

Tru64 UNIX Best Practice

Using a Processor Set to Reserve CPUs for a Key Application

March 2001

Product Version: **Tru64 UNIX Version 4.0D or higher**

This Best Practice describes how to set up processor sets (psets) for use by key applications running on the Tru64 UNIX operating system.

Contents

Using a Processor Set to Reserve CPUs for a Key Application

Is This Best Practice Right for You?	1
Before You Begin	2
Applying the Best Practice	7
Verifying Success	11
Troubleshooting	13
Alternative Practices	14
Comments and Questions	16
Legal Notice	16

Using a Processor Set to Reserve CPUs for a Key Application

A processor set is a set of one or more CPUs on a multiprocessor system. By default, the Compaq Tru64 UNIX operating system includes all CPUs in the system in a default processor set (pset) whose identifier is 0, and then runs all user and system processes on the CPUs in processor set 0. You can define an additional processor set and move one or more CPUs from the default processor set to the new one. When you do this, only the programs that you start in or the processes that you move to the new processor set can use its CPUs.

This Best Practice describes how to define a processor set and use it. See the Tru64 UNIX Best Practices Web page for more information about Best Practices documentation.

Is This Best Practice Right for You?

Not all Best Practices apply to all configurations, so you must be sure that this one is appropriate for your system and circumstances. To use this Best Practice, you must meet the requirements described in the following table:

Requirement	Description
Operating System	Tru64 UNIX Version 4.0D or higher. Version 5.0 or higher is required if it is critical that processors automatically restart in the processor set where they had been running when stopped and then restarted with the <code>psradm</code> utility.
System Configuration	An AlphaServer multiprocessor platform, either a traditional SMP or a NUMA system.
Application Type	Technical applications, such as realtime applications, that perform time-critical tasks.

Requirement	Description
System Down Time	Setting up, changing, deleting, and using processor sets is fully dynamic and does not require a reboot to take effect.
Use of System Resources	<p>A user-defined processor set is appropriate when the following conditions apply:</p> <ul style="list-style-type: none"> • It is acceptable to reduce system throughput in order to guarantee that a fixed number of CPUs are always available to one or more time-critical tasks. • You can allocate enough CPUs to the processor set to accommodate the maximum number of processes and threads that run simultaneously to do the time-critical tasks. (Depending on what the application needs, this number does not necessarily mean that there has to be one CPU per thread or task.) • Although the processes and threads running in the processor set require a dedicated set of CPUs, they can share memory and I/O resources with other processes. <p>See the “Performance Issues” section in <i>Before You Begin</i> for background information.</p>

If your system and application do not meet the previous requirements, see *Alternative Practices* for information.

Before You Begin

This Best Practice often refers you to books or reference pages in the Tru64 UNIX documentation set. If you are reading the HTML version of the Best Practice in your web browser, you can click on these cross-references to go to the book or reference page. In this case:

- The book or reference page appears in a second window, not the one where you are reading the Best Practice. You use the original window to navigate the Best Practice and the second window to navigate the other documents or web pages.
- Some of the cross-references are links to books and reference pages in the documentation set for Tru64 UNIX Version 5.1. If you are using another version of the operating system, you can start at the following URL to find the version of the book or reference page most appropriate for your software:

http://www.tru64unix.compaq.com/faqs/publications/pub_page/doc_list.html

Before you apply this Best Practice, you should have information about processor set interface choices, privileges, processor set restrictions, performance issues, and NUMA system issues.

Processor Set Interface Choices

This Best Practice describes how to set up and use a processor set by using commands. You can also set up and use processor sets by using application program interfaces (APIs). For information on these APIs, see the following reference pages: `create_pset(3)`, `destroy_pset(3)`, `assign_cpu_to_pset(3)`, `assign_pid_to_pset(3)`, `print_pset_error(3)`, and `processor_sets(4)`.

Privileges

Defining, deleting, and assigning CPUs to processor sets require superuser privilege. After a processor set is set up, running an application on a processor set or moving processes from one processor set to another does not require special privileges.

Default Processor Set Restriction

The system's master CPU must remain in the default processor set; therefore, you cannot assign the master CPU to a user-defined processor set. Furthermore, unless you explicitly move them, all system processes (such as `init`, `inetd`, and `vold`) run on CPUs in the default processor set. Any user processes that you do not explicitly move to another processor set also run in the default processor set. For these reasons, it is strongly recommended that you do not move so many CPUs out of the default processor set that you severely degrade the performance of the processes that will run there. It is likely that some experimentation will be required to determine the best configuration.

Performance Issues

Setting up a user-defined processor set can waste CPU resources. Even when the CPUs in a user-defined processor set are idle, the operating system cannot use them for any processes or threads that you do not start or move there. Therefore, you should create a user-defined processor set only when you can afford to trade off system throughput in order to guarantee that a fixed number of CPUs is always available to one or more time-critical applications.

Assigning an application to a user-defined processor set also restricts the processes and threads of that application to the CPUs in the user-defined processor set. Therefore, if your goal is immediate CPU availability for a time-critical task, you must be able to afford the cost of a user-defined processor set that provides sufficient CPU cycles to accommodate the maximum number of application processes and threads that will run simultaneously to perform that task.

Setting up a user-defined processor set does not reserve system resources other than CPUs; that is, a processor set does not reserve I/O bandwidth or memory for the application, and therefore is not the same as a system partition. You should set up a user-defined processor set when your application requires dedicated CPUs but can share memory and I/O resources with other applications.

Limiting a CPU to running one or a few processes and threads results in cache and scheduling efficiencies that can provide a significant performance boost to the application. However, that application will not realize the performance boost related to instantaneous CPU availability if the application's requirements for other contested resources (memory and I/O bandwidth) are not met as well. Therefore, before implementing a user-defined processor set, you should gather information about the total average and peak system load (use of CPU cycles, memory, and I/O channels) of all the applications running on the system. On a heavily loaded system that runs a mix of applications, running a selected application in a user-defined processor set might result in minimal performance improvements for that application at the cost of significant performance degradation for others.

Running the `sys_check` utility with the `-perf` option gathers comprehensive system performance information. You can also gather information by using the `collect` utility, which some administrators prefer when it is important to use an information-gathering tool that runs with less impact on system performance. See `sys_check(8)` and `collect(8)` for more information. Refer to the *System Configuration and Tuning* manual for more in-depth discussion of performance issues.

NUMA System Issues

When defining a processor set on a NUMA multiprocessor system (a GS80, GS160, or GS320 AlphaServer system), you need additional information. On a NUMA system, you must:

- Determine the RAD locations of system CPUs

NUMA systems are made up of Resource Affinity Domains (RADs). A RAD is a combination of system resources (CPUs, memory, and I/O channels) that work most efficiently when used together.

Although an application can use resources located in different RADs (a large application might need to), it runs most efficiently when using the CPUs, memory, and I/O channels from the same RAD. It is therefore important to create a user-defined processor set that includes CPUs from one RAD or the fewest number of RADs that are necessary to supply the number of CPUs needed. Otherwise, performance degradation can occur due to an unnecessarily high ratio of remote-to-local memory accesses.

For purposes of this Best Practice when applied to a GS80, GS160, or GS320 AlphaServer system, each RAD contains a maximum of four CPUs. CPUs 0 to 3 are always in RAD 0, CPUs 4 to 7 are always in RAD 1, and so forth. However, for portability to future generations of NUMA AlphaServer systems, scripts and programs must not rely on a fixed relationship of CPU numbers and RAD numbers. See the *NUMA Overview* for more information about Tru64 UNIX NUMA support.

- Determine whether any one RAD (or subset of RADs) has more direct I/O routing to the devices being read from or written to by the application you intend to run in the new processor set

If the I/O processor and host bus adapters for a particular RAD are the only ones connected to the devices containing data accessed by the application, CPUs from that RAD are the best candidates for inclusion in the user-defined processor set.

The `hwmgr -view hier` command displays the locations of I/O devices and other hardware components for each RAD. For GS80, GS160, and GS320 AlphaServer systems, the RAD groupings are bounded by lines that contain `bus wfqbbnumber`. If you know the device names where the application data is stored, you can look for these in the display and determine which RADs contain them. See `hwmgr(8)` for more information.

Gathering Physical Device Information for AdvFS and LSM Logical Devices

Site data is often managed by software, such as AdvFS, LSM, or both, that use logical names that mask the physical device names and other identifiers shown in output from the `hwmgr` command. Therefore, before using the `hwmgr -view hier` command, you might need to use

one or more utilities to get the physical device names that apply. The following information is a quick reference to the order in which you use command-line utilities to map AdvFS and LSM logical device names to physical devices:

If application data is stored in an AdvFS file domain, use the `df -t advfs` or `mount -t advfs` command to find the file domain names for the filesets accessed by the application. Then use the `showfdmn` command on each domain name to display the AdvFS volume names associated with the file domains. If the Vol Name value in this display is a path similar to `/dev/rznumber`, you have the information you need. (Starting with Tru64 UNIX Version 5.0, you will see `/dev/disk/dsknumber`, rather than `/dev/rznumber`.) If the Vol Name value in the output from the `showfdmn` command is in the format `/dev/vol/volume_name`, the AdvFS volume is an LSM volume and you continue looking.

- If application data is stored in LSM volumes, first use the `volprint -v` command on each LSM volume name to return the LSM logical disk names associated with the volumes. Then use the `voldisk list` command on each LSM logical disk name to reveal the special file paths (`pubpath` and `privpath`) to the physical devices.

When you have the physical device names that you need, you can use the `hwmgr -view devices` command to display a list of all the physical devices on the system along with other kinds of identifiers. If the application accesses disks or tape devices on another system through a network connection or the TruCluster Server memory channel, you will not see an identifier for a media device, such as `dsk2`; in this case, you need to determine the identifier of the network or memory channel port through which the disk or tape device is accessed. At this point, you are ready to search for RADs containing the most direct I/O channels by using the `hwmgr -view hier` command.

For more information about using the utilities mentioned in this section, see `df(1)`, `hwmgr(8)`, `mount(8)`, `showfdmn(8)`, `volprint(8)`, and `voldisk(8)`.

You can use the SysMan graphical user interface (GUI) to do the tasks discussed in this section. If you prefer this kind of interface, enter the `sysman -station` command and refer to the online help for guidance.

On NUMA systems, the kernel process (PID 0) has threads running on each RAD to handle I/O processing and memory maintenance. If you move all the CPUs in any RAD out of the default processor set, the kernel automatically migrates those kernel threads to the new processor set at

the time you move the last CPU. Keep this in mind when looking at thread count changes as you move CPUs from one processor set to another.

Applying the Best Practice

Before you apply this Best Practice, be sure to read the information in *Before You Begin*. In particular, remember that you need superuser privilege to create, delete, or assign CPUs to a processor set. Then, follow these steps:

1. Display the identifiers and state of system CPUs by using the `psrinfo` command:

```
# /usr/sbin/psrinfo
0      on-line  since 03/07/2001 14:59:46
1      on-line  since 03/07/2001 14:59:46
2      on-line  since 03/07/2001 14:59:46
3      on-line  since 03/07/2001 14:59:46
4      on-line  since 03/07/2001 14:59:46
5      on-line  since 03/07/2001 14:59:46
6      on-line  since 03/07/2001 14:59:46
7      on-line  since 03/07/2001 14:59:46
8      on-line  since 03/07/2001 14:59:46
9      on-line  since 03/07/2001 14:59:46
10     on-line  since 03/07/2001 14:59:46
11     on-line  since 03/07/2001 14:59:46
12     on-line  since 03/07/2001 14:59:46
13     on-line  since 03/07/2001 14:59:46
14     on-line  since 03/07/2001 14:59:46
15     on-line  since 03/07/2001 14:59:46
#
```

See `psrinfo(1)` for more information.

2. View baseline information about existing processor sets by using the `pset_info` command:

```
# /usr/sbin/pset_info
number of processor sets on system = 1

pset_id # cpus # pids # threads load_av created
0        16    49    324    0.14   03/07/2001 14:59:46

total number of processors on system = 16

cpu #    running primary_cpu pset_id assigned_to_pset
0      1        1        0      03/07/2001 14:59:46
1      1        0        0      03/07/2001 14:59:46
2      1        0        0      03/07/2001 14:59:46
3      1        0        0      03/07/2001 14:59:46
4      1        0        0      03/07/2001 14:59:46
5      1        0        0      03/07/2001 14:59:46
6      1        0        0      03/07/2001 14:59:46
7      1        0        0      03/07/2001 14:59:46
8      1        0        0      03/07/2001 16:33:40
```

```

9          1          0          0      03/07/2001 16:31:21
10         1          0          0      03/07/2001 16:31:21
11         1          0          0      03/07/2001 16:31:21
12         1          0          0      03/07/2001 16:31:21
13         1          0          0      03/07/2001 16:31:21
14         1          0          0      03/07/2001 16:31:21
15         1          0          0      03/07/2001 16:31:21

```

#

The command output shows that only the default processor set (0) exists on this system and that CPU 0 is the master CPU (`primary_cpu = 1`).

This example was generated on a GS160 AlphaServer platform. On this platform, CPUs 0 to 3 are in RAD 0, CPUs 4 to 7 are in RAD 1, CPUs 8 to 11 are in RAD 3, and CPUs 9 to 15 are in RAD 4.

See `pset_info(1)` for more information.

3. Create a new processor set by using the `pset_create` command:

```

# /usr/sbin/pset_create
pset_id = 5
#

```

Identifiers 0 and 1 are never assigned to a user-defined processor set. The default processor set has identifier 0, and identifier 1 is reserved for internal use by the operating system. The numbers assigned to user-defined processor sets are automatically incremented from 1 each time a new processor set is created. Furthermore, processor set identifiers are not reused when the processor sets are deleted. On the system used for this example, processor sets 2, 3, and 4 had been created and deleted before the time the example was created. The new processor set is therefore given 5 as an identifier.

See `pset_create(1)` for more information.

4. Run the `pset_info` command again to see the changes in CPU assignments. For the following example, additional system workload had been generated by running the `usex` (the Unix Systemx EXerciser) program, and the output from the `pset_info` command shows this additional workload:

```

# /usr/sbin/pset_info

number of processor sets on system = 2

pset_id # cpus # pids # threads load_av created
0        16    101    376    18.28  03/07/2001 14:59:46
5         0     0      0      0.00   03/07/2001 16:43:06

total number of processors on system = 16

cpu #    running primary_cpu pset_id assigned_to_pset

```

```

0          1          1          0      03/07/2001 14:59:46
1          1          0          0      03/07/2001 14:59:46
2          1          0          0      03/07/2001 14:59:46
3          1          0          0      03/07/2001 14:59:46
4          1          0          0      03/07/2001 14:59:46
5          1          0          0      03/07/2001 14:59:46
6          1          0          0      03/07/2001 14:59:46
7          1          0          0      03/07/2001 14:59:46
8          1          0          0      03/07/2001 16:33:40
9          1          0          0      03/07/2001 16:31:21
10         1          0          0      03/07/2001 16:31:21
11         1          0          0      03/07/2001 16:31:21
12         1          0          0      03/07/2001 16:31:21
13         1          0          0      03/07/2001 16:31:21
14         1          0          0      03/07/2001 16:31:21
15         1          0          0      03/07/2001 16:31:21

```

```
#
```

The new processor set is empty; however, note the increases in the number of processes, number of threads, and load averages for the 16 CPUs still in processor set 0.

5. Assign the appropriate number of CPUs to the new processor set by using the `pset_assign_cpu` command. In the following example, CPUs 8, 9, 10, and 11 are assigned to processor set 5:

```
# /usr/sbin/pset_assign_cpu 5 8 9 10 11
```

Because this is an AlphaServer GS160 system, all four CPUs assigned to processor set 5 are in the same RAD (RAD 3) to minimize the likelihood of cross-RAD memory accesses. On the AlphaServer GS80, GS160, and GS320 systems, RAD 0 contains the master CPU and therefore can contribute only three CPUs to a user-defined processor set.

After the processor set is in use, you can use the `pset_assign_cpu` command to dynamically adjust how many CPUs (and which ones) are assigned to processor set 5.

See `pset_assign_cpu(1)` for more information.

See the *Troubleshooting* section if you encounter an error using this command.

6. Verify the processor set creation and contents by using the `pset_info` command:

```
# /usr/sbin/pset_info
```

```
number of processor sets on system = 2
```

```

pset_id # cpus # pids # threads load_av created
0        12    49     294    0.15   03/07/2001 14:59:46

```

```

5          4          0          30          0.00          03/07/2001 16:43:06

total number of processors on system = 16

cpu #    running  primary_cpu  pset_id  assigned_to_pset
0        1        1            0        03/07/2001 14:59:46
1        1        0            0        03/07/2001 14:59:46
2        1        0            0        03/07/2001 14:59:46
3        1        0            0        03/07/2001 14:59:46
4        1        0            0        03/07/2001 14:59:46
5        1        0            0        03/07/2001 14:59:46
6        1        0            0        03/07/2001 14:59:46
7        1        0            0        03/07/2001 14:59:46
8        1        0            5        03/07/2001 16:47:27
9        1        0            5        03/07/2001 16:47:27
10       1        0            5        03/07/2001 16:47:27
11       1        0            5        03/07/2001 16:47:27
12       1        0            0        03/07/2001 16:31:21
13       1        0            0        03/07/2001 16:31:21
14       1        0            0        03/07/2001 16:31:21
15       1        0            0        03/07/2001 16:31:21

#

```

- Use the `runon` command to start a job on the new processor set or the `pset_assign_pid` command to move a process to the new processor set.

In the following example, the `usex` program is run again, this time on processor set 5:

```
# runon -p 5 usex -e
```

If other jobs are not already running on the processor set, you can use the `-x` option to the `runon` command to ensure exclusive use of the processor set by the specified program. This option prevents other users or applications from starting additional jobs on the processor set by using the `runon` command, `pset_assign_pid` command, or the `assign_pid_to_pset()` function. Processes started by users who do not use these interfaces and all system processes are restricted to the default processor set whether or not you include the `-x` option.

Note

Processes inherit the processor set assignments of their parent process. For example, starting a user shell in a processor set means that all programs spawned by that shell run in the same processor set as the shell.

In addition to starting applications on a processor set, you can use the `pset_assign_pid` command to move existing processes from one processor set to another. For the following example, assume that the

target processes can be isolated by finding the user account (dbadmin) from which they were started:

```
# ps -oprs,pset,pid,command,user -elf | grep dbadmin
... 0 6313 ... dbadmin
... 0 6311 ... dbadmin
# /usr/sbin/pset_assign_pid 5 6311 6313
# ps -oprs,pset,pid,command,user -elf | grep dbadmin
... 5 6313 ... dbadmin
... 5 6311 ... dbadmin
```

See `runon(1)` and `pset_assign_pid(1)` for more information.

See the *Troubleshooting* section if you encounter an error using the `pset_assign_pid` command.

Verifying Success

After you apply this Best Practice, you can verify whether it was successful by following these steps:

1. Use the `ps` command to get process-specific information. The following example shows information for the `usex` program that was launched by the `runon` command in Step 7 of Applying the Best Practice:

```
# ps -oprs,pset,pid,command,user -elf | grep usex
11  5 15616 sh -c usex -e root
 9  5 15617 /bin/sh -c usex root
10  5 15618 usex -e root
10  5 15619 usex -e root
 9  5 15620 usex -e root
10  5 15621 usex -e root
10  5 15622 usex -e root
10  5 15623 usex -e root
11  5 15624 usex -e root
10  5 15625 usex -e root
11  5 15626 usex -e root
10  5 15627 usex -e root
11  5 15628 usex -e root
10  5 15629 usex -e root
10  5 15630 usex -e root
 9  5 15631 usex -e root
11  5 15632 usex -e root
 8  5 15633 usex -e root
 9  5 15634 usex -e root
 9  5 15635 usex -e root
10  5 15636 usex -e root
10  5 15637 usex -e root
10  5 15638 usex -e root
11  5 15639 usex -e root
 8  5 15640 usex -e root
 8  5 15641 usex -e root
11  5 15642 usex -e root
10  5 15643 usex -e root
```

```

11  5 15644 usex -e      root
11  5 15645 usex -e      root
  8  5 15646 usex -e      root
11  5 15647 usex -e      root
  8  5 15648 usex -e      root
11  5 15649 usex -e      root
11  5 15650 usex -e      root
11  5 15651 usex -e      root
11  5 15652 usex -e      root
  8  5 15653 usex -e      root
11  5 15654 usex -e      root
11  5 15655 usex -e      root
11  5 15656 usex -e      root
10  5 15657 usex -e      root
  7  0 19243 grep usex     root
#

```

- Use the `pset_info` command to examine statistics for the number of processes and threads and for the CPU load averages in different processor sets:

```

# /usr/sbin/pset_info

number of processor sets on system = 2

pset_id # cpus # pids # threads load_av created
  0      12   49   294    0.17  03/07/2001 14:59:46
  5       4   55    85   12.41  03/07/2001 16:43:06

total number of processors on system = 16

cpu #   running primary_cpu pset_id assigned_to_pset
  0     1         1         0      03/07/2001 14:59:46
  1     1         0         0      03/07/2001 14:59:46
  2     1         0         0      03/07/2001 14:59:46
  3     1         0         0      03/07/2001 14:59:46
  4     1         0         0      03/07/2001 14:59:46
  5     1         0         0      03/07/2001 14:59:46
  6     1         0         0      03/07/2001 14:59:46
  7     1         0         0      03/07/2001 14:59:46
  8     1         0         5      03/07/2001 16:47:27
  9     1         0         5      03/07/2001 16:47:27
 10     1         0         5      03/07/2001 16:47:27
 11     1         0         5      03/07/2001 16:47:27
 12     1         0         0      03/07/2001 16:31:21
 13     1         0         0      03/07/2001 16:31:21
 14     1         0         0      03/07/2001 16:31:21
 15     1         0         0      03/07/2001 16:31:21
#

```

If the Best Practice was not successful, see the *Troubleshooting* section for information about identifying and solving problems.

Troubleshooting

If you encounter an error when applying this Best Practice or are not pleased with its results, use the following table to identify and solve problems:

Problem	Possible Causes and Solutions
<p>Error on attempt to assign a CPU to a processor set.</p> <p>Be aware that the messages for errors of this nature might not be helpful. For example, <code>fatal Mach error</code> followed by the number 5 is a catch-all condition that means the requested function could not be performed.</p>	<p>The specified processor set does not exist, the CPU has already been assigned to the processor set, or a process or thread was bound to that CPU by the <code>runon cpu_id</code> command.</p> <p>Double check the ordinal positions of the processor set and CPU identifiers in the <code>pset_assign_cpu</code> command (or the <code>assign_cpu_to_pset()</code> function call) to make sure that you entered valid arguments in the correct order.</p> <p>If processes or threads are bound to the CPU, you must kill or unbind them before assigning the CPU.</p>

Problem	Possible Causes and Solutions
Error on an attempt to move a process to a processor set.	<p>The process was bound to a specific CPU by a <code>runon cpu_id</code> command or to a specific RAD by a <code>runon -r</code> command. The three forms of the <code>runon</code> command are mutually exclusive. It is also possible that the process assignment error happened because another process has exclusive use of the processor set (was started with a <code>runon -x</code> command).</p> <p>Kill or unbind the process or processes causing the problem, or let them finish. If a conflicting <code>runon</code> command bound processes to a CPU or RAD, restart the program with the <code>runon -p</code> command. If the problem was caused by another process having exclusive use of the processor set, retry the <code>pset_assign_pid</code> command.</p>
CPU load is too high for the new processor set, or important processes running in other processor sets are starved for CPU time.	<p>You can move CPUs from one processor set to another to reach an appropriate balance of CPU resources among processor sets. Use the <code>pset_assign_cpu</code> command to do this. You can also adjust processor set locations of different programs or specific processes by using the <code>runon -p</code> or <code>pset_assign_pid</code> command.</p> <p>Any adjustments you make are subject to the restrictions discussed in preceding entries of this table. You must also keep in mind the issues mentioned in <i>Before You Begin</i>.</p>

Alternative Practices

If your system and application do not meet the requirements described in *Is This Best Practice Right for You?*, you can use one of the following alternative methods to ensure that application requirements for system resources are met:

- If the need to have CPUs instantly available for time-critical processes does not outweigh the cost of having CPUs remain idle when they can be used by other processes, use system tuning and class scheduling options to implement resource allocation and run-time priority levels that favor key applications. This is the best course of action when it is important that CPUs do not remain idle when other processes can benefit from the additional CPU cycles. Furthermore, creating a user-defined processor set does not address process contention for

memory and I/O resources. Tuning and scheduling adjustments can address contention for those resources.

See the *System Configuration and Tuning* manual for more information about system tuning.

See `class_admin(8)` and `class_scheduling(4)` for more information about class scheduling.

- If you want to dedicate memory and I/O channels in addition to CPUs to a key application, and you are using a GS160 or GS320 AlphaServer system, then you can set up a partition in which to run the application.

By using processor sets, you can control only CPU availability. Hardware partitions allow you to dedicate a complete set of system resources (memory and I/O as well as CPUs) to one or more key applications. From the perspective of the operating system software, running an application in a NUMA system partition is the same as running the application on a dedicated standalone system.

See your NUMA system hardware documentation for information on setting up a partition. See the “RADs and Partitioning” section of the *NUMA Overview* for information about the software implications of using partitions.

- If an application does not need the dedicated resource assignments provided by a partition, but you want more explicit control over where different applications run on a NUMA system, you can use the `runon -r` command to execute applications on a specific RAD.

Running a program on a specific RAD does not prevent the operating system from running other processes on that RAD; however, it lets you bind a program and associated threads to the most suitable combination of resources. This option is particularly valuable when system resources are not evenly split among all available RADs.

See `runon(1)` for information about the `-r` option of the `runon` command. Note that the `-r` option to the `runon` command is available starting with Tru64 UNIX Version 5.1.

- If your program spawns threads that can benefit from cache coherency across changes in program context, you can use the `runon cpu_id` command to bind the application process and threads to a particular CPU. This command does not prevent the operating system from running other processes on that CPU, but ensures that a suspended thread of the bound process always restarts on the same CPU where it last ran. Depending on the elapsed time and what has run during that

time, the state of the cache when the thread is restarted is more likely to be close to what it was when the thread was suspended.

See `runon(1)` for information about the `cpu_id` operand in the `runon` command.

Comments and Questions

We value your comments and questions on the information in this document. Please mail your comments to us at this address:

`best_practices@zk3.dec.com`

Legal Notice

COMPAQ and the Compaq logo are registered in U.S. Patent and Trademark Office. Tru64 is a trademark of Compaq Information Technologies Group, L.P.

UNIX and The Open Group are trademarks of The Open Group.

All other product names mentioned herein may be trademarks or registered trademarks of their respective companies.

Confidential computer software. Valid license from Compaq required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is provided "as is" without warranty of any kind and is subject to change without notice. The warranties for Compaq products are set forth in the express limited warranty statements accompanying such products. Nothing herein should be construed as constituting an additional warranty.