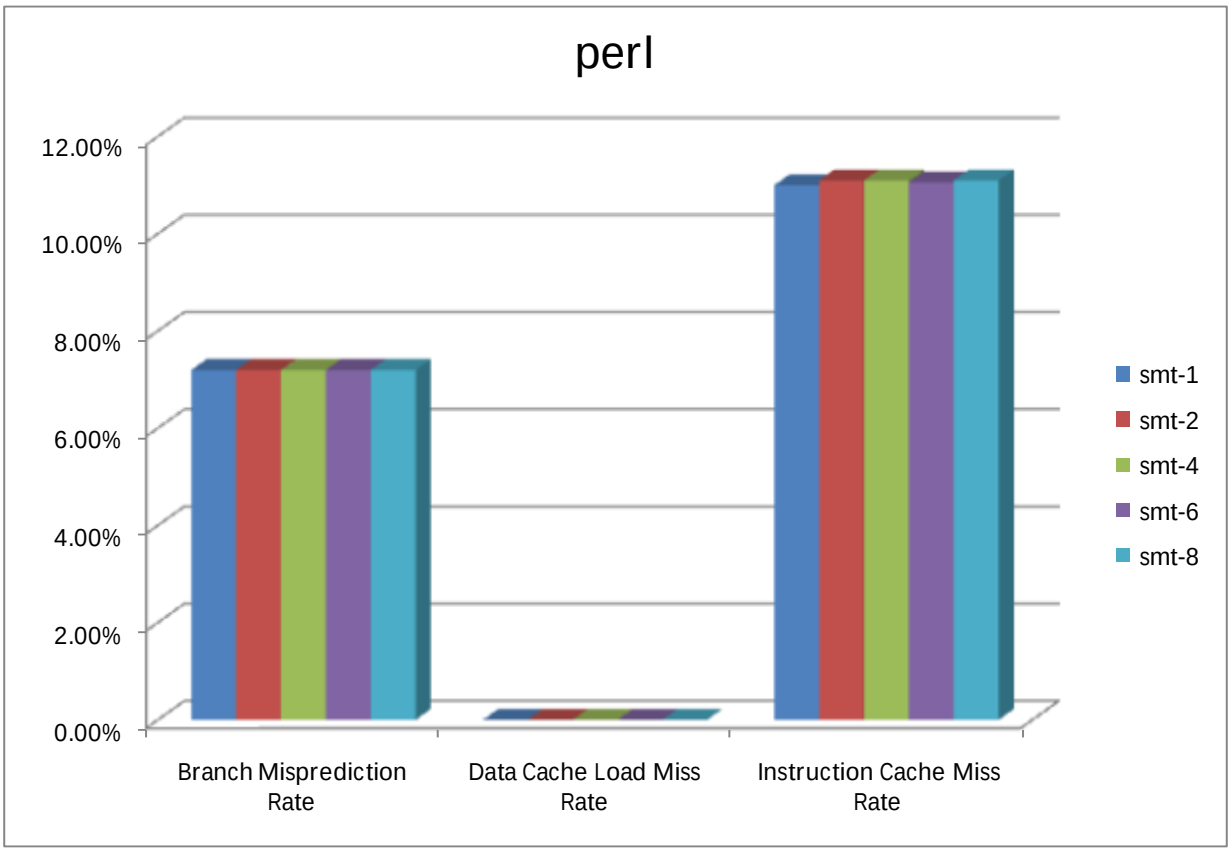
The data from PTLsim is displayed on the next three pages. This data shows the results of running the benchmarks across all five simulated machines. The reported values are each averages over five trials. There was little variation in the collected values for the Tar and Perl benchmarks but much

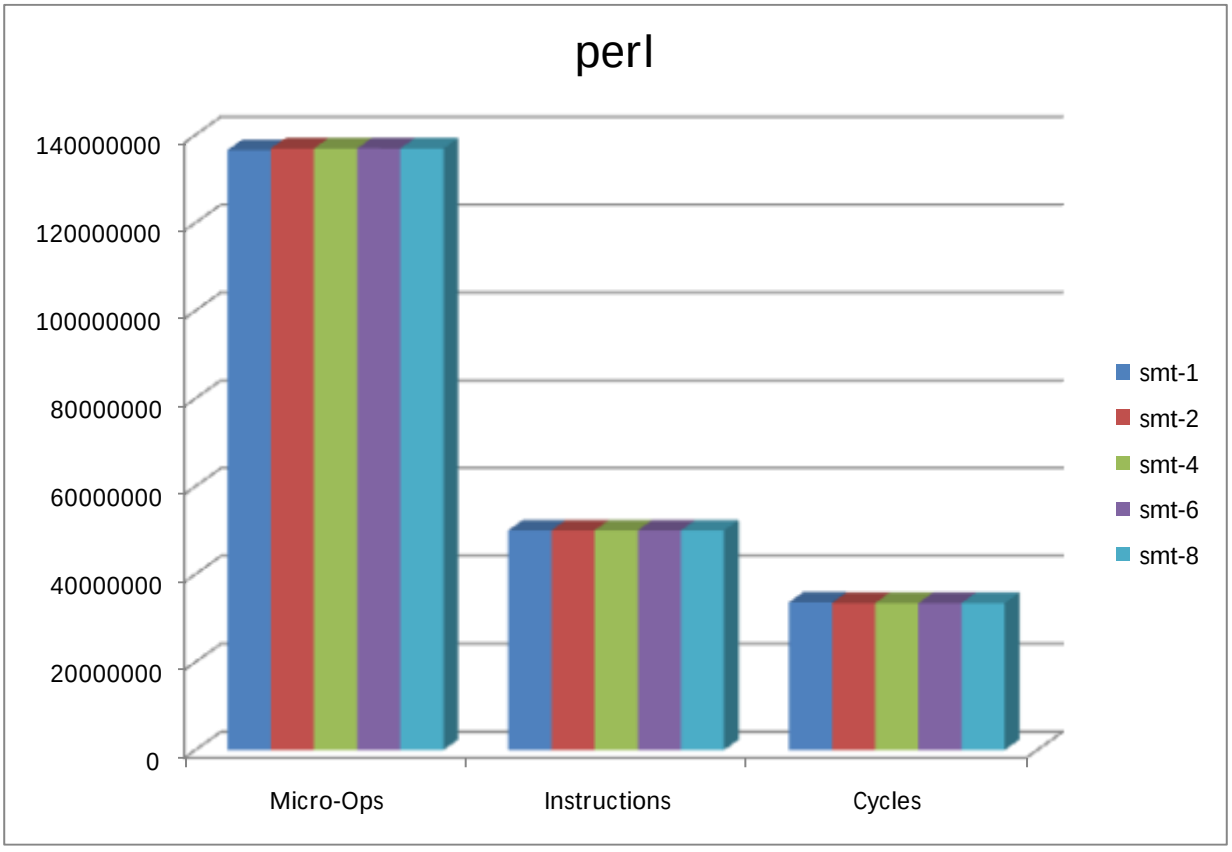
Setup	Description
1	HT on; demanding game; numerous background applications
2	HT off; demanding game; numerous background applications
3	HT on; movie playing; network file transfer; anti-virus scan; numerous background programs
4	HT on; movie playing; file compression; numerous background applications
5	HT on; demanding game; music playing; anti-virus scan; numerous background programs

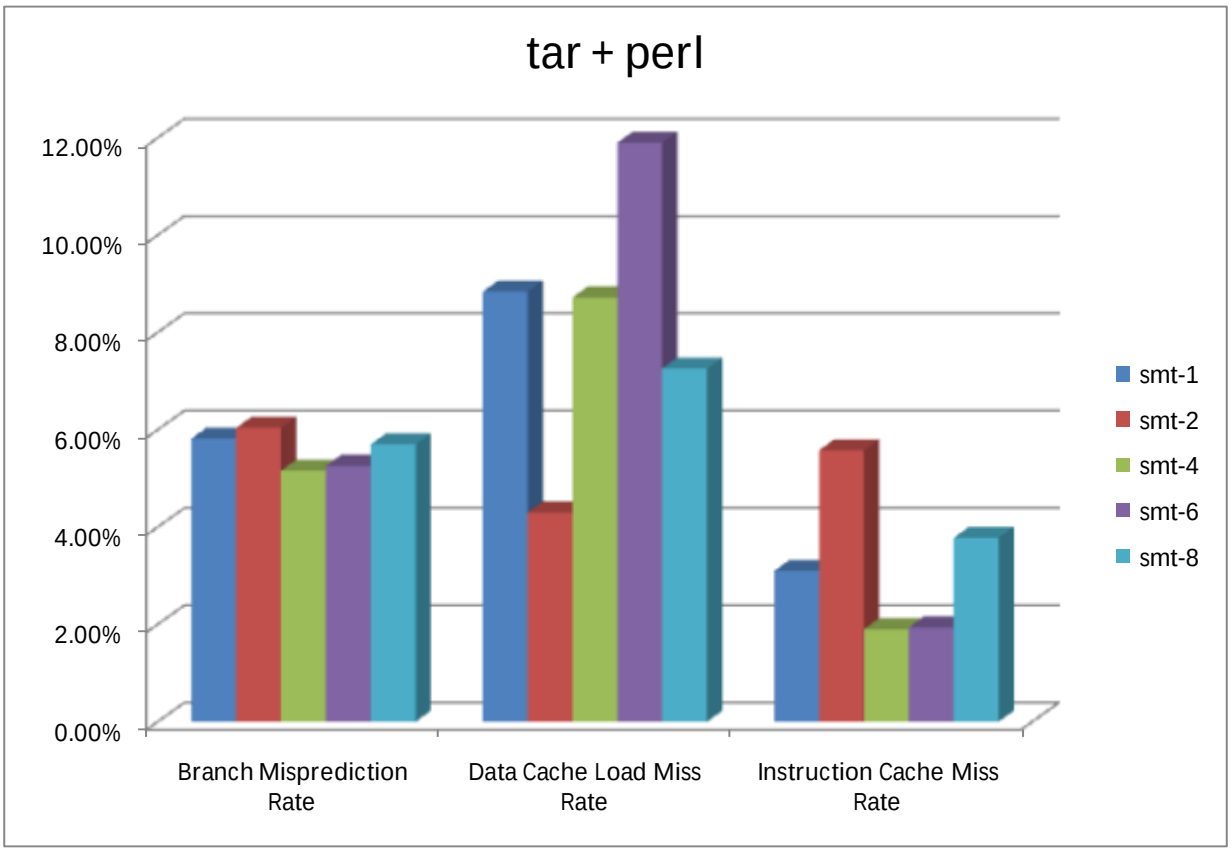


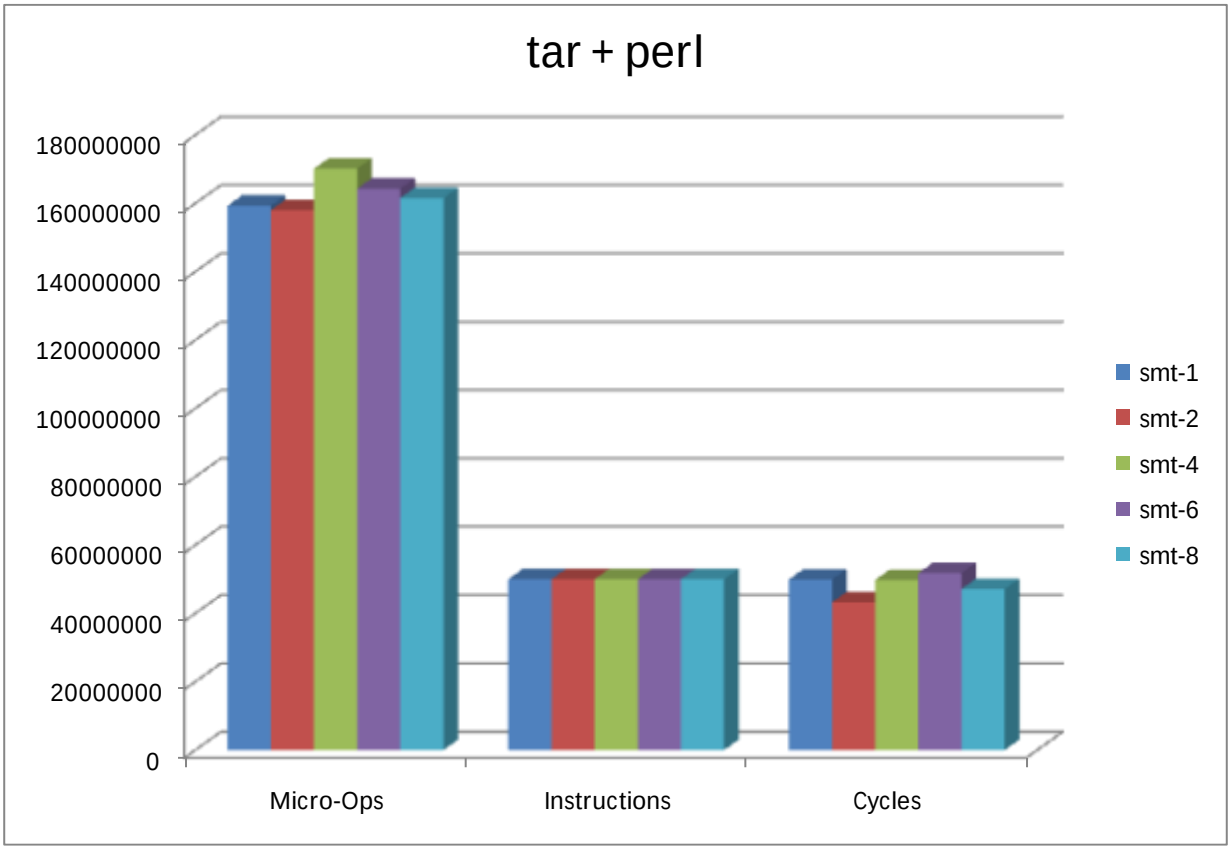

tar
16.00%
14.00%
12.00%
10.00%
8.00%
6.00%
4.00%
2.00%
0.00%
Branch Misprediction Rate
Data Cache Load Miss Rate
Instruction Cache Miss Rate
smt-1
smt-2
smt-4
smt-6
smt-8


tar
160000000
140000000
120000000
100000000
80000000
60000000
40000000
20000000
0
Micro-Ops
Instructions
Cycles
smt-1
smt-2
smt-4
smt-6
smt-8


perl
12.00%
10.00%
8.00%
6.00%
4.00%
2.00%
0.00%
Branch Misprediction Rate
Data Cache Load Miss Rate
Instruction Cache Miss Rate
smt-1
smt-2
smt-4
smt-6
smt-8


perl
140000000
120000000
100000000
80000000
60000000
40000000
20000000
0
Micro-Ops
Instructions
Cycles
smt-1
smt-2
smt-4
smt-6
smt-8


tar + perl
12.00%
10.00%
8.00%
6.00%
4.00%
2.00%
0.00%
Branch Misprediction Rate
Data Cache Load Miss Rate
Instruction Cache Miss Rate
smt-1
smt-2
smt-4
smt-6
smt-8


tar + perl
180000000
160000000
140000000
120000000
100000000
80000000
60000000
40000000
20000000
0
Micro-Ops
Instructions
Cycles
smt-1
smt-2
smt-4
smt-6
smt-8

variation in the final benchmark which ran both the Tar and Perl benchmarks simultaneously. For each trial, results were discarded if any errors occurred during execution. These errors were either program errors like invalid parameters, or system errors like segmentation faults, or PTLsim errors like failed assertions. These errors were rare and generally correctible with another run of the simulation environment. Thanks to PTLsim's incredibly fast design, each trial took only a few minutes.

Analysis

The data gained from the first part of the research shows that the processor is over burdened by the work that is processed. Furthermore, the results show that the problem is not necessarily with speed. Rather, the issue is that there are more processes ready to run than there are processors on which to run them. Even when Hyperthreading is turned on in the processor, the data shows that the machine would benefit from more processors. This provides evidence in favor of the idea that computers would benefit from co-processing units.

When we look closely at the first set of data we notice several important things. The first important thing is that when HyperThreading is off, the processor is definitely over worked by the load. This is evidenced by the absolute certainty with which Vtune asserts that processor activity is high, processor utilization is high, and too many threads are ready to run. The other setups in which HyperThreading is on, show much more varied results. In these cases the processor activity is sometimes high, about fifty percent of the time, and the processor either experiences low or high utilization. In all these cases, however, there are too many threads ready to run about fifty percent of the time. This indicates that a system with more parallel processing units would be better equipped to process all the threads running in parallel. The most important point to see in this is that with HyperThreading on, it is pertinent but not always necessary to further increase the amount of parallel thread processing.

The previous analysis provides some evidence that favors a cooperative multi-processing environment. Recently, this has been done with symmetric multi-threading and symmetric multi-processing systems. The results show however, that a symmetric system may be more than is necessary. This is evidenced by the fact that the recommendation is not one hundred percent. It is therefore not always a problem and offloading a thread to a second processing unit may only require a lightweight processor which is only sometimes available instead of a duplicate processor.

The second set of graphs show the results of running the three different benchmarks on different simulate hardware platforms. The various hardware platforms are systems which scale the number of threads processed in parallel. These platforms differ from the optimal design in two ways. First, they are not multi-processing systems. Instead, they are symmetric multi-threading systems. This means that there is no heterogeneity in the processing environment as there would be with co-processing units on the PCI bus. Second, the platforms have a single unmodified data cache which does not scale with the machine. This differs in that the proposed system would have a separate set of caches which would scale the amount of cached data and provide some isolation that would prevent threads from disrupting cache accesses of other caches. Though these two deficiencies are present, the cycle accurate simulation environment can still provide a rich set of results.

The first results are gathered from running "tar" in several instances. The benchmark used here actually starts four "tar" processes and executes them simultaneously by streaming data between the processes. The benchmark is very processor intensive and utilizes the disk. The disk attributes are not abstracted by PTLsim and so not considered here. The results show optimal utilization is achieved with an SMT processor that can process four threads in parallel. This is evidenced by the decreased miss rates, cycles, and micro-ops. Interestingly, we also see that as the parallel thread processing continues to scale up, the statistics get worse. It is therefore not always the case that a more parallelized system will be faster. Not surprisingly, the optimal point occurs when the amount of parallel processing that can be done matches the amount of parallel processing that is done. This also implies that adding an over abundance of symmetric processing resources may actually hinder execution and so a system which matches supply and demand closely should be sought after most. The intuition for this is that the least amount of system code executes when the threads are each matched with a processor. It is important to note, as well, that these programs were not run in an isolated environment and so various system tasks were also performed in the background. The minimal impact of these tasks was dwarfed by the amount of computation done in the benchmarks.

The next results occur from executing a Perl script which generated and printed random numbers. This script is very CPU intensive but runs entirely in a single thread. The results show that varying the capacity for parallel processing really doesn't improve or detract from program execution. There is a very slight

improvement in the instruction cache miss rate for a single threaded processor which may be attributed to less kernel code which tries to juggle threads running in separate contexts in the processor. This improvement is less than .1% however and therefore rather negligible. The other statistics show virtually no deviations and indicate no change. These results imply that unless there is work which can be parallelized, virtually no improvement will be observed. This is extremely important as it puts a significant burden on software developers to write code which takes advantage of improvements in the hardware which otherwise will not be realized.

The final set of results display statistics which occur when running both benchmarks. These results vary widely. Part of the reason for this is that, as explained previously, the simulation does not start at exactly the same place every time. Another reason for this is that the simulation environment is non-deterministic with this platform. The varied results show no definite trend across the five hardware models. The only general trend which can be deduced is that, to some extent, the data appears to be an average of the previous two benchmarks. This is what we would expect considering the threads are run in different contexts and so would combine with each other in a linearly weighted way. In particular, the core that can process eight threads in parallel shows this rather well. We cannot deduce more, however, since no application specific data is recorded by PTLsim. If we knew when and where each thread was running then we could observe how the threads affect each other. This would be particularly helpful in knowing how one thread might thrash the instruction cache and in turn affect the other threads. Not knowing application specific data is an inherent limitation of the PTLsim hardware architecture. Unfortunately, the variation in the data and the inherent limitation of PTLsim prevents any definite conclusions being drawn from this data.

The results in the first and second sets of data illustrate two interesting trends. The first trend is that, with Hyperthreading on, the system does not always experience a need for more symmetric parallel processing. This means that a device, like the one proposed which would not be a symmetric processing unit, might work extremely well in this context. These results follow directly from the Vtune data. The second trend is that while more parallel processing units improve execution, at a certain point, execution is actually hindered by two much parallel processing. Another result is that if there are no instructions to execute in parallel, then increased parallelism will have virtually no effect on execution. These results are specifically observed in the PTLsim/Xen data. The PTLsim classic data has not been included here for two reasons. The first reason is that it closely matches the collected data for the symmetric multi-threading processor with one processing unit. And the second reason is that it is not wholly relevant in the PTLsim/Xen scenario wherein the main results of the research are gained. Each of the graphs has provided interesting results.

Conclusion

The results of this research paper have explored to what extent PTLsim/Xen is able to simulate cooperative multi-processing environments in which parallel processing resources are connected via the PCI bus. PTLsim/Xen has shown itself to be extremely challenging to implement but also very robust in its capacity to model a wide variety of systems. Specific deficiencies in the implementation presented in this paper regard a lack of networking and graphics support. It is entirely likely however, that other researchers will be able to get these to work. This will allow for a wider variety of programs to serve as benchmarks. Considering the strong suit of PTLsim, which is processor simulation, the environment has been very powerful. The final system developed here could not utilize CMP systems but could easily scale the number of threads processed in parallel. Even though this falls short of the original simulation design, it is still an approximation. Other researchers may be able to overcome this approximation and obtain even more precise data.

The results from the collected data demonstrate several important aspects but are not conclusive enough to answer the original question. This is not a problem because the focus has been more on the simulation environment. I believe the results serve as both an example and a baseline. They illustrate what users should expect to be able to create and establish some basic properties of parallel processing systems. I have little doubt that further research can precisely answer the original question regarding co-processing units.

The virtualization technologies used in this research are very promising in their capacity to model hardware and gauge the effectiveness of changes. The environment in which this is done however, is overly complicated. Getting the three different kernels to work together with PTLsim is a challenging task. The recent advances in kernels is extremely important and differences between changes made even within a few weeks can make big differences in virtualization software now. Future techniques will probably use the newer kernel based virtual machine in which the Linux kernel developers are building in support. This should make things significantly easier for the user as

less issues are bound to occur between fewer kernels. This technology, known as KVM, is actually what PTLsim will use by the end of the year. Hopefully, this will make hardware simulation with virtualization easier and more popular.

Acknowledgements

A special thanks to my advisor, Dr. Glenn Reinman who really led me all along the way. Also thanks to Dr. Matt T. Yourst; our emails made the research platform possible. And finally, thanks to Dr. Amit Sahai who gave all of us in the research seminar this opportunity.

References

1. Yourst, Matt. "x86-64 Cycle Accurate Processor Simulation Design Infrastructure" 2006, 8 Oct. 1997 <http://www.ptlsim.org>.
2. Yourst, Matt T. "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator" IEEE International Symposium on Performance Analysis of Systems and Software, San Jose, California, April 2007
3. Ageia Technologies Incorporated, "PhysX by ageia" © 2007 <http://www.ageia.com/>
4. XenSource Incorporated, "Delivering the Power of Xen" © 2005-7 <http://www.xensource.com/>
5. University of Cambridge, "The Xen Virtual Machine Monitor" © University of Cambridge Computer Laboratory 2007 <http://www.cl.cam.ac.uk/research/srg/netos/xen/>
6. John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Kruger, Aaron E. Lefohon, Timothy J. Purcell. "A Survey of General-Purpose Computation on Graphics Hardware" In *Eurographics 2005, State of the Art Reports*, August 2005, pp.21-51
7. "Xen." Wikipedia, The Free Encyclopedia. 27 May 2007, 14:32 UTC. Wikimedia Foundation, Inc. 8 Jun 2007 <http://en.wikipedia.org/w/index.php?title=Xen&oldid=133857702>
8. "Qemu." Wikipedia, The Free Encyclopedia. 6 Feb 2005, 19:44 UTC. Wikimedia Foundation, Inc. 8 Jun 2007 <http://en.wikipedia.org/w/index.php?title=Qemu&oldid=16404237>.
9. "Virtual machine." Wikipedia, The Free Encyclopedia. 8 Jun 2007, 02:54 UTC. Wikimedia Foundation, Inc. 8 Jun 2007 <http://en.wikipedia.org/w/index.php?title=Virtual_machine&oldid=136749535>.
10. "An Introduction to Custom Xen Networking", Username: Steve from Debian Administration, Posted on Mon 27 Feb 2006 at 05:16 from http://www.debian-administration.org/articles/360
11. Rosen Rami, "Introduction to the Xen Virtual Machine" from Linux Journal on Thu, 2005-09-01 01:00 from http://www.linuxjournal.com/article/8540
12. Xen Wiki, "Xen Networking" from Xen Community Members on 2006-07-24 14:03:32 from http://wiki.xensource.com/xenwiki/XenNetworking
13. Xen, "User's Manual" Xen v3.0, ©2002-2005, University of Cambridge, UK, XenSource Inc., IBM Corp., Hewlett-Packard Co., Intel Corp., AMD Inc., and others. All rights reserved.
14. David Wentzlaff, "How to Tunnel X over SSH" <http://www.cag.lcs.mit.edu/~wentzlaf/faq/ssh_X.html>
15. Novell Users, "Bug 213145 - dmraid Causes Failure During Installation" on 2007-06-05 05:17:33 MST at <https://bugzilla.novell.com/show_bug.cgi?id=213145>