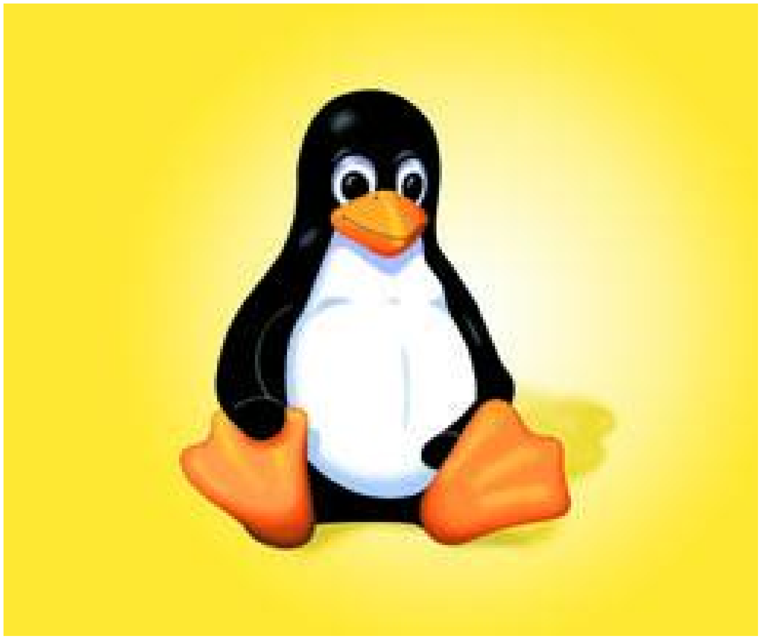
Institute of Fundamental Technological Research,
Polish Academy of Sciences,
Department of Mechanics and Physics of Fluids

Documentation on Linux clustering with openMosix



Version 1.0.1

Supervisor of the project: **Tomasz Kowalewski**

Administration: **Piotr Matejek**

Technical performers: **Ihor Trots** & **Vasyl Kovalchuk**

Cluster's web page: http://fluid.ippt.gov.pl/mosix/

Warsaw, 2004

Table of contents:

1 OpenMosix

Introduction

OpenMosix is a kernel extension for single-system image clustering. Clustering technologies allow two or more Linux systems to combine their computing resources so that they can work cooperatively rather than in isolation. OpenMosix is a tool for a Unix-like kernel, such as Linux, consisting of adaptive resource sharing algorithms. It allows multiple uniprocessors and symmetric multiprocessors (SMP nodes) running the same kernel to work in close cooperation. The openMosix resource sharing algorithms are designed to respond on-line to variations in the resource usage among the nodes. This is achieved by migrating processes from one node to another, preemptively and transparently, for load-balancing and to prevent thrashing due to memory swapping. The goal is to improve the cluster-wide performance and to create a convenient multiuser, time-sharing environment for the execution of both sequential and parallel applications. The standard runtime environment of openMosix is a computing cluster, in which the cluster-wide resources are available to each node.

The current implementation of openMosix is designed to run on clusters of X86/Pentium-based uniprocessors workstations that are connected by standard LANs. Possible configurations may range from a small cluster of PCs that are connected by 100Mbps Ethernet, to a high performance system.

Recommendations

For maximum cluster performance, you may want to consider building a cluster using the following components. These items are not absolutely required; some are completely optional (only for performance freaks), and others are recommended. We indicate which are which below:

- at least 100Mbit (fast) ethernet is recommended. Standard (10Mbit) ethernet won't give you very good cluster performance, but should be fine if you just want to play around with openMosix. Gigabit ethernet is optional but beneficial. However, don't feel that you absolutely *need* Gigabit ethernet; MOSIX can do just fine with fast Ethernet;
- hooking your machines' ethernet cards up to a dedicated high-speed switch is beneficial. By doing so, your systems will be able to communicate over ethernet in "full duplex" mode, effectively doubling bandwidth.
- a good amount of swap space is recommended. This will allow nodes to be removed from your cluster without causing the existing nodes from running out of virtual memory. Again, this is recommended and will only make a difference in extreme situations where you are pushing your cluster very hard.

If you have a limited number of machines, you may want to consider using a specially-wired ethernet cable to directly connect the systems to one another. By doing so, you can benefit from "switch-like" full-duplex performance at a potentially lower price. This trick is very helpful when used for 2 or 3-node clusters, since these configurations only require one or two NICs per machine respectively.

The openMosix solution

OpenMosix works in a fundamentally different way that PVM or MPI, extending the kernel so that *any* standard Linux process can take advantage of a cluster's resources. By using special adaptive load-balancing techniques, processes running on one node in the cluster can be transparently "migrated" to another node where they may execute faster. Because openMosix is transparent, the process that's migrated doesn't even "know" (or *need* to know) that it is running on a remote system. As far as that remote process and other processes running on the original node (called the "home node") are concerned, the process is running locally.

Since openMosix is completely transparent, no special programming is required to take advantage of openMosix's load-balancing technology. In fact, a default openMosix installation will automatically migrate processes to the "best" node without any user intervention, making openMosix an ideal turn-key clustering solution that can be of great benefit to very many people.

OpenMosix limitations -- processes

In addition, openMosix, like an SMP system, cannot cause a single process to run on multiple CPUs at the same time. So, openMosix won't be able to speed up a single process such as Netscape, except to migrate it to a node where it can execute most efficiently. In addition, openMosix can't currently allow multiple threads to execute on separate systems.

1.1 Installing openMosix

Installation overview

OpenMosix installation is relatively painless. To get openMosix up and running, we must first install a openMosix-enabled kernel on all the Linux systems that will be joined together into a single cluster. Then, we need to install the openMosix user tools. Finally, we must make the necessary changes to each nodes' system configuration files in `/etc`, and then reboot. When the systems come back up, openMosix process migration will be active and your cluster will be ready for use.

Downloading openMosix sources

To start the openMosix installation, first head over to *http://openmosix.sourceforge.net*, and click on the download and install the latest release of openMosix link. On this page, you'll be able to see the various versions of openMosix available. We'll be using openMosix for the 2.4 kernel, although there is a version of openMosix available for the 2.2 kernel as well.

Under "The latest distribution" section, you should see a link to download openMosix 2.4.x for kernel 2.4.y; x and y will vary as new kernel and openMosix releases are made available. Go ahead and download the openMosix source; at the time this documentation was written, the most recent openMosix source was openMosix-kernel-2.4.26.bz2.

OpenMosix and the kernel

Now, make a note of the Linux kernel version with which this particular version of openMosix was designed to operate. You *always* want to pair openMosix with the correct Linux kernel to ensure stable operation. In addition, it's highly recommended that you *do not* apply any additional non-trivial patches to your Linux kernel besides the openMosix patches themselves. Once you've downloaded the correct openMosix source, go ahead and download the appropriate 2.4 kernel from *http://www.kernel.org/pub/linux/kernel/v2.4*. The version of openMosix that we're using over here (2.4.24) was designed to work with Linux 2.4.24, so we went ahead and downloaded linux-2.4.24.tar.gz.

Patching the kernel

After you've downloaded the correct kernel for your version of openMosix, it's time to extract your kernel sources and apply the openMosix patch. Here's how.

```
mv linux linux.old (if linux is a directory)
rm linux (if linux is a symlink)
mv openMosix-2.4.24-2.bz2 /usr/src/linux
cd /usr/src/linux
bunzip openMosix-2.4.24-2.bz2
```

Now, we apply the openMosix patch:

```
cd /usr/src/linux
cat openMosix-1. | patch –Np1
```

Now that the patch is applied is applied to the kernel sources, we're ready to configure, compile and install a new openMosix-enabled Linux kernel.

Kernel configuration overview

Now, it's time to configure the kernel:

```
cd /usr/src/linux (if you aren't there already)
make menuconfig
```

You'll be greeted with the blue Linux kernel configuration screen. Go ahead and configure your kernel with the options it needs to run on your underlying hardware. When you're done, head over to the new "`openMosix`" configuration section, which should be the first configuration category listed. In the following panels, We'll explain all the important openMosix options:

```
[*] openMosix process migration support
```

This option enables openMosix proper. It's required.

Complex network option

```
[ ] Support clusters with a complex network topology
```

Enable this option if the nodes on your cluster are not on a simple LAN, or if the network cards you are using in your nodes vary widely in their performance characteristics. If you are using similar or identically-performing computers (nodes) throughout your cluster, and all machines are an equal "distance" away (from a network perspective), then you should leave this option disabled for increased performance.

Security and MFS options

```
    [*] Stricter security on openMosix ports
```

Stricter security on OpenMosix ports is optional. This option will guard openMosix's TCP/UDP ports against being abused by people on hosts outside the cluster. This is highly recommended if your systems are addressable by systems that are not part of your cluster. Remote (guest) processes are not accessible to the other processes that run at the same node (locally or originated from other nodes) and vice versa. They do not belong to any particular user (on the remote node, where they run) nor can they be sent signals or otherwise manipulated by local processes. Their memory cannot be accessed and they can only be forced, by the local system administrator, to migrate out.

```
    [*] openMosix File-System
```

This enables the openMosix filesystem, a handy cluster filesystem that can be used to share and copy files throughout your cluster. MFS is very highly recommended and is a very handy addition to openMosix.

DFSA option

```
    [*] Direct File-System Access
```

Enabling this option can allow increased performance for processes migrated away from their "`home`" node. It's a nice extension, but is still considered experimental. You can choose to enable it for improved performance in certain situations, or leave it disabled.

Finishing up config

Now, go ahead and save your kernel configuration options. An important note**:**
make sure that your openMosix configuration options are identical throughout your entire cluster. Other kernel options may vary due to hardware differences, but openMosix will expect all nodes to have identical openMosix functionality. In addition, all nodes of your cluster must use the same version of openMosix and the Linux kernel. Besides this requirement, feel free to mix different Linux distributions.

Kernel installation tricks

Now, compile and install the kernel you just configured.

```
make dep clean bzImage modules modules_install
```

After compilation install the new kernel with the openMosix options within you boot−loader e.g. insert an entry for the new kernel in `/etc/lilo.conf` and run lilo after that.

Reboot and your openMosix−cluster (node) is up!

Repeat this process on all remaining nodes in your cluster. You may want to copy your `/usr/src/linux` directory over to the other systems to speed things up. Otherwise, copy the `/usr/src/linux/.config` file from your current node to any new nodes before typing "`make menuconfig`". That way, you'll start with your previous node's kernel configuration, ensuring that all your openMosix kernel configuration options are consistent throughout the entire cluster.

Installing user tools

Now we're ready to install the openMosix user tools. The openMosix-tools can be downloaded from the same web page as openMosix source. First, extract the openMosix-tools into Linux directory:

```
mv openMosix-tools-0.3.6.2.tar.gz /usr/src/linux
cd /usr/src/linux
tar –zxvf openMosix-tools-0.3.6.2.tar.gz
ln –s /usr/src/linux /usr/src/linux-openmosix
cd /usr/src/linux/openmosix-tools-0.3.6-2
./configure
make
make check
make install
```

Once you've installed the openMosix user tools on every node in your cluster.

If you prefer to install the openMosix-tools without using the RPM, you'll have to do some editing by yourself. Note: Even if you install the tools with an RPM, you can still roll your own kernel, so even for the developer there's no need to install the tools by hand. After running "`make install`", you should perform the following steps:

- edit `/etc/inittab`:

The lines that starts `/etc/rc.d/rc*`, `/sbin/update` and `/sbin/shutdown` should be changed so they're instead started with `/bin/mosrun -h`.

Example:

```
    si::sysinit:/etc/rc.d/rc.sysinit
should be changed to
    si::sysinit:/bin/mosrun -h /etc/rc.d/rc.sysinit
```

If you installed `mosrun` in a directory other than `/bin`, please make sure that directory gets mounted at boot time or your system might not be able to find the `mosrun` executable while it's in the early init stage (which basically means that your system would hang shortly after the kernel starts up).

- edit `/etc/rc.d/init.d/sshd`

Since now all daemons are locked from migration, all their children will be too. This makes all processes started from a ssh-login also being locked from migration. Solution: In the beginning of the `start()` function, put this line:

```
test -f /proc/$$/lock && echo 0 > /proc/$$/lock
```

- edit `/etc/cron.daily/slocate.cron`

You don't want `slocate` to look in the MFS-filesystem (if you choose to try that). Put `mfs` in there with `nfs` and the others.

1.2 Configuration steps openMosix

/etc/openmosix.map

Let's get the configuration files set up now. First, we'll need to create a file called "`/etc/openmosix.map`" that will describe all the nodes in our cluster. When you're done editing this file, it should be copied verbatim to every node. All `/etc/openmosix.map` files should be identical throughout the entire cluster.

The format of the `openmosix.map` file is simple. Each line should contain a node number, the IP address or hostname of the node, and a "span", normally one. If a node has more than one network–interfaces it can be configured with the `ALIAS` option in the range–size field. For example, this openmosix.map file defines a eight-node cluster:

```
1 148.81.54.xx ALIAS
1 10.0.0.1 1
2 10.0.0.2 1
. . . . . .
8 10.0.0.8 1
```

Always be sure to run the same openMosix version and configuration on each of your Cluster nodes!

Start openMosix with the "`setpe`" utility on each node on every node in your openMosix cluster:

```
/usr/local/sbin/setpe –w –f /etc/openmosix.map
```

Installation finished now, the cluster is up and running :)

MFS

Next, we need to create a mount point and `/etc/fstab` entry for MFS, the openMosix filesystem. First, create a `/mfs` directory on each node:

```
mkdir /mfs
```

Then, add an `mfs` entry to your `/etc/fstab` file:

```
cluster    /mfs mfs  dfsa=1    0 0
```

Understanding MFS

So, what does MFS do? Well, when we reboot and openMosix is enabled, you'll be able to access filesystems throughout your cluster by perusing directories inside `/mfs`. For example, you can access the `/etc/hosts` file on node 2 by editing `/mfs/2/etc/hosts`, etc. MFS is extremely handy, and openMosix also takes advantage of MFS to improve performance.

Network configuring

Slackware Linux stores gateway configuration information in the file '`/etc/rc.d/rc.inet1.conf`', which contains next variables:

```
# Config information for eth0:
IPADDR[0]="148.81.54.xx"
NETMASK[0]="255.255.255.0"
# Config information for eth1:
IPADDR[1]="10.0.0.1"
NETMASK[1]="255.0.0.0"
# Default gateway IP address:
GATEWAY="148.81.54.254"
```

System startup

Before we restart all the nodes in our new cluster, we need to add a couple of lines to each node's system startup scripts. First, add the following lines to your `/etc/rc.d /local` startup file, so that the following commands are run at startup:

```
setpe -w -f /etc/openmosix.map

/etc/init.d/openmosix start
```

These commands initialize the openMosix node for use, and should execute *after* the local MFS filesystem has been mounted and *after* the network has been set up. Typically, if they are added to a "`local`" startup script, they'll execute at the right time.

Shutdown

One of the nice things about openMosix is that it's perfectly safe to reboot a node while your cluster is in use. Any remote processes will simply migrate away before your system reboots.

1.3 OpenMosix in action

Determining cluster status

From your master node, you should be able to run any of the native status or monitoring commands to check on the current cluster status. `mosmon` provides fundamental node resource information;

```
mosmon
```

`mtop` is the openMosix version of “`top`”, which shows where processes are in the cluster and what CPU and memory resources they are consuming;

```
mtop
```

`showmap` will dump a text copy of the cluster node database;

```
showmap
```

Testing the cluster

Since you just rebooted all the machines, there probably isn't much happening on the nodes at the moment. So, let's do a little openMosix test so that we can see process migration in action. To do this, we're going to create several very CPU-intensive processes on the local system, and then watch as openMosix migrates these processes to the most efficient node. While keeping the "`mon`" program running, open at least two new shells, and run the following command in each of these shells:

```
awk 'BEGIN {for(i=0;i<10000;i++)for(j=0;j<10000;j++);}'
```

Understanding migration

With at least two of these commands running, head back to "`mon`". You should see the system load on your current node shoot up to 2 or more. But, after a few seconds, you should see the load on your current node drop and the load on another one or two of your opexMosix nodes increase. Congratulations; you've just witnessed openMosix process migration in action! OpenMosix detected that the new processes you created could run faster if some of them were migrated to other nodes in your cluster, and openMosix did just that.

As far as the migrated processes are concerned, they're running on your local node, also called the process's "`home`" node. They have no idea that they are actually running on a remote CPU. In fact, the processes will still be listed in their home node's "`ps`" list, and they won't even show up in the remote node's "`ps`" list.

1.4 OpenMosixview

Introduction

OpenMosixview is cluster−management GUI for openMosix cluster and everybody is invited to download and use it. The openMosixview−suite contains 5 useful applications for monitoring and administrating openMosix cluster.

- *openMosixview* the main monitoring+admistration application

- *openMosixprocs* a process−box for managing processes

- *openMosixcollector* collecting daemon which logs cluster+node informations

- *openMosixanalyzer* for analyzing the data collected by the openMosixcollector

- *openMosixhistory* a process−history for your cluster

All parts are accessible from the main application window. The most common openMosix commands are executable by a few mouse clicks. An advanced execution dialog helps to start applications on the cluster. "Priority−sliders" for each node simplifying the manual and automatic load−balancing. OpenMosixview is fully designed for openMosix cluster only. The openMosixview website and all further developing is located at the domain *http://www.openmosixview.com/*.

Requirements

`QT >= 2.3.0`

`root` rights !

`rlogin` and `rsh` (or `ssh`) to all cluster nodes without password the openMosix userland tools `mosctl`, `migrate`, `runon`, `iojob`, `cpujob` ... (download them from the www.openmosix.org website)

Documentation about openMosixview is a full HTML−documentation that included in every package. You find the start page of the document in your openMosixview installation directory:

```
openmosixview-1.5/docs/index.html
```

Installation

After you've downloaded the latest version of openMosixview, it's time to extract openMosixview sources and install it. Here's how.

```
mv openmosixview-1.5.tar.gz /usr/local
tar –zxvf openmosixview-1.5.tar.gz
cd openmosix-1.5
./setup % for automatic setup−script
```

Manual compiling

Set the QTDIR-Variable to your actual QT-Distribution, e.g.

```
   export QTDIR=/usr/lib/qt-3.3.3 (for bash)
or
   setenv QTDIR /usr/lib/qt-3.3.3 (for csh)
```

Create the link `/usr/lib/qt` pointing to your QT-3.3.3 installation e.g. if QT-3.3.3 is installed in `/usr/local/qt-3.3.3`

```
   ln -s /usr/local/qt-2.3.0 /usr/lib/qt
```

Then you have to set the QTDIR environment variable to

```
   export QTDIR=/usr/lib/qt (for bash)
or
   setenv QTDIR /usr/lib/qt (for csh)
```

After that the rest should work fine:

```
make
make install
```

Then do the same in the subdirectory `openmosixcollector`, `openmosixanalyzer`, `openmosixhistory, openmosixmigmon, openmosixpidlog.` Copy all binaries to `/usr/bin`

```
cd openmosixview-1.5
cp openmosixview/openmosixview /usr/bin
cp openmosixprocs/openmosixprocs /usr/bin
cp openmosixcollector/openmosixcollector /usr/bin
cp openmosixanalyzer/openmosixanalyzer /usr/bin
cp openmosixhistory/openmosixhistory /usr/bin
cp openmosixmigmon/openmosixmigmon /usr/bin
cp openmosixpidlog/openmosixpidlog/usr/bin
```

And copy the `openmosixcollector` init−script to your init−directory, e.g.,

```
   cp openmosixcollector/openmosixcollector.init
      /etc/init.d/openmosixcollector
or
   cp openmosixcollector/openmosixcollector.init
      /etc/rc.d/init.d/openmosixcollector
```

Now copy the `openmosixprocs` binary on each of your cluster-nodes to `/usr/bin/`

```
 rcp openmosixprocs/openmosixprocs
     your_node:/usr/bin/openmosixprocs
```

You can now execute `openmosixview`

```
openmosixview
```

2 IP Masquerading and IP-tables

If a guarded configuration is implemented and it is necessary for nodes to contact the outside world, network address translation is the best option. Network Address Translation, commonly referred to as NAT, is a technique devised for reusing IP addresses as a stopgap measure to slow the depletion of the address space. NAT permits IP address reuse by allowing multiple networks to use the same addresses but having them communicate between each other through a pair of non-shared IP address. IP masquerading is a type of NAT performed by the worldly node of a openMosix cluster that makes external network connections appear to originate from the single worldly node. This feature allows the internal nodes to originate network connections to external Internet hosts but provides security by not having a mechanism for an external host to set up a connection to an internal node.

A nice feature about IP masquerading is that it doesn't involve too many steps to set up. Only the node performing the masquerading requires any amount of reconfiguration. The internal nodes simply require their default route to be set to the internal network address of the worldly node. You can do this with the route command as follows:

```
   route add default gw 10.0.0.1
```

These should be set to the IP address of the worldly node and the primary internal network interface name of the internal node. A typical network configuration file for an internal node might look something like the following:

```
# Config information for eth0: (for node02 in given case)
IPADDR[0]="10.0.0.2"
NETMASK[0]="255.0.0.0"
# Default gateway IP address:
GATEWAY="10.0.0.1"
```

Configuring the worldly node to perform the IP masquerading requires more work. The first requirement is to compile your kernel with support for network firewalls, IP forwarding/gatewaying, and IP masquerading. There are also some additional options you may wish to include, but these are the essential ones.

After installing a kernel capable of IP masquerading, you need to enable IP forwarding. IP forwarding is the process by which a host will forward to its destination a packet it receives for which it is not itself the destination. This allows internal node packets to be forwarded to external hosts.

The last step is to configure IP masquerading rules. You don't want your worldly node to forward packets coming from just anywhere, so you have to tell it specifically what packets to forward. Currently, you can do this by using the `iptables` utility with 2.4.x Linux kernels; this is the evolution of the pre-2.4.x `ipfwadm` and `ipchains` utilities.

The program `iptables` configures firewall packet filtering and forwarding rules. It can specify rules based on the source and destination of a packet, the network interfaces on which a packet is received or transmitted, the network protocol used, destination and source ports, and quite a bit of other information. For the purposes of setting up a worldly node, you can use `iptables` to tell the kernel to masquerade only for packets originating from an internal node.

Downloading iptables sources

To start the `iptables` installation, first head over to http://www.netfilter.org, and click on the download and install the latest release of `iptables` link. On this page, you'll be able to see the various versions of `iptables` available. We'll be using `iptables-1.2.11:` http://www.netfilter.org/downloads.html#iptables-1.2.11.

2.1 Applying `iptables` and `Patch-o-Matic` kernel patches

As both the 2.4.x kernel train and the `iptables` program development progresses, the compile configuration options will change over time. If you are compiling against a newer or previous kernel or `iptables` version, the dialogs and even commands might look different. It is recommended that you update to the newest versions of both the kernel and `iptables` for added capability, performance, and stability of the kernel.

Next, depending on the version of the Linux kernel and `iptables` archive you downloaded, you might want to apply some `iptables` "patch-o-matic" patches against the kernel. These optional patches might fix some known problems or add additional functionality you might need (H.323 protocol, specific issues with network games, etc.). It should be noted that the Patch-O-Matic patches used to come with the `iptables` archive. This is no longer the case and you have to download them (if any) seperately. You can find the various URLs for downloading `iptables`, the Patch-o-matic system (patch-o-matic-ng-20040621.tar.bz2), e.g. http://www.netfilter.org/downloads.html#iptables-1.2.11.

You might consider applying any appropriate or optional patches to the kernel's MASQ code before you compile the final kernel. The IP MASQ code found in the stock kernels is already very useful and does not require any specific patching in order for the system to work for NAT-friendly network applications. Many of these patches are only to fix possible known bugs, add new features (some are /very/ cool), etc.

Put the `iptables` package and optional Patch-O-matics into a directory, say "`/usr/local`". Next, go into this new netfilter directory and uncompress the `iptables` archive with the command:

```
mv iptables-1.2.11.tar.bz2 /usr/local
cd /usr/src/local
tar xyvf iptables-1.2.11.tar.bz2
tar xyvf patch-o-matic-ng-20040621.tar.bz2
```

Now, go into the new `iptables-1.2.11` directory and run the command

```
cd iptables-1.2.11
make KERNEL_DIR=/usr/src/linux
```

After compiling `iptables-1.2.11`, you need to install it. To do this, run the command

```
 Make install KERNEL_DIR=/usr/src/linux
```

Next, if you are interested in applying a `Patch-O-Matic` patch set, go into the `patch-o-matic-X` directory and run the command

```
cd /usr/local/patch-o-matic-X
KERNEL_DIR=/usr/src/linux
./runme pending
```

Note: this assumes that your 2.4.x kernel sources are in the `/usr/src/linux` directory.

You can also run the "runme" program in a batch mode to speed things up, add experimental patches, etc. if you'd like. To better understand your options, simply run the "`./runme`" command by itself. Please note that these prompts will change over time.

If everything patches fine, you should see something like the text

```
Excellent! Kernel is now ready for compilation.
```

towards the bottom of the screen. Beyond that, you don't have to install anything at this point. The next step is to compile the new patched kernel.

Ok, now the new kernel is ready to be compiled but you should make sure that you also have the proper matching `iptables` program on your machine too. Run the command:

```
whereis iptables
```

and make sure its installed on the machine (the default place is in `/usr/local/sbin/iptables`). If you cannot find it or patched up your kernel sources as shown above, it is recommended just to re-compile it up as shown above.

Now that the kernel sources are patched up, you need to configure it to know what kinds of features you need (HD support, Networking support, MASQ support, etc.). Here are the minimum kernel configuration options required to enable `IP Masquerade` functionality. Basically, compiling things as modules gives you added flexibility to what is or isn't installed into the kernel (reduces unneeded memory use for things you aren't/won't use and modules also allow for drop-in software upgrades [usually no need to reboot the machine]). On the flip side, kernel modules add more complexity to your configuration and sometimes the kernel auto-loader might make mistakes (not that I've ever seen this happen). Compiling things directly into the kernel makes things simpler but you loose a huge level of flexibility. The following kernel configuration example is a mixture of both a selection of kernel modules and building them in monolithically (you probably will always need MASQ functionality ready to go).

You will need to answer either 'yes', 'no' or 'module' to the following program. Not all options will be available without the proper kernel patches. This shouldn't be an issue as most 3rd party patches are only needed for a very select group of users.

Run the following commands to configure your kernel:

```
cd /usr/src/linux
make menuconfig
```

Please note the following kernel prompts reflect a 2.4.24 kernel (with some of the optional `Patch-O-Matic` additions). Please read the following carefully for recommendations:

```
[ Code maturity level options ]


 * Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) [Y/n/?]
    - YES: though not required for IP MASQ, this option allows the kernel to create
           the MASQ modules and enable the option for port forwarding

  * Enable loadable module support (CONFIG_MODULES) [Y/n/?]
    - YES: allows you to load kernel IP MASQ modules

  * Set version information on all module symbols (CONFIG_MODVERSIONS) [Y/n/?]
    - YES: allows newer kernels to load older modules if possible

  * Kernel module loader (CONFIG_KMOD) [Y/n/?]
 -      OPTIONAL: Recommended : allows the kernel to load various kernel modules as
                  it needs them

  == Non-MASQ options skipped
  ==   (CPU type, memory, SMP, FPU, specific stuff)

[ General setup ]

  * Networking support (CONFIG_NET) [Y/n/?]
    - YES: Enables the network subsystem

  == Non-MASQ options skipped
  ==   (specific hardware, PCI, kernel binaries, PCMCIA, etc.)


  * Sysctl support (CONFIG_SYSCTL) [Y/n/?]
    - YES:  Enables the ability to enable disable options such as forwarding,
            dynamic IPs, etc. via the /proc interface

 [ Block devices ]

  == Non-MASQ options skipped
  ==   (kernel binaries, power management, PnP, RAID, etc.)

    == Don't forget to compile in support for hardware that you might need:
    ==   IDE controllers, HDs, CDROMs, etc.
```

```
[ Networking options ]

  * Packet socket (CONFIG_PACKET) [Y/m/n/?]
    - YES: Though this is OPTIONAL, this recommended feature will allow you
           to use TCPDUMP to debug any problems with IP MASQ

  * Packet socket: mmapped IO (CONFIG_PACKET_MMAP) [N/y/?] y
    - YES: Speed up the packet protocol

  * Kernel/User netlink socket (CONFIG_NETLINK) [Y/n/?]
    - OPTIONAL:  Recommended : this feature will allow the logging of
                 advanced firewall issues such as routing messages, etc

  * Routing messages (CONFIG_RTNETLINK) [N/y/?] (NEW) y
    - OPTIONAL: Allows for support of advanced kernel routing messages
                if you enabled the CONFIG_NETLINK option

  * Netlink device emulation (CONFIG_NETLINK_DEV) [N/y/m/?] (NEW)
    - NO:  This option does not have anything to do with packet firewall
           logging

  * Network packet filtering (replaces ipchains) (CONFIG_NETFILTER) [N/y/?] y
    - YES: Enable this option to let IPTABLES configure the TCP/IP subsection
           of the kernel.  By enabling this, then you can turn on advanced
           routing mechanisms like IP Masq, packet filtering, etc.

  * Network packet filtering debugging (CONFIG_NETFILTER_DEBUG) [N/y/?] (NEW) n
    - NO: Not required for Masquerading functionality though it may help
          for troubleshooting.  There might be a performance penalty when
          enabling this.

  * Socket Filtering (CONFIG_FILTER) [Y/n/?]
    - OPTIONAL:  Recommended : Though this doesn't have anything do with IPMASQ,
                 if you plan on implimenting a DHCP server on the internal network,
                 you will need to enable this option.

  * Unix domain sockets (CONFIG_UNIX) [Y/m/n/?]
    - YES:  This enables the UNIX TCP/IP sockets mechanisms

  * TCP/IP networking (CONFIG_INET) [Y/n/?]
    - YES: Enables the TCP/IP protocol

  * IP: multicasting (CONFIG_IP_MULTICAST) [N/y/?]
    - OPTIONAL:  You can enable this if you want to be able to receive
                 Multicast traffic.  Please note that your ISP must
                 support Multicast as well for this all to work at all

  * IP: advanced router (CONFIG_IP_ADVANCED_ROUTER) [Y/n/?]
    - OPTIONAL:  Though there is nothing in this section mandatory for
                 Masquerade, some specific options might be useful

    == Non-MASQ options skipped
    ==   ( autoconf, tunneling )

  * IP: multicast routing (CONFIG_IP_MROUTE) [N/y/?] n
    - OPTIONAL:  Though not needed for IPMASQ, enabling this feature will
                 let you route multicast traffic through your Linux box.
                 Please note that this requires that your ISP be multicast
                 enabled as well.

    == Non-MASQ options skipped
    ==   (ARPd)

  * IP: TCP Explicit Congestion Notification support (CONFIG_INET_ECN) [N/y/?] n
    - NO: Though enabling this option would be great, there are many Internet
          sites out there that will block this.  Hit the "?" when configuring the
          kernel to learn more about it but it is recommended to say NO for now.

  * IP: TCP syncookie support (disabled per default) (CONFIG_SYN_COOKIES) [Y/n/?]
    - YES: Recommended : for basic TCP/IP network security
```

```
[ Networking options --> IP: Netfilter Configuration ]


  * Connection tracking (required for masq/NAT) (CONFIG_IP_NF_CONNTRACK) [N/y/m/?] (NEW) m
    - YES: (Module) This enables the kernel to track various network connections.
           This option is required for Masquerading support as well as to enable
           Stateful tracking for various filewall mechanisms.  Please note that
           if you compile this directly into the kernel, you cannot enable
           the legacy IPCHAINS or IPFWADM compatibility modules.

  * FTP protocol support (CONFIG_IP_NF_FTP) [M/n/?] (NEW) m
    - YES: (Module) This enables the proper Masquerading of FTP connections if
           CONFIG_IP_NF_CONNTRACK was enabled above

  * IRC protocol support (CONFIG_IP_NF_IRC) [M/n/?] (NEW) m
    - YES: (Module) This enables the proper Masquerading of IRC connections if
           CONFIG_IP_NF_CONNTRACK was enabled above

  * Userspace queueing via NETLINK (EXPERIMENTAL) (CONFIG_IP_NF_QUEUE) [N/y/m/?] (NEW) m
    - OPTIONAL: Though this is OPTIONAL, this feature will allow IPTABLES to
                copy specific packets to UserSpace tools for additional checks

  *  IP  tables  support  (required  for  filtering/masq/NAT)  (CONFIG_IP_NF_IPTABLES)  [N/y/m/?]
     (NEW) m
    - YES: (Module) Enables IPTABLES support

  * limit match support (CONFIG_IP_NF_MATCH_LIMIT) [N/y/m/?] (NEW) y
    - OPTIONAL:  (Module) Recommended : Though not required, this option can used to
                 enable rate limiting of both traffic and loggin messages help
                 slow down denial of service (DoS) attacks.

  * MAC address match support (CONFIG_IP_NF_MATCH_MAC) [N/y/m/?] (NEW) m
    - OPTIONAL:  Though not required, the option can allow you to
                 filter traffic based upon the SOURCE Ethernet MAC address.

  * netfilter MARK match support (CONFIG_IP_NF_MATCH_MARK) [N/y/m/?] (NEW) y
    - YES: (Module) Recommended : This enables IPTABLES to take action upon marked packets.
           This mechanism can allow for PORTFW functionality, TOS marking, etc.

  * Multiple port match support (CONFIG_IP_NF_MATCH_MULTIPORT) [N/y/m/?] (NEW) y
    - YES: (Module) Recommended : This enables IPTABLES to accept mutliple SRC/DST port
           ranges (non-contiguous) instead of one port range per IPTABLES
           statement.

  * TOS match support (CONFIG_IP_NF_MATCH_TOS) [Y/m/n/?] n
    - OPTIONAL:  This allows IPTABLES to match packets based upon their
                 DIFFSERV settings.

  * LENGTH match support (CONFIG_IP_NF_MATCH_LENGTH) [N/m/?] (NEW) n
    - OPTIONAL:  This allows IPTABLES to match packets based upon their
                 packet length.

  * TTL match support (CONFIG_IP_NF_MATCH_TTL) [N/m/?] (NEW) ? n
    - OPTIONAL:  This allows IPTABLES to match packets based upon their
                 TTL settings.

  * tcpmss match support (CONFIG_IP_NF_MATCH_TCPMSS) [N/y/m/?] m
    - OPTIONAL: (Module) Recommended :  This option allows users to examine the MSS value in
                 TCP SYN packets.  This is an advanced knob but can be very valuable in
                 troubleshooting MTU problems.

  * Connection state match support (CONFIG_IP_NF_MATCH_STATE) [M/n/?]  m
    - YES: (Module) Recommended : This option allows for Stateful tracking of network
            connections.

  * Unclean match support (EXPERIMENTAL) (CONFIG_IP_NF_MATCH_UNCLEAN) [N/y/m/?] y
    - YES: (Module) Recommended :  This option allows for connection tracking on odd packets.
           It cal also help in the detection of possibly malicious packets.
           This can be a valuable tool in tracking hostile people on the network.
  * Owner match support (EXPERIMENTAL) (CONFIG_IP_NF_MATCH_OWNER) [N/y/m/?] n
    - OPTIONAL:  This option allows IPTABLES to match traffic based upon the
                 user login, group, etc. who created the traffic.

  * Packet filtering (CONFIG_IP_NF_FILTER) [N/y/m/?] ? y
    - YES: (Module) This option allows for the kernel to be able filter traffic at
            the INPUT, FORWARDING, and OUTPUT traffic points.

    * REJECT target support (CONFIG_IP_NF_TARGET_REJECT) [N/y/m/?] (NEW) y
      - YES: (Module) With this option, a packet firewall can send an ICMP Reject packet
              back to the originator when a packet is blocked.

  * MIRROR target support (EXPERIMENTAL) (CONFIG_IP_NF_TARGET_MIRROR) [N/y/m/?] (NEW) n
    - OPTIONAL: This option allows the packet firewall to mirror the exact same
                network packet back to the originator when it is supposed to be
                blocked.  This is similar to the REJECT option above but it actually
                sends the original packet back to the originator.  i.e. a
                hostile user could actually portscan themselves.


  * Full NAT (CONFIG_IP_NF_NAT) [M/n/?] m
    - YES: (Module) This option enables the future menus to enable Masquerading,
           PORTFWing, Full (1:1) NAT, etc.


  * MASQUERADE target support (CONFIG_IP_NF_TARGET_MASQUERADE) [M/n/?] (NEW) m
    - YES: (Module) This option specifically enables Masquerade into the
           kernel

  * REDIRECT target support (CONFIG_IP_NF_TARGET_REDIRECT) [N/y/m/?] n
    - OPTIONAL: Not needed for normal MASQ functionality though people who
                want to do transparent proxy via Squid will want this.

  * Basic SNMP-ALG support (EXPERIMENTAL) (CONFIG_IP_NF_NAT_SNMP_BASIC) [N/m/?] n
    - OPTIONAL: This enables IPTABLES to properly NAT internal SNMP packets so
                that machines with duplicate addressing ranges can be properly
                managed.


  * Packet mangling (CONFIG_IP_NF_MANGLE) [N/y/m/?] y
    - YES: (Module) This option allows for advanced IPTABLES packet manipulation
           options.


  * TOS target support (CONFIG_IP_NF_TARGET_TOS) [N/y/m/?] (NEW) n
    - OPTIONAL: Enables the kernel to modify the TOS field in a packet
                before routing it on

  * MARK target support (CONFIG_IP_NF_TARGET_MARK) [N/y/m/?] (NEW) m
    - OPTIONAL: (Module) Recommended : This enables the kernel to manipulate
                packets based upon the MARK field.  This can be used for PORTFW
                as well as many other things.

  * LOG target support (CONFIG_IP_NF_TARGET_LOG) [N/y/m/?]  m
    - YES: (Module)  This allows for the logging of packets before they are accepted,
           denied, rejected, etc.

  * TCPMSS target support (CONFIG_IP_NF_TARGET_TCPMSS) [N/y/m/?] ? m
    - YES: (Module) This option help some people with MTU problems.  Typically,
           most users have to set their Internet connection's MTU to
           1500 as well as ALL internal machines to 1500.  With this
           option, this whole MTU issue might be finally solved.

  * ipchains (2.2-style) support (CONFIG_IP_NF_COMPAT_IPCHAINS) [N/y/m/?] m
    - OPTIONAL: (Module) Recommended : If you have an existing IPCHAINS ruleset
                (2.2.x kernels) and enable this option, you can continue to use the
                IPCHAINS program and the majority of your old ruleset except for the use
                of any 2.2.x kernel-specific modules. Please note that if this IPCHAINS
                module is loaded, ALL IPTABLES modules will be non-operational. This
                is an either/or deal only intended for legacy rulesets.
```

```
  * ipfwadm (2.0-style) support (CONFIG_IP_NF_COMPAT_IPFWADM) [N/y/m/?] n
    - OPTIONAL: If you have an existing IPFWADM ruleset (2.0.x kernels) and
           enable this option, you can continue to use the IPFWADM program and
           the majority of your old ruleset except for the use of any 2.0.x
           kernel-specific modules.   Please note that if this IPFWADM module
           is loaded, ALL IPTABLES modules will be non operational.  This is
           an either/or deal only intended to support legacy rulesets.


    == Non-MASQ options skipped
    ==   (IPv6, khttpd, ATM, IPX, AppleTalk, etc.) --

  * Fast switching (read help!) (CONFIG_NET_FASTROUTE) [N/y/?] n
    - NO: This performance optimization is NOT compatible with IP MASQ and/or
          packet filtering


    == Non-MASQ options skipped
    == (QoS, Telephony, IDE, SCSI, 1394FW, I2O, etc)

      == Don't forget to compile in support for hardware that you might need:
      ==   IDE:    HDs, CDROMs, etc.
      ==   SCSI:   HDs, CDROMs, etc.

[ Network device support ]

  * Network device support (CONFIG_NETDEVICES) [Y/n/?]
    - YES: Enables the Linux Network device sublayer

    == Non-MASQ options skipped
    ==   (Arcnet)


  * Dummy net driver support (CONFIG_DUMMY) [M/n/y/?]
    - YES:  Though OPTIONAL, this option can help when debugging problems

    == Non-MASQ options skipped
    == (EQL, etc..)

    == Don't forget to compile in support for hardware that you might need:
    ==   NICs:   eth, tr, etc.
    ==   MODEMs: ppp (ppp async) and/or slip
    ==   WANs:   T1, T3, ISDN, etc.
    ==   ISDN:   for internal ISDN modems


    == Non-MASQ options skipped
    ==   (Amateur Radio, IrDA, ISDN, USB, etc.)

[ Character devices ]

    == Don't forget to compile in serial port support if you are a modem user
    == Don't forget to compile in mouse support

    == Non-MASQ options skipped
    ==   (I2C, Watchdog cards, Ftape, Video for Linux, etc. )

[ File systems ]

    == Non-MASQ options skipped
    ==   (Quota, ISO9660, NTFS, etc )

  * /proc filesystem support (CONFIG_PROC_FS) [Y/n/?]
    - YES:  Required to dynamically configure the Linux forwarding
            and NATing systems


    == Non-MASQ options skipped
    ==   (Console drivers, Sound, USB, Kernel Hacking)
```

So go ahead and select "exit" and you should be prompted to save your config.

Note: these are just the kernel components you need for IP Masquerade networking support. You will need to select whatever other options needed for your specific setup.

Now compile the kernel

```
    make dep clean bzImage modules modules_install
```

You will then have move over the kernel binary, update your bootloader (LILO, Grub, etc.), and reboot.

Assigning Private Network IP Addresses to the Internal LAN

Since all internal MASQed machines should not have official Internet assigned addressees, there must be a specific and accepted way to allocate addresses to those machines without conflicting with anyone else's Internet address.

There are 3 blocks of IP addresses that can be used in a non-connected or "private" networks:

```
    The Internet Assigned Numbers Authority (IANA) has reserved the following
three blocks of the IP address space for private networks:


                  10.0.0.0        -   10.255.255.255
                  172.16.0.0      -   172.31.255.255
                  192.168.0.0     -   192.168.255.255


    We will refer to the first block as "24-bit block", the second as "20-bit
block", and the third as "16-bit" block".  Note that the first block is
nothing but a single class A network number, while the second block is a set
of 16 continuous class B network numbers, and the third block is a set of 255
continuous class C network numbers.
```

So, if you're using a Class-C network, you should number your TCP/IP enabled machines as 192.168.0.1, 192.168.0.2, 192.168.0.3, .., 192.168.0.x

192.168.0.1 is usually set as the internal gateway or Linux MASQ machine that reaches the external network. Please note that 192.168.0.0 and 192.168.0.255 are the Network and Broadcast address respectively (these addresses are reserved). Avoid using these addresses on your machines or your network will not function properly.

2.2 Configuring `IP Masquerade` on Linux 2.4.x Kernels

Please note that `ipchains` is **no longer the primary firewall configuration tool** for the 2.4.x kernels. The new kernels now use the `iptables` toolkit though the new 2.4.x kernels can still run most old `ipchains` or `ipewadm` rulesets via a compatiblity module. It should be noted that when in this mode, no `iptables` modules can be loaded. It should also be noted that none of the 2.2.x `ipmasq` modules are compatible with 2.4.x kernels.

Ok, the `/etc/rc.d/rc.local` script will load the script called `/etc/rc.d/rc.firewall` once after every reboot. The script will load all required `ipmasq` modules as well as enable the `ipmasq` function. In advanced setups, this same file would contain very secure firewall rulesets as well. Anyway, create the file `/etc/rc.d/rc.firewall-2.4` with the following initial simple ruleset:

```
<rc.firewall-2.4 START>
```

```
#!/bin/sh
#
# rc.firewall-2.4
FWVER=0.75
#
#               Initial SIMPLE IP Masquerade test for 2.4.x kernels
#               using IPTABLES.
#
#               Once IP Masquerading has been tested, with this simple
#               ruleset, it is highly recommended to use a stronger
#               IPTABLES ruleset either given later in this HOWTO or
#               from another reputable resource.
#
#
# Log:
#       0.75 - Added more kernel modules to the comments section
#       0.74 - the ruleset now uses modprobe vs. insmod
#       0.73 - REJECT is not a legal policy yet; back to DROP
#       0.72 - Changed the default block behavior to REJECT not DROP
#       0.71 - Added clarification that PPPoE users need to use
#              "ppp0" instead of "eth0" for their external interface
#       0.70 - Added commented option for IRC nat module
#            - Added additional use of environment variables
#            - Added additional formatting
#       0.63 - Added support for the IRC IPTABLES module
#       0.62 - Fixed a typo on the MASQ enable line that used eth0
#              instead of $EXTIF
#       0.61 - Changed the firewall to use variables for the internal
#              and external interfaces.
#       0.60 - 0.50 had a mistake where the ruleset had a rule to DROP
#              all forwarded packets but it didn't have a rule to ACCEPT
#              any packets to be forwarded either
#            - Load the ip_nat_ftp and ip_conntrack_ftp modules by default
#       0.50 - Initial draft
#
#

echo -e "\n\nLoading simple rc.firewall version $FWVER..\n"


# The location of the iptables and kernel module programs
#
#   If your Linux distribution came with a copy of iptables,
#   most likely all the programs will be located in /sbin.  If
#   you manually compiled iptables, the default location will
#   be in /usr/local/sbin
#
# ** Please use the "whereis iptables" command to figure out
# ** where your copy is and change the path below to reflect
# ** your setup
#
#IPTABLES=/sbin/iptables
IPTABLES=/usr/local/sbin/iptables
DEPMOD=/sbin/depmod
MODPROBE=/sbin/modprobe
```

```
#Setting the EXTERNAL and INTERNAL interfaces for the network
#
#  Each IP Masquerade network needs to have at least one
#  external and one internal network.  The external network
#  is where the natting will occur and the internal network
#  should preferably be addressed with a RFC1918 private address
#  scheme.
#
#  For this example, "eth0" is external and "eth1" is internal"
#
#
#  NOTE:  If this doesnt EXACTLY fit your configuration, you must
#         change the EXTIF or INTIF variables above. For example:
#
#            If you are a PPPoE or analog modem user:
#
#               EXTIF="ppp0"
#
#
#
EXTIF="eth0"
INTIF="eth1"
echo "   External Interface:  $EXTIF"
echo "   Internal Interface:  $INTIF"


#======================================================================
#== No editing beyond this line is required for initial MASQ testing ==


echo -en "   loading modules: "

# Need to verify that all modules have all required dependencies
#
echo "  - Verifying that all kernel modules are ok"
$DEPMOD -a

# With the new IPTABLES code, the core MASQ functionality is now either
# modular or compiled into the kernel.  This HOWTO shows ALL IPTABLES
# options as MODULES.  If your kernel is compiled correctly, there is
# NO need to load the kernel modules manually.
#
#  NOTE: The following items are listed ONLY for informational reasons.
#        There is no reason to manual load these modules unless your
#        kernel is either mis-configured or you intentionally disabled
#        the kernel module autoloader.
#
#
#
#

# Upon the commands of starting up IP Masq on the server, the
# following kernel modules will be automatically loaded:
#
# NOTE:  Only load the IP MASQ modules you need.  All current IP MASQ
#        modules are shown below but are commented out from loading.
# ===============================================================



echo "----------------------------------------------------------------------"

#Load the main body of the IPTABLES module - "iptable"
#  - Loaded automatically when the "iptables" command is invoked
#
#  - Loaded manually to clean up kernel auto-loading timing issues
#
echo -en "ip_tables, "
$MODPROBE ip_tables
```

```
#Load the IPTABLES filtering module - "iptable_filter"
#  - Loaded automatically when filter policies are activated


#Load the stateful connection tracking framework - "ip_conntrack"
#
# The conntrack  module in itself does nothing without other specific
# conntrack modules being loaded afterwards such as the "ip_conntrack_ftp"
# module
#
#  - This module is loaded automatically when MASQ functionality is
#    enabled
#
#  - Loaded manually to clean up kernel auto-loading timing issues
#
echo -en "ip_conntrack, "
$MODPROBE ip_conntrack


#Load the FTP tracking mechanism for full FTP tracking
#
# Enabled by default -- insert a "#" on the next line to deactivate
#
echo -en "ip_conntrack_ftp, "
$MODPROBE ip_conntrack_ftp


#Load the IRC tracking mechanism for full IRC tracking
#
# Enabled by default -- insert a "#" on the next line to deactivate
#
echo -en "ip_conntrack_irc, "
$MODPROBE ip_conntrack_irc


#Load the general IPTABLES NAT code - "iptable_nat"
#  - Loaded automatically when MASQ functionality is turned on
#
#  - Loaded manually to clean up kernel auto-loading timing issues
#
echo -en "iptable_nat, "
$MODPROBE iptable_nat


#Loads the FTP NAT functionality into the core IPTABLES code
# Required to support non-PASV FTP.
#
# Enabled by default -- insert a "#" on the next line to deactivate
#
echo -en "ip_nat_ftp, "
$MODPROBE ip_nat_ftp


#Loads the IRC NAT functionality into the core IPTABLES code
# Required to support NAT of IRC DCC requests
#
# Disabled by default -- remove the "#" on the next line to activate
#
#echo -e "ip_nat_irc"
#$MODPROBE ip_nat_irc

echo "----------------------------------------------------------------------"

# Just to be complete, here is a partial list of some of the other
# IPTABLES kernel modules and their function.  Please note that most
# of these modules (the ipt ones) are automatically loaded by the
# master kernel module for proper operation and don't need to be
# manually loaded.
# --------------------------------------------------------------------
#
```

```
#    ip_nat_snmp_basic - this module allows for proper NATing of some
#                        SNMP traffic
#
#    iptable_mangle    - this target allows for packets to be
#                        manipulated for things like the TCPMSS
#                        option, etc.
#
# --
#
#    ipt_mark       - this target marks a given packet for future action.
#                     This automatically loads the ipt_MARK module
#
#    ipt_tcpmss     - this target allows to manipulate the TCP MSS
#                     option for braindead remote firewalls.
#                     This automatically loads the ipt_TCPMSS module
#
#    ipt_limit      - this target allows for packets to be limited to
#                     to many hits per sec/min/hr
#
#    ipt_multiport  - this match allows for targets within a range
#                     of port numbers vs. listing each port individually
#
#    ipt_state      - this match allows to catch packets with various
#                     IP and TCP flags set/unset
#
#    ipt_unclean    - this match allows to catch packets that have invalid
#                     IP/TCP flags set
#
#    iptable_filter - this module allows for packets to be DROPped,
#                     REJECTed, or LOGged.  This module automatically
#                     loads the following modules:
#
#                     ipt_LOG - this target allows for packets to be
#                               logged
#
#                     ipt_REJECT - this target DROPs the packet and returns
#                                  a configurable ICMP packet back to the
#                                  sender.
#
#

echo -e "   Done loading modules.\n"



#CRITICAL:  Enable IP forwarding since it is disabled by default since
#
#           Redhat Users:  you may try changing the options in
#                          /etc/sysconfig/network from:
#
#                       FORWARD_IPV4=false
#                             to
#                       FORWARD_IPV4=true
#
echo "   Enabling forwarding.."
echo "1" > /proc/sys/net/ipv4/ip_forward



# Dynamic IP users:
#
#   If you get your IP address dynamically from SLIP, PPP, or DHCP,
#   enable this following option.  This enables dynamic-address hacking
#   which makes the life with Diald and similar programs much easier.
#
echo "   Enabling DynamicAddr.."
echo "1" > /proc/sys/net/ipv4/ip_dynaddr
```

```
# Enable simple IP forwarding and Masquerading
#
#  NOTE:  In IPTABLES speak, IP Masquerading is a form of SourceNAT or SNAT.
#
#  NOTE #2:  The following is an example for an internal LAN address in the
#            192.168.0.x network with a 255.255.255.0 or a "24" bit subnet mask
#            connecting to the Internet on external interface "eth0".  This
#            example will MASQ internal traffic out to the Internet but not
#            allow non-initiated traffic into your internal network.
#
#
#         ** Please change the above network numbers, subnet mask, and your
#         *** Internet connection interface name to match your setup
#


#Clearing any previous configuration
#
#  Unless specified, the defaults for INPUT and OUTPUT is ACCEPT
#    The default for FORWARD is DROP (REJECT is not a valid policy)
#
echo "   Clearing any existing rules and setting default policy.."
$IPTABLES -P INPUT ACCEPT
$IPTABLES -F INPUT
$IPTABLES -P OUTPUT ACCEPT
$IPTABLES -F OUTPUT
$IPTABLES -P FORWARD DROP
$IPTABLES -F FORWARD
$IPTABLES -t nat -F

echo "   FWD: Allow all connections OUT and only existing and related ones IN"
$IPTABLES -A FORWARD -i $EXTIF -o $INTIF -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -i $INTIF -o $EXTIF -j ACCEPT
$IPTABLES -A FORWARD -j LOG

echo "   Enabling SNAT (MASQUERADE) functionality on $EXTIF"
$IPTABLES -t nat -A POSTROUTING -o $EXTIF -j MASQUERADE

echo -e "\nrc.firewall-2.4 v$FWVER done.\n"
```

```
<rc.firewall-2.4 STOP>
```

Once you are finished with editing the `/etc/rc.d/rc.firewall` ruleset, make it executable by typing in `chmod 700 /etc/rc.d/rc.firewall-2.4`

Now that the firewall ruleset is ready, you need to let it run after every reboot. You could either do this by running it by hand everytime or add it to the boot scripts.

There are two ways to load things in Slackware: `/etc/rc.d/rc.local` or editing the `/etc/rc.d/rc.inet2 file`. The first method is the easiest. All you have to do is add the line:

```
echo "Loading the rc.firewall ruleset.."
/etc/rc.d/rc.firewall-2.4
```

Notes on how users might want to change the above firewall ruleset:

You could also have IP Masquerading enabled on a per machine basis instead of the above method, which is enabling an entire TCP/IP network. For example, say if you wanted only the 10.0.0.2 and 10.0.0.8 hosts to have access to the Internet and not any of the other internal machines. We would change the in the "Enable simple IP forwarding and Masquerading" section (shown above) of the `/etc/rc.d/rc.firewall` ruleset.

```
#!/bin/sh
#
# Partial 2.4.x config to enable simple IP forwarding and Masquerading
# v0.61
#
#  NOTE:  The following is an example to allow only IP Masquerading for the
#         10.0.0.2 and 10.0.0.8 machines with a 255.255.255.0 or a
#         "/24" subnet mask connecting to the Internet on interface eth0.
#
#         ** Please change the network number, subnet mask, and the Internet
#         ** connection interface name to match your internal LAN setup
#
echo "  - Setting the default FORWARD policy to DROP"
$IPTABLES -P FORWARD DROP

echo "  - Enabling SNAT (IPMASQ) functionality on $EXTIF"
$IPTABLES -t nat -A POSTROUTING -o $EXTIF -s 192.168.0.2/32 -j MASQUERADE
$IPTABLES -t nat -A POSTROUTING -o $EXTIF -s 192.168.0.8/32 -j MASQUERADE

echo "  - Setting the FORWARD policy to 'DROP' all incoming / unrelated traffic"
$IPTABLES -A INPUT -i $EXTIF -m state --state NEW,INVALID -j DROP
$IPTABLES -A FORWARD -i $EXTIF -m state –state NEW,INVALID -j DROP
```

It appears that a common mistake with new IP Masq users is to make the first command simply the following:

```
iptables -t nat -A POSTROUTING -j MASQUERADE
```

Do no make your default policy **masquerading**. Otherwise, someone can manipulate their routing tables to tunnel straight back through your gateway, using it to masquerade their own identity.

Again, you can add these lines to the `/etc/rc.d/rc.firewall` file, one of the other `rc` files you prefer, or do it manually every time you need IP Masquerade.

3 Parallel virtual machine PVM

What is PVM?

PVM is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource, or a "parallel virtual machine". The individual computers may be shared- or local-memory multiprocessors, vector supercomputers, specialized graphics engines, or scalar workstations and PCs, that may be interconnected by a variety of networks, such as Ethernet or FDDI.

The PVM software must be specifically installed on every machine that is to be used in your "virtual machine". There is no "automatic" installation of executables onto remote machines in PVM, although simply copying the `pvm3/lib` and `pvm3/bin` directories to another *similar* machine (and setting `$PVM_ROOT` and `$PVM_ARCH` - see below) is sufficient for running PVM programs. Compiling or building PVM programs requires the full PVM installation.

User programs written in `C`, `C++` or `Fortran` can access PVM through provided library routines.

3.1 Building and installing

Downloading PVM sources

The latest version of PVM and other information may be obtained from the web at http://www.netlib.org/pvm3/. On this page, you'll be able to find the various versions of `PVM` available. We'll be using `pvm3.4.5.`

Unpacking

This distribution contains source code, some simple examples, and run-time support for PVM version 3. The files in the source distribution unpack into a "`pvm3`" directory. The `pvm3` directory can reside in either a private or shared disk area. Installations for multiple machine architectures can coexist in the same shared filesystem because compiled files are placed in different subdirectories named for each architecture (`$PVM_ARCH`).

Setting Up PVM

To build the full PVM software, you will need to have a "make" utility, a `C` or `C++` compiler, and a Fortran compiler installed on your system.

Before building or running PVM, you must set the environment variable "`PVM_ROOT`" to the path where PVM resides, i.e. the path of the directory with this "Readme" file. This can be in a private area, for example `$HOME/pvm3`, or a public one, such as `/usr/local/pvm3`.

PVM uses two environment variables when starting and running. Each PVM user needs to set these two variables to use PVM. The first variable is `PVM_ROOT`, which is set to the location of the installed `pvm3` directory. The second variable is `PVM_ARCH`, which tells PVM the architecture of this host and thus what executables to pick from the `PVM_ROOT` directory. The easiest method is to set these two variables in your `.profile` file. Here is an example for setting `PVM_ROOT`:

```
export PVM_ROOT=$HOME/pvm3
```

The recommended method to set PVM_ARCH is to append the file PVM_ROOT/lib/bashrc.stub onto your .profile file. The stub should be placed after PATH and PVM_ROOT are defined. This stub automatically determines the PVM_ARCH for this host and is particularly useful when the user shares a common file system (such as NFS) across several different architectures.

The PVM source comes with directories and makefiles for Linux and most architectures you are likely to have. Building for each architecture type is done automatically by logging on to a host, going into the PVM_ROOT directory, and typing make. The makefile will automatically determine which architecture it is being executed on, create appropriate subdirectories, and build pvmd3, libpvm3.a, and libfpvm3.a, pvmgs, and libgpvm3.a. It places all these files in PVM_ROOT/lib/PVM_ARCH with the exception of pvmgs, which is placed in PVM_ROOT/bin/PVM_ARCH.

Starting PVM

Before we go over the steps to compile and run parallel PVM programs, you should be sure you can start up PVM and configure a virtual machine. On any host on which PVM has been installed you can type

```
pvm
```

and you should get back a PVM console prompt signifying that PVM is now running on this host. You can add hosts to your virtual machine by typing at the console prompt

```
pvm> add hostname
```

You also can delete hosts (except the one you are on) from your virtual machine by typing

```
pvm> delete hostname
```

If you get the message "Can't Start pvmd", PVM will run autodiagnostics and report the reason found.

To see what the present virtual machine looks like, you can type

```
pvm> conf
```

To see what PVM tasks are running on the virtual machine, you type

```
pvm> ps –a
```

Of course, you don't have any tasks running yet. If you type "`quit`" at the console prompt, the console will quit, but your virtual machine and tasks will continue to run. At any command prompt on any host in the virtual machine you can type

```
pvm
```

and you will get the message "pvm already running" and the console prompt. When you are finished with the virtual machine you should type

```
pvm> halt
```

This command kills any PVM tasks, shuts down the virtual machine, and exits the console. This is the recommended method to stop PVM because it makes sure that the virtual machine shuts down cleanly.

If you don't wish to type in a bunch of hostnames each time, there is a hostfile option. You can list the hostnames in a file one per line and then type

```
pvm hostfile
```

PVM will then add all the listed hosts simultaneously before the console prompt appears. Several options can be specified on a per host basis in the hostfile; see PVM documentation if you wish to customize your virtual machine for a particular application or environment.

PVM may also be started in other ways. The functions of the console and a performance monitor have been combined in a graphical user interface called XPVM, which is available from the PVM web site. If XPVM has been installed at your site, then it can be used to start PVM.

XPVM

XPVM provides a graphical interface to the PVM console commands, along with animated views to monitor the execution of PVM programs. These views provide information about the interactions among tasks in a parallel PVM program to assist in debugging and performance tuning.

XPVM is written in `C` using allowing extensibility to include a variety of views. XPVM is available via Netlib in the “`pvm3/xpvm`” sub-directory via http://www.netlib.org/pvm3/xpvm/index.html. The latest version of XPVM and other information may be obtained from the web at http://www.netlib.org/utk/icl/xpvm/xpvm.html.

To start PVM with this interface type:

```
xpvm
```

The menu button labeled "hosts" will pull down a list of hosts you can add. By clicking on a hostname it is added and an icon of the machine appears in an animation of the virtual machine. A host is deleted if you click on a hostname that is already in the virtual machine. On startup XPVM reads the file $HOME/.xpvm_hosts, which is a list of hosts to display in this menu. Hosts without leading "&" are added all at once at start up. The quit and halt buttons work just like the PVM console. If you quit XPVM and then restart it, XPVM will automatically display what the running virtual machine looks like. If there are errors they should appear in the window where you started XPVM.

3.2 Running PVM Programs

In this section is described how to compile and run the example programs supplied with the PVM software. These example programs make useful templates on which to base your own PVM programs.

The first step is to copy the example programs into your own area:

```
cp -r $PVM_ROOT/examples $HOME/pvm3/examples
cd $HOME/pvm3/examples
```

The examples directory contains a Makefile.aimk and Readme file that describe how to build the examples. PVM supplies an architecture independent make, aimk that automatically determines PVM_ARCH and links any operating system specific libraries to your application. aimk was automatically added to your $PATH when you placed the bashrc.stub in your .bashrc file. Using aimk allows you to leave the source code and makefile unchanged as you compile across different architectures.

The master/worker programming model is the most popular model used in cluster computing. To compile the master/slave C example, type

```
aimk master slave
```

If you prefer to work with Fortran, compile the Fortran version with

```
aimk fmaster fslave
```

Depending on the location of PVM_ROOT, the INCLUDE statement at the top of the Fortran examples may need to be changed. If PVM_ROOT is not HOME/pvm3, then change INCLUDE to point to $PVM_ROOT/include/fpvm3.h. Note that PVM_ROOT is not expanded inside the Fortran, so you must insert the actual path.

The makefile moves the executables to $HOME/pvm3/bin/PVM_ARCH, which is the default location PVM will look for them on all hosts. If your file system is not common across all your PVM hosts, then you will have to build or copy (depending on the architecture) these executables on all your PVM hosts.

From one window start up PVM and configure some hosts. These examples are designed to run on any number of hosts, including one. In another window, `cd` to the location of the PVM executables and type

```
master
```

The program will ask about the number of tasks. This number does not have to match the number of hosts in these examples. Try several combinations. The first example illustrates the ability to run a PVM program from a prompt on any host in the virtual machine. This is how you would run a serial `a.out` program on a workstation. The next example, which is also a master/slave model called `hitc`, shows how to spawn PVM jobs from the PVM console and also from XPVM.

The model `hitc` illustrates dynamic load balancing using the pool of tasks paradigm. In this paradigm, the master program manages a large queue of tasks, always sending idle slave programs more work to do until the queue is empty. This paradigm is effective in situations where the hosts have very different computational powers because the least-loaded or more powerful hosts do more of the work and all the hosts stay busy until the end of the problem. To compile hitc, type

```
aimk hitc hitc_slave
```

Since `hitc` does not require any user input, it can be spawned directly from the PVM console. Start the PVM console, and add a few hosts. At the PVM console prompt, type

```
pvm> spawn -> hitc
```

The "`->`" spawn option causes all the print statements in `hitc` and in the slaves to appear in the console window. This can be a useful feature when debugging your first few PVM programs. You may wish to experiment with this option by placing print statements in `hitc.f` and `hitc_slave.f` and recompiling.

To get an idea of XPVM's real-time animation capabilities, you again can use `hitc`. Start up XPVM, and build a virtual machine with a few hosts. Click on the "tasks" button and select "spawn" from the menu. Type "hitc" where XPVM asks for the command, and click on "start". You will see the host icons light up as the machines become busy. You will see the `hitc_slave` tasks get spawned and see all the messages that travel between the tasks in the *Space Time* display. Several other views are selectable from the XPVM "views" menu. The "task output" view is equivalent to the "->" option in the PVM console. It causes the standard output from all tasks to appear in the window that pops up.

Programs that are spawned from XPVM (and the PVM console) are subject to one restriction: they must not contain any interactive input, such as asking for how many slaves to start up or how big a problem to solve. This type of information can be read from a file or put on the command-line as arguments, but there is nothing in place to get user input from the keyboard to a potentially remote task.

4 Message Passing Interface MPI

Introduction

MPI (Message-Passing Interface) is a standard specification for message-passing libraries. `MPICH` is a freely available implementation of the MPI standard that runs on a wide variety of systems. `MPICH` contains, along with the MPI library itself, a programming environment for working with MPI programs. The programming environment includes a portable startup mechanism, several profiling libraries for studying the performance of MPI programs, and an X interface to all of the tools. This document describes how to obtain, install and use the `MPICH` implementation of MPI. The details of the `MPICH` implementation are described in [1]; related papers include [2] and [3]. This document describes version `mpich-1.2.6`.

Downloading `MPICH`

The first step is to download `MPICH` and install any necessary patches. The easiest way to get `MPICH` is to use the web page www.mcs.anl.gov/mpi/mpich/download.html. Get the file '`mpich.tar.gz`'. Unpack the '`mpich.tar.gz`' file into a build directory. We recommend using a locally mounted partition rather than an NFS (network file system) partition. For example, on many systems, '`/usr/local/`'.

```
mv mpich.tar.gz /usr/local
cd /usr/local
tar –zxvf mpich.tar.gz
```

If your tar does not accept the z option, use

```
gunzip –c mpich.tar.gz | tar zxovf -
```

Apply any patches. Check the web page www.mcs.anl.gov/mpi/mpich/buglist-tbl.html for any patches that need to be applied. Normally, versions of `MPICH` that already have these important patches applied are available; they are indicated by a fourth number in the release name (e.g., 1.2.2.3). But in some cases, a patch is made available early. The patch page has instructions on applying the patches.

Now you are ready to build `MPICH`.

4.1 Configuring, Making and Installing

Before you can use `MPICH`, you must configure and make it. The configuration process analyzes your system and determines the correct options and settings; it also creates the '`Makefile`'s that are used to make `MPICH`.

1. Decide where you want to install `MPICH`. This step is not strictly necessary (except in the case of the `ch_p4mpd` device that is the most general and supports SMP nodes, MPMD programs, and heterogeneous collections of systems. The `ch_p4mpd` device (so far) supports only homogenous clusters of uniprocessors, but provides for far faster and more scalable startup); however, installing `MPICH` (which can be done without any privileges in a user directory) both makes it easier to manage updates to `MPICH` and allows you to reduce the amount disk space that `MPICH` takes up, since the installed version contains only the libraries, header files, documentation, and supporting programs. We recommend an install path that contains the version number of `MPICH`. For example, if you are installing `MPICH` for others to use, and you have the required access rights, you could choose '`/usr/local/mpich-1.2.6/`'. If you are installing it just for your own use, you could use something like '`/home/me/software/mpich-1.2.`6'.

2. Invoke `configure` with the appropriate prefix and `make`:

```
./configure PREFIX=/usr/local/mpich-1.2.6
make
```

This may take a while, depending on the load on your system and on your file server, it may take anywhere from a few minutes to an hour or more.

3. (Optional) On workstation networks, or to run on a single workstation, edit the file '`mpich/util/machines/machines.xxx`' (where `xxx` is `MPICH`'s name for your machine's architecture; you will recognize it) to reflect your local host names for your workstations. If you want to, you can skip this step because by default, five copies of the machine you have built `MPICH` on will be there to begin with. On parallel machines, this step is not needed. See the 'README' file in the '`mpich/util/machines`' directory for a description of the format.

Installing `MPICH` for Others to Use

This step is optional. However, if you are installing `MPICH`, you should make sure that you specified the directory into which `MPICH` is to be installed when you configure `MPICH` by using the `-prefix` option. For example, if you plan to install `MPICH` into `'/usr/local/mpich-1.2.6/'`, then you should configure with the option `-prefix='/usr/local/mpich-1.2.6/'`. If there is any possibility at all that you will build `MPICH` for several systems and/or devices, you should include that information in the prefix. For example, by using

```
-prefix='/usr/local/mpich-1.2.6/linux/ch_p4'
```

you can later add

```
-prefix='/usr/local/mpich-1.2.6/linux/ch_p4smp'
```

for a version that is built with the configure option -comm=shared (suitable for clusters of symmetric multiprocessors, hence the "smp" in the directory name). Once you have tested all parts of the MPI distribution (including the tools, particularly upshot and/or nupshot), you may install MPICH into a publicly available directory, and disseminate information for other users, so that everyone can use the shared installation. To install the libraries and include files in a publicly available place, change to the top-level MPICH directory, and do

```
make install
```

The man pages will have been copied with the installation, so you might want to add '/usr/local/mpich-1.2.6/man' to the default system MANPATH. The man pages can be conveniently browsed with the mpiman command, found in the mpich/bin directory. It is possible to specify the directory into which MPICH should be installed after building MPICH by setting the value of PREFIX when executing the installation step:

```
make install PREFIX=/usr/local/mpich-1.2.6
```

Installation will consist of an 'include', 'lib', 'bin', 'sbin', 'www', and 'man' directories and a small 'examples' directory. Should you wish to remove the installation, you can run the script sbin/mpiuninstall.

However, some features, particularly the ability of Totalview to show MPICH message queues, will work only if MPICH is configured with the prefix set to the installation directory.

A good way to handle multiple releases of MPICH is to install them into directories whose names include the version number and then set a link from mpi to that directory. For example, if the current version is 1.2.6, the installation commands to install into '/usr/local' are

```
make install PREFIX=/usr/local/mpi-1.2.6
rm /usr/local/mpi
ln -s /usr/local/mpi-1.2.6 /usr/local/mpi
```

The script 'bin/mpiinstall' provides more control over the installation of MPICH (in fact, make install just runs this script). For example, you can change the protection of the files when they are installed with the options -mode=nnnn (for regular files) and -xmode=nnnn (for executables and directories). You can set the directory into which the man pages will be placed with -manpath=<path>. The option -help shows the full set of options for mpiinstall.

You can test the installation by using configure in 'mpich/examples/test'. For example, if you have installed MPICH into '/usr/local/mpich-1.2.6/' for architecture Linux and device ch_p4, execute

```
cd examples/test
./configure -mpichpath=/usr/local/mpich-1.2.6/linux/ch_p4/bin
make testing
```

The test codes in the '`mpich/examples/test`' directory may be used with *any* implementation of MPI, not just the `MPICH` implementation.

Running examples (Optional)

1. Build and run a simple test program:

```
cd examples/basic
make cpi
../../bin/mpirun -np 4 cpi
```

At this point you have run an MPI program on four processors of your system.

2. At this point you can announce to your users how to compile and run MPI programs, using the installation you have just built in '`/usr/local/mpich-1.2.6/`' (or wherever you have installed it). They can also copy the '`Makefile`' in '`/usr/local/mpich-1.2.6/examples`' and adapt it for their own use.

Sample MPI programs

The `MPICH` distribution contains a variety of sample programs, which are located in the `MPICH` source tree. Most of these will work with any MPI implementation, not just `MPICH`.

`examples/basic` contains a few short programs in `Fortran`, `C`, and `C++` for testing the simplest features of MPI.

`examples/test` contains multiple test directories for the various parts of MPI. Enter "`make testing`" in this directory to run our suite of function tests.

`examples/perftest` Performance benchmarking programs. See the script `runmpptest` for information on how to run the benchmarks. These are relatively sophisticated.

`mpe/contrib/mandel` A Mandelbrot program that uses the MPE graphics package that comes with `mpich`. It should work with any other MPI implementation as well, but we have not tested it. This is a good demo program if you have a fast X server and not too many processes.

`mpe/contrib/mastermind` A program for solving the Mastermind puzzle in parallel. It can use graphics (gmm) or not (mm).

Tutorial material on MPI can also be found at www.mcs.anl.gov/mpi.

4.2 Programming Tools

The `MPICH` implementation comes with a variety of tools for building, running, debugging, and analyzing MPI programs.

Compiling, linking, and running programs

The `MPICH` implementation provides four commands for compiling and linking `C` (`mpicc`), `C++` (`mpiCC`), `Fortran 77` (`mpif77`), and `Fortran 90` (`mpif90`) programs.

Use these commands just like the usual `C`, `Fortran 77`, `C++`, or `Fortran` compilers. For example,

```
mpicc –c file.c
mpif77 -c file.f
mpiCC -c file.cc
mpif90 -c file.f90
```

and

```
mpicc -o file file.o
mpif77 -o file file.o
mpicxx -o file file.o
mpif90 -o file file.o
```

Commands for the linker may include additional libraries. For example, to use routines from the C math library, use

```
mpicc -o file file.o -lm
```

Note that while the suffixes `.c` for `C` programs and `.f` for `Fortran-77` programs are standard, there is no consensus for the suffixes for `C++` and `Fortran-90` programs. The ones shown here are accepted by many but not all systems. `MPICH` tries to determine the accepted suffixes, but may not always be able to.

If you want to see the commands that would be used without actually running them, add the command line argument `-show`.

In addition, the following special options are supported for accessing some of the features of the `MPE` environment for monitoring MPI calls from within an application:

`-mpilog` Build version that generates `MPE` log files.
`-mpitrace` Build version that generates traces.
`-mpianim` Build version that generates real-time animation.

Running programs with mpirun

To run an MPI program, use the `mpirun` command, which is located in '`/usr/local/mpich-1.2.6/bin`'. For almost all systems, you can use the command

```
mpirun -np 4 a.out
```

to run the program 'a.out' on four processors. The command `mpirun -help` gives you a complete list of options. On exit, `mpirun` returns the status of one of the processes, usually the process with rank zero in `MPI_COMM_WORLD`.

Workstation clusters and the ch_p4 device

Most massively parallel processors (MPPs) provide a way to start a program on a requested number of processors; `mpirun` makes use of the appropriate command whenever possible. In contrast, workstation clusters require that each process in a parallel job be started individually, though programs to help start these processes exist. Because workstation clusters are not already organized as an MPP, additional information is required to make use of them. MPICH should be installed with a list of participating workstations in the file '`machines.<arch>`' in the directory '`/usr/local/mpich/share`'. This file is used by `mpirun` to choose processors to run on.

Checking your machines list

Use the script '`tstmachines`' in '`/usr/local/mpich/sbin`' to ensure that you can use all of the machines that you have listed. This script performs an `rsh` and a short directory listing; this tests that you both have access to the node and that a program in the current directory is visible on the remote node. If there are any problems, they will be listed. These problems *must* be fixed before proceeding.

The only argument to `tstmachines` is the name of the architecture; this is the same name as the extension on the machines file. For example,

```
/usr/local/mpich/tstmachines LINUX
```

that a program in the current directory can be executed by all of the machines in the LINUX machines list. This program is silent if all is well; if you want to see what it is doing, use the `-v` (for verbose) argument:

```
/usr/local/mpich/tstmachines –v LINUX
```

The output from this command might look like in our cluster

```
Trying true on node01 ...
...
Trying true on node08 ...
Trying ls on node01 ...
...
Trying ls on node08 ...
Trying user program on node01 ...
...
Trying user program on node08 ...
```

If `tstmachines` finds a problem, it will suggest possible reasons and solutions.

Changing the Remote Shell Program

You can change the remote shell command that the `ch_p4 device` uses to start the remote processes with the environment variable `P4_RSHCOMMAND`. For example, if the default remote shell program is `rsh` but you wish to use the secure shell `ssh`, you can do

```
export P4_RSHCOMMAND ssh
mpirun -np 4 a.out
```

This only works for different remote shell commands that accept the same command line arguments. If you are having trouble using the remote shell commands, consider using either the secure shell or the `ch_p4mpd` device.

The P4 Procgroup File

For even more control over how jobs get started, we need to look at how `mpirun` starts a parallel program on a workstation cluster. Each time `mpirun` runs, it constructs and uses a new file of machine names for just that run, using the machines file as input. (The new file is called PIyyyy, where yyyy is the process identifier). If you specify `-keep_pg` on your `mpirun` invocation, you can use this information to see where `mpirun` ran your last few jobs.

You can construct this file yourself and specify it as an argument to `mpirun`. To do this for `ch_p4`, use

```
mpirun -p4pg pgfile myprog
```

where `pfile` is the name of the file. The file format is defined below.

This is necessary when you want closer control over the hosts you run on, or when `mpirun` cannot construct it automatically. Such is the case when

-- You want to run different executables on different hosts (your program is not SPMD).

-- You want to run on a network of shared-memory multiprocessors *and* need to specify the number of processes that will share memory on each machine.

The format of a `ch_p4 procgroup` file is a set of lines of the form

```
<hostname> <#procs> <progname> [<login>]
```

An example of such a file, where the command is being issued from host node01, might be

```
node01 0 /users/mpi/myprog
node02 1 /users/mpi/myprog
dragon 1 /home/user/myprog userlogin
```

The above file specifies three processes, one on each of two linux and one on another workstation where the user’s account name is different. Note the 0 is in the first line. It is there to indicate that no *other* processes are to be started on host node01 than the one started by the user by his command.

You might want to run all the processes on your own machine, as a test. You can do this by repeating its name in the file:

```
node01 0 /users/mpi/myprog
node01 1 /users/mpi/myprog
node01 1 /users/mpi/myprog
```

This will run three processes on node01, communicating via sockets.

4.3 Some details

Setting up `rsh`

If you are using `rsh` with the `ch_p4` device, you may need to set up your machine to allow the use of `rsh`. You should do this only if you are a system administrator and understand what you are doing *or* you are using an isolated network. For example, if you are using Linux on a collection of computers at your home or at work, and these machines are *not* connected to a larger network, you should follow these directrions. If any of the machines are connected to another network, talk to the administrator of the network about the policy for using `rsh`.

The following explains how to setup a single machine so that it can use rsh to itself to start processes. To setup rsh, you need to ensure that you have a file '/etc/hosts.equiv' that contains at least

```
localhost
your_machine_name
```

where your_machine_name is the name that you've given your machine in '/etc/hosts'.

You may also need to ensure that the files '/etc/hosts.allow' and '/etc/hosts.deny' are empty.

When using a machine that is not networked, for example, a laptop while travelling, you may need to change your network settings. Under some versions of Linux, use the netcfg or netconfig and set Hostname to localhost and Domain to localdomain.

Fortran Compilers

MPICH provides support for both Fortran 77 and Fortran 90. Because MPICH is implemented in C, using MPICH from Fortran can sometimes require special options. This section discusses some of the issues. Note that configure tries to determine the options needed to support Fortran. You need the information in this section only if you have problems.

Fortran 77 and Fortran 90

Selecting Fortran 90 with Fortran 77 should be done only when the two compilers are compatible, supporting the same datatypes and calling conventions. In particular, if the Fortran 90 compiler supports an 8-byte integer type, the Fortran 77 compiler must support integer*8 (this is needed by the MPI-IO routines for the value of MPI_OFFSET_KIND). In addition, both compilers must support the same functions for accessing the command line, and the code for those commands must reside in the same library. If the two Fortran compilers are not compatible, you should either select the Fortran 90 compiler as both the Fortran 77 and Fortran 90 compiler (relying on the upward compatibility of Fortran), or build two separate configurations of MPICH. For example,

```
export FC f90
export F90 f90
configure
```

will use f90 for both Fortran 77 and Fortran 90 programs. In many systems, this will work well. If there are reasons to have separate Fortran 90 and Fortran 77 builds, then execute the following commands (where MPICH is to be installed into the directory '/usr/local'):

```
export FC f77
configure --disable-f90 PREFIX=/usr/local/mpich-1.2.6/f77-nof90
make
make install
export FC f90
export F90 f90
configure PREFIX=/usr/local/mpich-1.2.6/f90
make
make install
```

This sequence of commands will build and install two versions of MPICH. An alternative approach that installs only a single version of MPICH is described later.

Fortran 90 Modules

If configure finds a Fortran 90 compiler, by default MPICH will try to create a Fortran 90 module for MPI. In fact, it will create two versions of an mpi module: one that includes only the MPI routines that do not take "choice" arguments and one that does include choice argument. A choice argument is an argument that can take any datatype; typically, these are the buffers in MPI communication routines such as MPI_Send and MPI_Recv.

The two different modules can be accessed with the -nochoice and -choice option to mpif90. The choice version of the module supports a limited set of datatypes (numeric scalars and numeric one- and two-dimensional arrays). This is an experimental feature.

The reason for having two versions of the MPI module is that it is very difficult to provide a completely correct module that includes all of the functions with choice arguments. As it is, on many systems, the size of the Fortran 90 module to handle the routines with choice arguments will be larger than the entire C version of the MPI library. If you are uninterested in the Fortran 90 MPI module, or you wish to keep the installed version of MPICH small, you can turn off the creation of the Fortran 90 MPI module with the configure option

```
    --disable-f90modules
```

Configuring for Multiple Fortran Compilers

In some environments, there are several different Fortran compilers, all of which define Fortran datatypes of the same size, and which can be used with the same C libraries. These compilers may make different choices for Fortran name mappings (e.g., the external format of the names given to the linker) and use different approaches to access the command line. This section describes how to configure MPICH to support multiple Fortran compilers.

However, if any of these steps fails, the best approach is to build a separate MPICH installation for each Fortran compiler.

The first step is to configure `MPICH` with the `--with-flibname` option. For example, if one of the compilers is `g77`, use

```
export FC g77
./configure --with-flibname=mpich-g77 ... other options ...
```

After you build, test, *and install* this version of `MPICH`, you can configure support for additional Fortran compilers as follows:
- change directory to ‘src/fortran’
- execute

```
export FC pgf77
./configure --with-mpichconfig --with-flibname=mpich-pgf77
make
make install-alt
```

To use a particular `Fortran` compiler, either select it on the `mpif77` command line with the `-config=name` option (e.g., `-config=pgf77`) or select a particular `mpif77` command (e.g., `mpif77-pgf77`).

Using Shared Libraries

Shared libraries can help reduce the size of an executable. This is particularly valuable on clusters of workstations, where the executable must normally be copied over a network to each machine that is to execute the parallel program. However, there are some practical problems in using shared libraries; this section discusses some of them and how to solve most of those problems. **Currently, shared libraries are not supported from `C++`.**

In order to build shared libraries for `MPICH`, you must have configured and built `MPICH` with the `--enable-sharedlib` option. Because each Unix system and in fact each compiler uses a different and often incompatible set of options for creating shared objects and libraries, `MPICH` may not be able to determine the correct options. Currently, `MPICH` understands Solaris, GNU `gcc` (on most platforms, including Linux and Solaris), and IRIX.

Once the shared libraries are built, you must tell the `MPICH` compilation and linking commands to use shared libraries (the reason that shared libraries are not the default will become clear below). You can do this either with the command line option `-shlib` or by setting the environment variable `MPICH_USE_SHLIB` to yes. For example,

```
   mpicc -o cpi -shlib cpi.c
or
   export MPICH_USE_SHLIB yes
   mpicc -o cpi cpi.c
```

Using the environment variable `MPICH_USE_SHLIB` allows you to control whether shared libraries are used without changing the compilation commands; this can be very useful for projects that use makefiles.

Running a program built with shared libraries can be tricky. If you have trouble, particular if programs will either not start or start and issue error messages about missing libraries, see documentation on page www.mcs.anl.gov/mpi/mpich/docs.html.

References:

[1] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A highperformance, portable implementation of the MPI Message-Passing Interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[2] William Gropp and Ewing Lusk. A high-performance MPI implementation on a sharedmemory vector supercomputer. *Parallel Computing*, 22(11):1513–1526, January 1997.

[3] William Gropp and Ewing Lusk. Sowing MPICH: A case study in the dissemination of a portable environment for parallel scientific computing. *IJSA*, 11(2):103–114, Summer 1997.

[4] James Boyle, Ralph Butler, Terrence Disz, Barnett Glickfeld, Ewing Lusk, Ross Overbeek, James Patterson, and Rick Stevens. *Portable Programs for Parallel Processors*. Holt, Rinehart, and Winston, New York, NY, 1987.

[5] Ralph Butler and Ewing Lusk. User's guide to the p4 parallel programming system. Technical Report ANL-92/17, Argonne National Laboratory, Argonne, IL, 1992.

[6] Anthony Chan, William Gropp, and Ewing Lusk. User's guide for mpe extensions for mpi programs. Technical Report ANL-98/xx, Argonne National Laboratory, 1998. The updated version is at ftp://ftp.mcs.anl.gov/pub/mpi/mpeman.ps.

[7] James Cownie and William Gropp. A standard interface for debugger access to message queue information in MPI. In Jack Dongarra, Emilio Luque, and Tom`as Margalef, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1697 of *Lecture Notes in Computer Science*, pages 51–58. Springer Verlag, 1999.

[8] Message Passing Interface Forum. MPI: A message-passing interface standard. Computer Science Dept. Technical Report CS-94-230, University of Tennessee, Knoxville, TN, 1994.

[9] W. Gropp and E. Lusk. Scalable Unix tools on parallel processors. In *Proceedings of the Scalable High-Performance Computing Conference, May 23–25, 1994, Knoxville, Tennessee*, pages 56–62, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1994. IEEE Computer Society Press.

[10] William Gropp, David Gunter, and Valerie Taylor. FPMPI: A fine-tuning performance profiling library for MPI, November 2001. Poster presented at SC2001.

[11] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI—The Complete Reference: Volume 2, The MPI-2 Extensions*. MIT Press, Cambridge, MA, 1998.

[12] William Gropp and Ewing Lusk. Installation and user's guide for mpich, a portable implementation of MPI. Technical Report ANL-01/x, Argonne National Laboratory, 2001. The updated version is at ftp://ftp.mcs.anl.gov/pub/mpi/mpichman.ps.

[13] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface,* 2nd edition. MIT Press, Cambridge, MA, 1999.

[14] William Gropp, Ewing Lusk, and Rajeev Thakur. *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press, Cambridge, MA, 1999.

[15] William D. Gropp and Ewing Lusk. Reproducible measurements of MPI performance characteristics. In Jack Dongarra, Emilio Luque, and Tom`as Margalef, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1697 of *Lecture Notes in Computer Science*, pages 11–18. Springer Verlag, 1999.

[16] William D. Gropp and Barry Smith. Chameleon parallel programming tools users manual. Technical Report ANL-93/23, Argonne National Laboratory, Argonne, IL, March 1993.

[17] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. *International Journal of Supercomputer Applications*, 8(3/4):165–414, 1994.

[18] Peter S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufman, 1997.

[19] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI—The Complete Reference: Volume 1, The MPI Core,* 2nd edition. MIT Press, Cambridge, MA, 1998.

[20] Putchong Uthayopas, Thara Angskun, Jullawadee Maneesilp, "*Building A Parallel Computer from Cheap PC: SMILE Cluster Experience*", Parallel Research Group, Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Bangkok.

[21] David W. Walker, " *The Design of a standard massage passing interface for distributed memory concurrent computers*", Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, Dec 1993.

[22] Al Geist, Adam Beguelin, Jack Dongarra, Weisheng Jiang, Robert Manchek, Vaidy Sundram, "*PVM:Parallel Virtual Machine, A User Guide and Tutorial for Networked Parallel Computing*", MIT Press, Cambridge, London, 1994

[23] R. A. Aydt, "*The Pablo Self_De_ning Data Format*", University of Illinois at Urbana-Champaign, Department of Computer Science, February 1993.

[24] M. T. Heath, J. A. Etheridge, "*Visualizing the Performance of Parallel Programs*", IEEE Software, Volume 8, Number 5 September 1991, pp. 29-40.