Live cd cluster performance

Haronil Estevez*
Department of Computer Science
Columbia University, New York
Advisor: Professor Stephen A. Edwards

May 10, 2004

Contents

*he99@columbia.edu

1 Abstract

In this paper, I present a performance comparison of two linux live cd distributions, Knoppix (v.3.3) and Quantian (v 0.4.96). The library used for performance evaluation is the Parallel Image Processing Toolkit (PIPT), a software library that contains several parallel image processing routines. A set of images was chosen and a batch job of PIPT routines were run and timed using both live cd distributions. The point of comparison between the two live cds was the total time the batch job required for completion.

2 Background

2.1 Knoppix

Knoppix is a debian linux based live cd created by Klaus Knopper. It is a bootable CD with a collection of GNU/Linux software, automatic hardware detection, and support for many peripherals. Running Knoppix on a PC requires only 16 MB of RAM for text mode or 96 MB of RAM for graphics mode. The Knoppix CD holds up to 2 GB of executable software and requires no installation. [**1**]. It is an attempt to create a fully featured rescue/demo system on a single CD and to unburden the user of hardware identification and configuration of drivers, devices and X11 for his or her specific hardware. The Knoppix application file system is an iso9660 (via cloop) file system that is transparently decompressed and read-optimized. The file system uses zlib/gzip compression.[**2**]

2.2 Quantian

Quantian is a customization of Knoppix intended for use in easily creating computer clusters. It is closely related to clusterKnoppix (itself a Knoppix customization) and focuses on

scientific computing. Both Quantian and clusterKnoppix differ from Knoppix most significantly in their choice of linux kernel. They both use the openMosix linux kernel extension, discussed below.

2.3 OpenMosix

openMosix is a tool for a Unix-like kernel, such as Linux, consisting of adaptive resource sharing algorithms. These algorithms are used to respond on-line to variations in the resource usage among nodes of a cluster through the use of process migration. Process migration occurs preemptively and transparently, for load-balancing and to prevent thrashing due to memory swapping. Users can run parallel applications by initiating multiple processes in node, and then allow the system to assign the processes to the best available nodes at that time. The dynamic load-balancing algorithm continuously attempts to reduce the load differences between pairs of nodes, by migrating processes from higher loaded to less loaded nodes. openMosix also contains a memory ushering algorithm geared to place the maximal number of processes in the cluster-wide RAM, to avoid as much as possible thrashing or the swapping out of processes.[**3**]

2.4 PIPT

The Parallel Image Processing Toolkit is designed principally as an extensible framework containing generalized parallel computational kernels to support image processing. The toolkit uses a message-passing model of parallelism designed around the Message Passing Interface (MPI) standard. A master node distributes equally sized image slices for processing to worker nodes using the MPI libraries and collects and recombines the processed image once all nodes have finished processing their image slice. [**4**]

2.5 MPI

The Message Passing Interface (MPI) is intended for use in the implementation of parallel programs on distributed memory architectures. A set of routines that support point-to-point communication between pairs of processes forms the core of MPI. Routines for sending and receiving blocking and nonblocking messages are provided. Implementations of MPI include mpich (http://www-unix.mcs.anl.gov/mpi/mpich/) and lam (http://www.lam-mpi.org). This paper uses mpich for the compilation and use of PIPT. [**5**]

3 Testing Setup

3.1 PC's

5 Pentium machines were used for testing:
(1) Pentium II 230 MHz, 96 MB RAM
(2) Pentium II 350 MHz, 128 MB RAM
(3) Pentium II 230 MHz, 64 MB RAM
(4) Pentium II 350 MHz, 128 MB RAM
(5) Pentium II 230 MHz, 128 MB RAM

3.2 LAN

A LAN was created consisting of the 5 PC's, networked together using an Etherfast Lynksis Ethernet Hub. Each node was assigned local IP addresses accessible only within the LAN. An additional ethernet card was installed in node (1) for connectivity to the internet. All 5 machines ran SSH servers, which mpich was compiled to use for launching programs in non-local nodes.

3.3 Test Data

I originally planned on using images ranging from 500 KB to 200 MB. However, PIPT's distribution of image slices by the master node of an image of size greater than about 10 MB resulted in sluggish or dropped ssh connections and PIPT out of memory errors. Based on these results, 15 TIFF files ranging from 500 KB to 8.4 MB were used for testing of the PIPT routines. Because PIPT uses the libtiff library (a library for reading, writing, and manipulating TIFF images), it shared some of its limitations. For example, format 28 TIFF images could not be read by PIPT and had to be replaced. To test if individual TIFF images could be successfully read by the libtiff library(and thus by PIPT), the X program **xv** was used, which also uses the libtiff library.

3.4 Selection of PIPT Test Routines

Since only 15 images would be used for testing, I selected the 15 most computationally intensive PIPT routines (out of 63 available). PIPT includes a test suite suitable for just such an endeavor. It was important to select routines that required a significant amount of time because of two reasons: 1) Routines that took a short amount of time to complete would most likely not be affected by parallelization and might even take longer when parallelized

due to the initial ssh connections necessary. 2) Routines that do not last very long would not have time to migrate when used in a Quantian environment. I ran all 63 PIPT routines in a batch job, 5 times each, and stored the results in a flat text file (run on the CS department machines). A perl script was used for parsing the text file and the averaging the run time

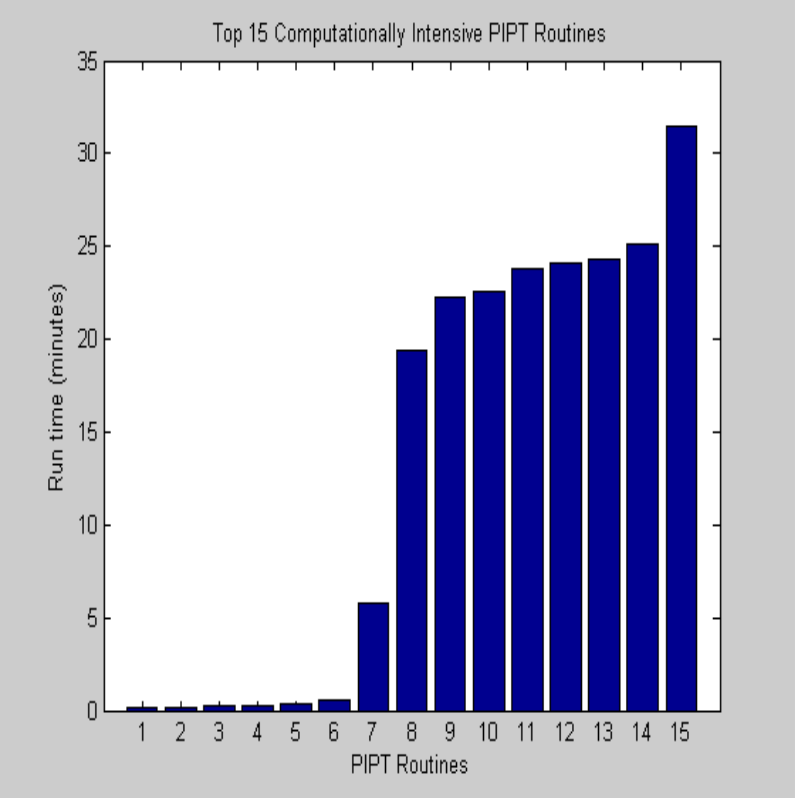

Figure 1: PIPT Routines: [1]AddUniform [2]CrossMedian [3]AddGaussian [4]ATrimmed-CrMean [5]AddBitError [6]SquareMedian [7]ATrimmedSqMean [8]CoOccurMax [9]CoOccurEnergy [10]CoOcuurContrast [11]CoOccurInverseMoment [12]CoOccurHomogeneity [13]CoOccurCorrelation [14]CoOccurEntropy [15]CoOccurCluster

for each PIPT routine. **Figure 1** contains the resulting average run times of the top 15 routines. Of these, only the top 8 required more than a few seconds to complete, most likely because they are calculating a feature matrix of the image instead of only applying a filter or enhancement to the image. Thus only the top 8 routines running time will be used as a point of comparison between the two live cd distributions.

3.5 PIPT process migration

For the compilation of PIPT for use with the Quantian live cd, several steps had to be taken in order for PIPT processes to achieve migratability. First PIPT was compiled normally without any modifications to its Makefile or special compilation flags. Then a simple two node cluster running Quantian was used to test the migratability of a PIPT process. This test resulted in negative results, the PIPT process did not migrate. The openMosix userland tool **openMosixprocs** was used to monitor the PIPT process and determine why it was not migratable. The tool revealed that the process was not migratable because it uses the **pthreads** library's **clone_vm** function. The **clone_vm** function causes threads to have shared memory. In an openMosix environment such as that provided by the Quantian live cd, processes that use shared threads are not allowed to migrate. Migration of processes that use shared memory was accomplished by The MAASK Team[6], which implemented a Distributed Shared Memory (DSM) patch for Openmosix. However, pthread migration has not been achieved because of the difficulty of handling pthread synchronization issues on a cluster platform.

A solution was found that only required slight changes to the PIPT configure script and passing one additional compilation flag. The configure script packaged with PIPT for creating a Makefile was edited so that the variables **HAVE_PTHREADS**,**HAVE_SOLTHREADS**, and **FOUND_PTHREADS** are set to 0. In addition, script code for verifying existence of the pthreads library was removed. Finally, PIPT was compiled with the additional compilation flag –**without-pthreads**. The process migration test from the previous paragraph was run and resulted in successful PIPT process migration.

4 Knoppix Test Results

The selected 8 PIPT routines were run on the 5 PC's, all running a customized version of Knoppix 3.3. I customized the Knoppix 3.3 CD to include mpich and PIPT. The customization process[**7**] involved:
(1) Copying the Knoppix CD contents to hard disk
(2) chrooting into the copied contents
(3) removing large installed packages (such as OpenOffice)
(4) Installing mpich and adding PIPT
(5) Creating the compressed Knoppix file system
(6) Creating the Knoppix ISO image

PIPT's included Sample driver program was used with a batch file, using 1 manager node and 4 worker nodes for each PIPT routine. The manager node splits up each image and sends an image slice to each worker node for processing and then collects and recombines the processed image. The Sample driver program takes a batch file with a specific PIPT format. Results were stored in a flat text file. A perl script was used to parse the file for each routine's running time. The averages were then manually calculated. **Figure 2** contains the average running times of the chosen 8 PIPT routines. The total running time for the batch job was approximately 3 hours and 10 minutes.

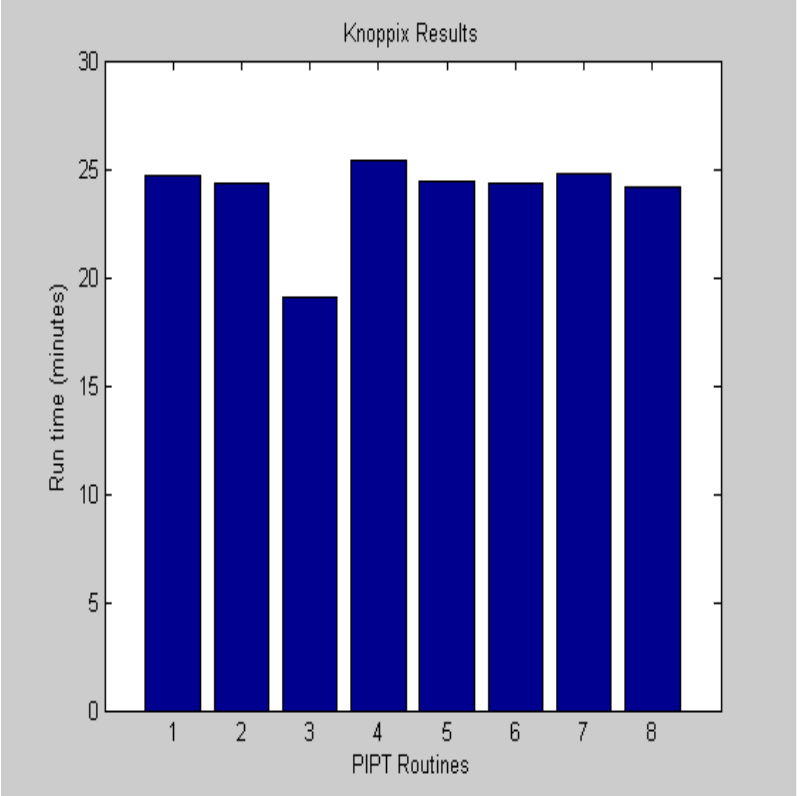

Figure 2: [1]CoOccurMax [2]CoOccurEnergy [3]CoOcuurContrast [4]CoOccurInverseMoment [5]CoOccurHomogeneity [6]CoOccurCorrelation [7]CoOccurEntropy [8]CoOccurCluster

5 Quantian Test Results

Each PIPT routine was run using PIPT's include Sample driver program. Instead of having a manager/worker relationship, each node worked on a completing a single PIPT routine. In order to allow for process migration, the first 5 PIPT routines were run simultaneously on one node (labeled 34). Then the next 3 PIPT routines were run simultaneously. Results were stored in a flat text file. A perl script was used to parse the file for each routine's running time. **Figure 3** contains the running times of the chosen 8 PIPT routines. The total running time for all routines was approximately 3 hours and 4 minutes.

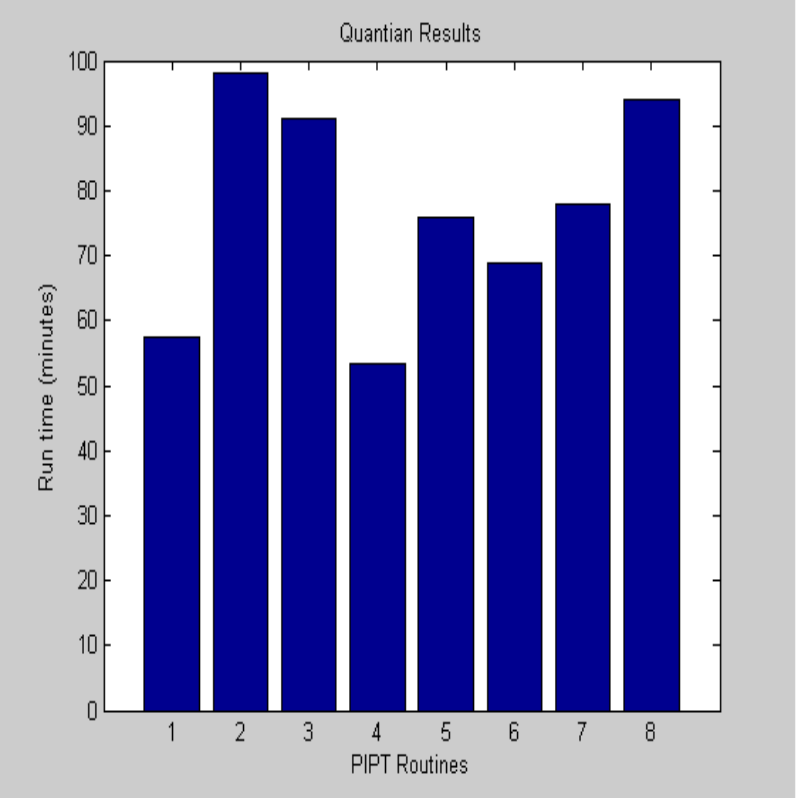

Figure 3: [1]CoOccurMax [2]CoOccurEnergy [3]CoOcuurContrast [4]CoOccurInverseMoment [5]CoOccurHomogeneity [6]CoOccurCorrelation [7]CoOccurEntropy [8]CoOccurCluster

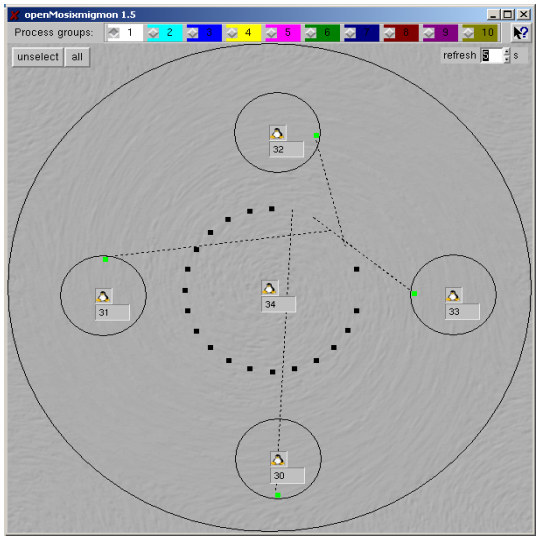

Figure 4: snapshot of initial 5 PIPT routines

5.1 Process migration

Soon after the first 5 PIPT routines were run on the master node, 4 out of 5 of them migrated to another node. Impressively, load balancing occurred relatively fast, with each node having one PIPT process. **Figure 4** contains a snapshot of the load of each node when observed by the openMosix userland tool **openMosixmigmon** soon after running the initial 5 PIPT routines. The center node (labeled 34) contains one PIPT routine, while nodes 30-33 each contain one migrated PIPT routine each represented by dotted lines moving from the center circle (node 34) to the surrounding smaller circles (nodes 30-33). All 4 migrated PIPT routines remained in the node to which they initially migrated.

The next 3 PIPT routines had slightly different migration activity. Soon after running them on the master node, all 3 migrated to another node. However, 1 of the 3 PIPT processes then began a long series of migrations between two specific nodes. This process migrated 51 times before it finished running.**Figure 5** and **Figure 6** depict the two load states the cluster was in while the process migrated back and forth between the node labeled 30 and node labeled 32.

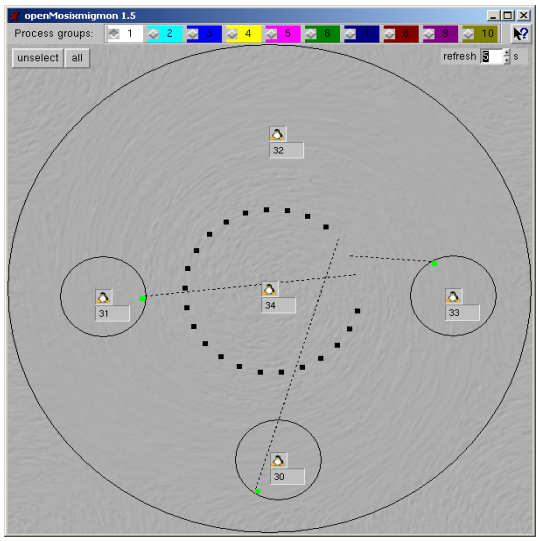

Figure 5: snapshot of initial load resulting from next 3 PIPT routines

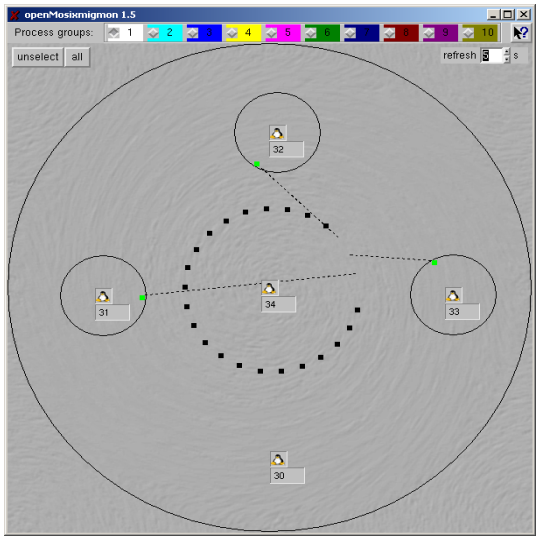

Figure 6: snapshot of subsequent load resulting from next 3 PIPT routines

6 Conclusion

The difference in run times for the 8 PIPT routines when run on a Knoppix cluster using mpich and when run on a Quantian cluster was negligible (less than 10 minutes). The individual run times of each PIPT routine when run on a Knoppix cluster using mpich were better simply because all 5 machines were working on the same job simultaneously, whereas in the Quantian setup each machine worked on one job at a time.

Because the cluster was heterogeneous, one would expect that Quantian's dynamic load balancing scheme would assign more work to nodes with more processing power or less RAM. However, because live cds make greater use of a machine's RAM for running applications, the load of each machine quickly shot up to 100% when a PIPT routine migrated to or was run on it, regardless of the amount of RAM each machine possessed. Quantian also offers explicit control of load balancing by the user. For example, the machines with more processing power or RAM could be assigned to handle more of the load of the entire cluster. Usage of this load balancing control could perhaps improve the performance of a heterogeneous network such as the one used in this paper.

Some features of PIPT that were not used include load-balancing and alternative modes of image slice distribution and collection. These features were not used because the Sample driver program used for testing did not offer them.

7 References

1. "Knoppix General FAQ", http://www.knoppix.net/docs/index.php/FaqGeneral, April 2004

2. Klaus Knopper, "Building a self-contained auto-configuring Linux system on an iso9660 filesystem", October 2000

3. Moshe Bar,Maya Kagliwal, Krushna Badge, Asmita Jagtap, Anuradha Khandekar, Snehal Mundle, "Introduction to openMosix", *Linux-Kongress*, 2003

4. J.M. Squyres, A. Lumsdaine, R.L. Stevenson, "A toolkit for parallel image processing",*SPIE Annual Meeting*, San Diego, 1998

5. David W. Walker, "The Design Of A Standard Message Passing Interface For Distributed Memory Concurrent Computer", *Parallel Computing*, Vol. 20, No. 4, pp 656-673, April 1994

6. “Maya Kagliwal, Krushna Badge, Asmita Jagtap, Anuradha Khandekar, Snehal Mundle,“MigSHM: Shared Memory over openMosix”, Cummins College of Engineering, University of Pune, India, 2002

7. “Knoppix Remastering Howto”, http://www.knoppix.net/docs/index.php/KnoppixRemasteringHowto, April 2004

8 Acknowledgements

I would like to thank my mother Patria, my brother Nestor, and my sister Amalfi for their support. I would like to thank Professor Stephen A. Edwards for overseeing this project and for his guidance.