SPP Tips and Tricks

- Sn_cnsl
- System Console
- File System
- Test Station Console
- Booting
- Problems Booting
- Partitioning Disks
- Creating Stripes
- Setting the the correct boot device on Multi Node machines
- How much paging space should I have?

Sn_cnsl

To Start sn_cnsl:

sn_cnsl -S 1
: Start the console with spy and show the last few lines of output.

sn_cnsl -F 1
: Start the console forced and show the last few lines of output.

sn_cnsl -s 1
: Same as -S but don't show lines.

sn_cnsl -f 1
: Same as -F but don't show lines.

While in sn_cnsl:

^ec.
: Quit sn_cnsl.

^ecf
: Force the console.

^ecs
: Spy the conosle

System Console

The system console starts with '-name system_console' so if you want to turn on logging set the following defaults in sppuser's .Xdefaults file.

```
system_console*logFile:                 /tmp/Mylog
system_console*logging:                 true
system_console*background:              white
system_console*font:                    fixed
```

```
system_console*foreground:                black
```

Now, a word about .Xdefaults. An application looks for defaults in the following order. (from the X manpage).

```
Resource Manager something like xrdb.
Screen Resources also xrdb.
Application Specific files /usr/lib/X11/appdefaults.
XENVIRONMENT which if not set defaults to .Xdefaults
```

Sn.cnsld does not run as sppuser it run's as root, so if you don't load sppuser's .Xdefaults using xrdb, sn.cnsld is going to look in /.Xdefaults.

File System (This applies to Landmark only)

The server's buffer cache block size is 64K, and the default filesystem block size is 8K if you are looking into increasing the server's buffer cache, (it's a tunable) you might instead consider reformating your file system to something larger than 8K.

Test Station Console

If running X on the test station, either as sppuser, or another user make sure there is a console 'xterm -C', this is not the 'System Console.'

Booting

From Test Station

1. Log into the test station as sppuser.
2. cd /spp/os
   This is where the kernel, server, and other needed files are stored.
3. If installing a new kernel, copy the new kernel to something like 'mach.new', and the server to 'server.new.'

   ```
   % cp /mach ./mach.new
   % cp /server ./server.new
   ```

   There are probably two symbolic links mach and server, delete these links.

   ```
   % ls -l
   . . .
   lrwxrwxrwx 1 sppuser 19 May 13 11:23 mach_kernel@ -> /spp/os/mach.ppp_r2
   . . .
   lrwxrwxrwx 1 sppuser 21 May 13 11:23 server@ -> /spp/os/server.ppp_r2
   . . .
   % rm mach_kernel server
   ```

   Now link mach and server to your mach.new, and server.new

   ```
   % ln -s mach.new mach_kernel
   % ln -s server.new server
   % ls -l
   . . .
   ```

```
lrwxrwxrwx 1 sppuser 19 May 13 11:25 mach_kernel@ -> /spp/os/mach.new
. . .
lrwxrwxrwx 1 sppuser 21 May 13 11:25 server@ -> /spp/os/server.new
. . .
```

If they aren't symbolic links move mach to mach.orig, and server to server.orig, and then create the symbolic links.

```
% ls -l
. . .
-rwxr-x--- 1 sppuser 19 May 13 11:23 mach
. . .
-rwxr-x--- 1 sppuser 21 May 13 11:23 server
. . .
% mv mach mach.orig
% mv server server.orig
% ln -s mach.new mach_kernel
% ln -s server.new server
```

Now, you can boot the system.

```
% sppboot
```

From Disk

1. Log into the test station as sppuser.
2. cd /spp/os This is where the kernel, server, and other needed files are stored.
3. If you're installing a new kernel you can't boot from disk :)
4. Pull the primary loader, and obp from the EEPROM to memory, and then start obp.
   ```
   % ccmu pull
   % do_reset
   ```
5. You should get an 'ok' prompt in your System Console window (this will take a few seconds/minutes) when you get the prompt type boot in the System Console window.

   If there is a prompt, and you can't type in the window. Try a 'Ctrl-e c f' to force the console.

Problems Booting

If you get the following error message while booting:

```
Node 0: Alert: Unable to get boot configuration variable ROOT_FS_NODE value
(returned 0x4).
```

The machine will hang due to a software conflict between OBP 1.1 and the 1.0 release of the OS. This normally should not happen. The only case of this happening was due to multiple sd0 labaled disks on multiple scsi interfaces on the same node.

The **ROOT_FS_NODE** value is the indicator. The variable is no longer valid in newer versions of OBP.

Partitioning Disks

```
%diskutil
DiskUtil: sel d <disk> <- ex. sd1
DiskUtil: sh p


   Logical disk name: sd1
   partition table: (space available for file systems = 2098744)
   part    offset      size    | partition description          | flags
   ------------------------------------------------------------------------
    a:          0K  1024000K  |                                |
    b:    1024000K  1024000K  |@                               |


DiskUtil: make p b size 0
DiskUtil: sh p


   Logical disk name: sd1
   partition table: (space available for file systems = 2098744)
   part    offset      size    | partition description          | flags
   ------------------------------------------------------------------------
     a:          0K  1024000K  |                                |



DiskUtil: make p a size 2097152000
DiskUtil: sh p
```

This should make your partition 2G. FYI 2097152000 is 2048000 * 1024.

To make a partition an additional swap partition you probably just need to do:

```
DiskUtil: set p <x> F D
```

This should set the D flag in the flags field.

Be sure to reboot, and then newfs any partitions you changed.

```
% newfs /dev/rdsk/sd<xp> scalios
```

And you should be done.

Creating Stripes

Here's how to create a stripe.

With the machine in single user mode.

```
% diskutil
DiskUtil: create stripe <stripe name> (<partitions>)
```

or

```
DiskUtil: create stripe <stripe name> [ Blocksize <size>] ( <partitions>)
```

An Example:

```
% diskutil

DiskUtil: create stripe stripe0 (sd0a sd1a sd2a sd3a)
```

Would create a 4 way stripe on sd[0-3]a.

Setting the the correct boot device on Multi Node machines

When connecting multiple machine on the same DART bus it is necessary to redefine the boot device on the the nodes that are not 0. What will happen is that the node1-n will be alaised to boot off of node0's sda0. At this time the other nodes cannot get to the disk and the boot will fail.

On normal nodes you will see the following defined if you use the command **devalias**:

```
ok devalias sd0a

sd0a /landmarc@0,ffec0000/sbus@f,fcffff00/Convex,afws@1,10000:narrow/sd@2,0:a
```

This will be fine for node0. However, the rest of the nodes will also have sdoa defined this way. The suggestions is to make a new alias for each node that points to that nodes sd0a. Using the new alias you will set the boot-device to point to that alias.

For node1:

```
ok nvalias bd
/landmarc@10000,ffec0000/sbus@f,fcffff00/Convex,afws@1,10000:narrow/sd@2,0:a
```

This will set an aliase to the drive at scsi address 2 on node1, which is generally the boot device. The **bd** can be any alias, **bd** is just an example. To make this drive the boot-device issue the following:

```
ok setenv boot-device bd
```

It is possible to set the boot device directly from the device tree without using the alias. For node2 it would look like the following:

```
ok setenv boot-device
/landmarc@20000,ffec0000/sbus@f,fcffff00/Convex,afws@1,10000:narrow/sd@2,0:a
```

You should notice the difference in each of the node2 device lines. The **/landmarc@#** tells you where the disk is. The **#** indicates the node.

Paging

How much paging space should I have?

The general rule is to have twice as much paging space as you have memory.

Can I stripe the Default Pager

No, the default pager doesn't go through the file system device, it works directly with the raw device. Almost the same effect can be achieved by setting

up multiple Default Pagers. For best performance, one default pager per disk should be set up.

Last modified: Tue Jun 6 16:32:08 CDT 1995
Shane Holder, SPP Product Support, holder@convex.com

The following was put together by Hank Michels. Thanks Hank. :)

Paging on SPP in a nutshell

I got a couple of emails lately asking for more information on paging and swapping on SPP. Here is a brief summary:

Swapping:

There is no such thing in SPP-UX - just paging. However all pages of a process can be paged out.

Paging:

There are two pager in SPP-UX:

- the default pager (which is part of the kernel)
- the vnode pager (which is part of the server)

Default Pager:

All dynamic allocated data is going to the default pager partition(s), this is all malloced data, data on the stack etc.

The default pager is in the kernel and is backed by one or more paging _partitions_. Default location is sd0b, the second partition on the root disk, default size is one Gigabyte.

A partition is marked as default pager partition by using the diskutil(1m) utility:

```
Diskutil: SET Partition  Flag Default_pager
```

You need to reboot to activate this partition.

To reclaim a default pager partition for regular filesystem use, you need to:

```
Diskutil: UNSet Partition  Flag Default_pager
```

and then reboot.

Use diskutil(1m) also to check for the default pager partitions:

```
DiskUtil: sel d sd1
DiskUtil: sh pa
Logical disk name: sd1
partition table: (space available for file systems = 2098752)
part    offset      size    | partition description          | flags
-------------------------------------------------------------------
 a:          8K  2098744K  |@                                | *D
             ^                                                 ^^
             | if the partition is at the                     /  \
             | beginning of the disk you              * means   Default
```

```
                | need an offset of 8K                          active   Pager Flag
```

Please note that currently there must be at least one default pager partition active per node. The kernel will attempt to page to its local paging partition, and will seek space on other nodes if it cannot find any. (please note: There are some problems with remote paging at the moment)

The following are boot messages for the default pager:

```
[72000001 001204a4 1:5] (default pager): Added paging device sd2b (262144 pages)
[72000001 001204a4 2:1] (default pager): Added paging device sd4a (524688 pages)
[72000001 001204a4 3:1] (default pager): Added paging device sd5a (524688 pages)
[72000001 001204a4 0:3] (default pager): Added paging device sd0b (262144 pages)
[72000001 001204a4 1:1] (default pager): Added paging device sd3a (524684 pages)
[72000001 001204a4 0:2] (default pager): Added paging device sd1a (524686 pages)
                        ^                                          ^^^^^^
                        node the disk is connected to             one page
                                                                     is 4K
```

In this example 6 default pager partitions were activated, two on node zero and node one, one on node two and three.

Vnode Pager:

All static data (initalized data) and mmap'ed data goes to the vnode pager.

The vnode pager is in the server and is backed by one or more _files_ in one or more filesystem. Default location is /paging/space, which is in the root filesystem. The size and name of the vnode paging file are defined in /etc/src.sh (here a vnode pager file of 16M to 64M is defined):

```
PAGING_FILE=/paging/space ; export PAGING_FILE
PAGING_MINSIZE=16M ; export PAGING_MINSIZE
PAGING_MAXSIZE=64M ; export PAGING_MAXSIZE
```

The vnode pager file(s) are activated with the swapon command. On activation the file is created with a size defined by PAGING_MINSIZE. It can grow until PAGING_MAXSIZE is reached or the filesystem the paging file lives in is full. In this case the behaviour of the server is somehow undefined (there is a good chance the server panics then). Please note that on multi node systems the server does not check if a vnode pager file is on a _local_ filesystem. This simply means that all data which is paged by the vnode pager goes to a random node (if you have several vnode paging files distributed over your nodes (it probably goes always via node 0)) or to node 0 (if you have just one vnode pager file) Normaly a vnode pager file of 64MB should be sufficient, since most application have just a small number of static data. However, if many Fortran applications are running on the complex there is a good chance that the default vnode paging space of 64MB is not sufficient.Also if one mmap's a large region of anonymous memory, or does a MAP_PRIVATE of a large file, then the paging file will need to be sufficiently large to accommodate it.

If necessary, you can add default pager files on the fly. Just do:

```
swapon -l  -h
```

To get rid of a vnode pager file you need to reboot and then rm(1) it.

The following is a boot message for the vnode pager:

```
swapon: adding /paging/space as paging file: low water
```

```
mark 16777216, high water mark 67108864
```

Remarks:

Please note that the way paging is done under SPP-UX is different to HP-UX. The default and vnode pager are part of the OSF1/AD mikrokernel architecure SPP-UX is based on. However there is a good chance that the vnode pager goes away one day. :-)

We believe that all usage of the /paging/space file has been moved into the default pager. However, the file remains, and the change in question has been put in very recently. May 30, 1995

Also there are still a couple of bugs in both the default and the vnode pager code (well, the default pager is approaching bug-free status). Sometimes the system panics if there is not enough paging space available.

Unfortunatley there is currently no utility available to monitor the usage of the paging partitions and file, the only way to get an idea of the vnode pager usage is to watch the size of the paging files (like /paging/space)

If you have any questions or corrections, please let me know....

Hank

With all of the above said, thanks again Hank, here's some information on how to configure Virtual Memory (VM) on the SPP.

Configuring VM on the SPP

Swap/Paging partitions

The default pager understands how to make efficient use of multiple paging partitions, so the more disks configured with paging partitions, the better performance will be on the SPP.

The recommended configuration for paging on the SPP is to have twice as much paging space as physical RAM, and have the paging distributed across all disks on the node. For example:

```
System configuration:
        4 Nodes
       1G physical RAM/node
        8 Disks on node 0
        4 Disks on nodes 1, 2 and 3
```

For information on how to create partitions see Partitioning Disks in the SPP Tips and Tricks Document.

1G of RAM per node would indicate the need for 2G of swap space for each node.

```
Node 0:
        256M per disk.  8 disks * 256M = 2G

Nodes 1-3:
```

```
512M per disk.  4 disks * 512M = 2G
```

This is not a requirement, but a suggestion

Problems with paging

There are several problems with paging on the SPP. In time these problems will be resolved, most likely in the 5.0 - 6.0 release.

The system crashes when it runs out of paging space.

> This problem has been worked around by creating a high watermark tunable that forces the system to use a percentage of total virtual memory available instead of all available memory.
>
> The reason for this "hack" is that for some elements of the OS, there is no way to determine how much space is actually being used, for example, if a machine has 512M of virtual memory (virtual is physical ram + swap space) and the OS says it is using 180M, then there is 332M available for user processes. If the OS is actually using 256M, and reporting only 180M then there is actually only 256M of space for user processes.
>
> What the watermark does is force the system to only use a percentage of VM, so it doesn't matter how much memory the system thinks it's using, we know it's using more, and thus set the watermark appropriately. The default for this tunable is 90%, so that means only use 90% of available virtual space.

Can't log into my system when no local paging space available

> This is a new problem due to the above fix. When no local paging space is available new processes are not allowed to start. This is a regression in the system, and should be fixed in a future release.

System gets slow when paging remotely

> If a system consists of multiple nodes, it is possible for a process to start paging remotely, this means that if a node runs out of local paging space it will start using the paging space on other nodes.
>
> Remote paging is **extremely** slow, if a process ends up paging to a remote node, it can have a slowdown effect on the rest of the processes if it monopolizes the memory on that node. If the system is trying to make "free" pages, it currently doesn't distinguish between "local" pages and "remote" pages. The resulting slowdown in the "recycling" of pages shrinks the pool of free pages and limits the continuing availability of pages. THIS is what will appear to slow down the entire node.

Can't get statistics on paging space

> Yes, we know, and this will be fixed in a future release of the OS.

Send mail to Shane

Shane Holder <holder@convex.com>

Last modified: Mon Jun 5 12:07:34 CDT 1995

Explanation of GMEM

Now that multinode operation is becoming the norm, I thought it might be a good time to revisit some of the restrictions regarding the configuration of subcomplex global memory. In particular, there are some counterintuitive aspects to what can be done to maximize the amount of global memory. Since the memory configuration problem is difficult to explain, I'll start at the very beginning and work up from there. There is a summary at the end.

The PA-RISC 7100/7200 chips present only 32 address bits to the world. To address memory on other nodes, you could distribute a maximum of 4GB of memory across all of the nodes. That was deemed unacceptable for a system of the size of an SPP, so an alternative was found to allow you to access up to 4GB of memory on node as well as memory on another node. The mechanism that gives you this functionality is the BDT (Block Descriptor Table), one of which is on each node. From a given address, 8 bits are extracted and used to index the BDT to yield a 4-bit code. The 4-bit code gives the node number that the memory is on. Each BDT entry "maps" a 16MB region of memory. Virtually all physical memory management that relates to global memory is done in units of 16MB.

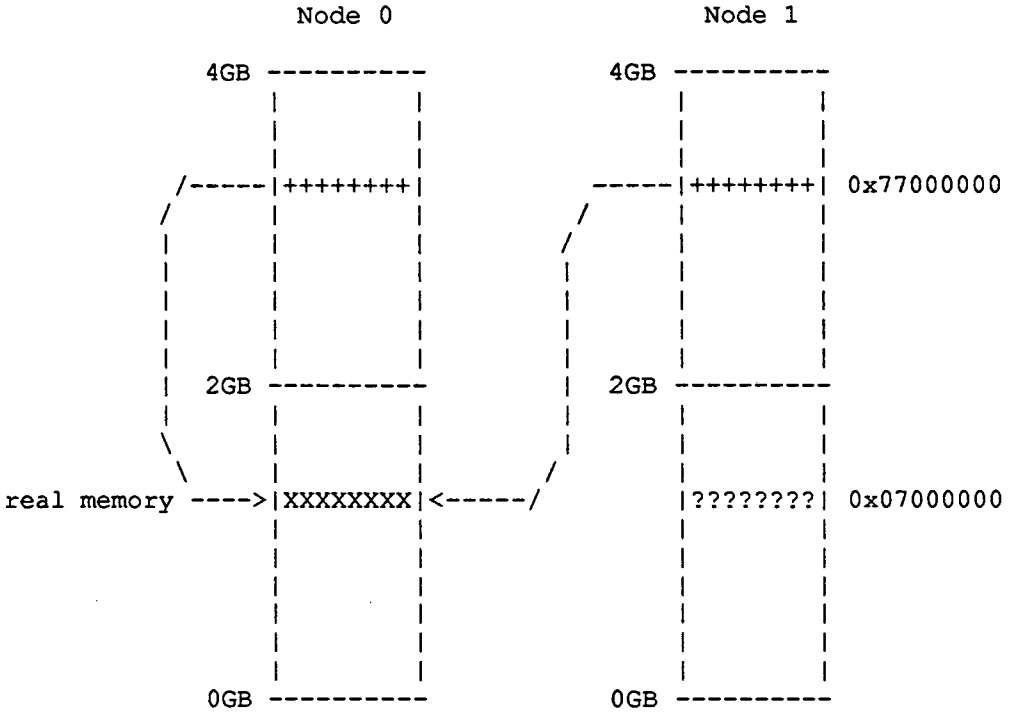
The 256 available BDT entries are used/allocated in several different ways. First, 16 are off limits because they map the I/O space. 4 or 5 others are used by the OS. That leaves 238 entries for normal use. All of the physical memory on the machine is divided up into 16MB chunks that each correspond to a BDT entry. So if you have a system with 2GB of physical memory, there are 128 additional BDT entries used by the system to map the physical RAM, leaving 108 free to create global memory pools.

By definition, global memory is memory that can be accessed from any node participating in the subcomplex that the global memory is in. This is done through the use of BDT's. For example, consider an address that is in a global memory region but is remote for a particular node. The upper 8 bits of the address are used to index that node's BDT and yield the 4-bit node number that the memory is really on. The resulting 36-bit address is accepted by the hardware and the memory is fetched. Because any address in a global memory region is accessible from any node, for every 16MB region of global memory, every node participating in that subcomplex must have a BDT entry allocated to refer to that memory. Returning to the example of a machine with 2GB on each node, that means that no matter how many nodes you have, you can never have more than 108 * 16 ~= 1.7GB in a global memory pool. However, a machine with only 1GB of physical memory per node and 4 nodes could potentially allocate 2.7GB because more BDT's are free after taking care of basic housekeeping issues.

The above calculations are both first-order and ignore some other issues. One of these issues is how the CTI cache size can affect how much memory is eligible for allocation to a global pool. The CTI cache is a physically indexed direct mapped cache with 64 byte lines. What that means is that the hardware deals with global memory in chunks of 64 bytes, so even if you only want one 4-byte word the hardware deals with the entire chunk of 64 bytes that the word lies in. Further, where a given 64 byte line ends up in the CTI cache is determined by it's physical address. Finally, there is only one place in the CTI cache that a given 64-byte line can be placed. If another line is there, it is evicted. Due to a hardware error, memory located in the uppermost 16MB chunk of the CTI cache can be corrupted. As a result, in the current 3.0.2 OS, no memory block that would map to that section of the CTI cache is eligible to be part of a global memory region. So if you have a 64MB CTI cache, 16/64 = 1/4th of your memory cannot be used as global as it might be pulled into the upper block of the CTI cache on some other node. Since no chunk that is in a global pool will map

to the uppermost 16MB chunk, the effective size of your CTI cache is actually 16MB less than what is configured. This particular problem has been partly alleviated for the next release of the OS.

The other principal problem has to do with BDT aliasing. BDT aliases are the degree of freedom used to create global memory. On any machine, the number of aliases for a BDT entry is usually 4GB divided by the the amount of physical memory on the machine (some aliases are disallowed because the would land in the I/O space). Consider the 2-node, 2GB per node case (please forgive the ASCII art).



This diagram depicts a single 16MB chunk contributed by node 0 to a global pool. On a node with 2GB of physical memory, each 16MB chunk has 2 "aliases" corresponding to the masking of the uppermost bit in the original 32-bit physical address. An access on either node to 0x77000000 will result in accessing the data in the physical memory on node 0 (the chunk filled with X's). One thing that is bad about this picture is that the memory at address 0x07000000 on node 1 cannot be used as global memory and instead must be used as node private. If we add another 2GB node to the above system, the memory at address 0x07000000 also cannot be used as global.

On 2 node machine with 1GB per node, there are 4 aliases for a given 16MB chunk. As a result, the 16MB chunk on node 1 can be utilized. Further, if we added one more node we could use the corresponding 16MB chunk on that node as well. However, if a fourth node was added, there would be insufficient aliases to allow the use of that 16MB chunk on the fourth node for global memory.

To summarize, the amount of global memory that is allocatable is a complex function of:

1. free physical memory
2. # of aliases, which is dependent on the largest amount of physical memory on any node participating in the subcomplex
3. size of the CTI cache.

In general, plenty of free memory (like at the time of boot), less amount of physical memory (to get the largest number of aliases), and a large CTI cache will maximize possible global memory on

systems with many nodes. On systems with fewer nodes than aliases, maximum freedom is available.

Please call or email me if you have any questions regarding this.

- Dave Boles

The following memory totals are guidelines of what the maximum achievable amounts of global memory are likely to be at boot, with a 2% buffer cache configured. There are no guarantees that these amounts are achievable, although many of them have been tested and confirmed.

Memory configs are listed as (global mem per node) / (total global mem)

2 nodes

cti cache	physical memory			
	256	512	1024	2048
64	80 / 160	336 / 672	832 / 1664	816 / 1632
128	16 / 32	256 / 512	768 / 1536	816 / 1632
256	-	128 / 256	640 / 1280	816 / 1632
512	-	-	384 / 768	704 / 1408

4 nodes

cti cache	physical memory			
	256	512	1024	2048
64	80 / 320	336 / 1344	576 / 2304	400 / 1600
128	16 / 64	256 / 1024	544 / 2176	400 / 1600
256	-	128 / 512	480 / 1920	400 / 1600
512	-	-	304 / 1152	352 / 1408

6 nodes

cti cache	physical memory			
	256	512	1024	2048
64	80 / 480	336 / 2016	384 / 2304	272 / 1632
128	16 / 96	256 / 1536	352 / 2112	272 / 1632
256	-	128 / 768	304 / 1824	272 / 1632
512	-	-	192 / 1152	224 / 1344

8 nodes

cti cache	physical memory			
	256	512	1024	2048
64	80 / 640	288 / 2304	288 / 2304	192 / 1536
128	16 / 128	224 / 1792	256 / 2048	192 / 1536
256	-	128 / 1024	224 / 1792	192 / 1536
512	-	-	128 / 1024	176 / 1408

Shane Holder <holder@convex.com>
Last modified: Wed May 31 11:34:47 CDT 1995

Sub-Complex Configuration Guide

This guide is not to be the definitive answer to Sub-Complex (SC) configurations, but rather, as the name implies, a guide.

The SC's should be configured based on what the customer is going to be doing with the system, there is no correct configuration. Let's go over a few scenarios.

Single Node

This one's easy, SC's aren't very useful in this configuration. Don't use them unless the customer insists. It will be more trouble than it's worth.

Multi Node

This is where things get highly dependent on what your customer is doing.

I'm not going to worry about memory considerations at this point. "How much GMEM should I have?" will get explained later. So ignore the GMEM=for now.

Generic Codes

The system is to be used to run several small jobs, like in an educational environment, or as a general purpose compute server, where there are several mixes of code types and sizes.

The best thing to do with this type of site is to create a mix of different sized SC's. This is difficult to do on a two node system, but if there are more nodes the SC can be more creative. On a four node system I'd do something like: (what follows is the output of `'scm -c'`, please refer to scm(4))

```
NODEID=0: CTICACHE=16:
NODEID=1: CTICACHE=16:
NODEID=2: CTICACHE=16:
NODEID=3: CTICACHE=16:

#Subcomplex Id 1
SC=System:
        UID=0: GID=0: PERM=0555:
        POLICY=1:
        NODEID=0:
                GMEM=0:
                PROCID=0:
                PROCID=1:
#Subcomplex Id 2
SC=gsm6:
        UID=0: GID=3: PERM=0777:
        POLICY=1:
        NODEID=0:
                GMEM=128:
                PROCID=2:
                PROCID=3:
                PROCID=4:
                PROCID=5:
                PROCID=6:
                PROCID=7:
```

```
#Subcomplex Id 3
SC=gsm8:
        UID=0: GID=3: PERM=0777:
        POLICY=1:
        NODEID=1:
                GMEM=256:
                PROCID=0:
                PROCID=1:
                PROCID=2:
                PROCID=3:
                PROCID=4:
                PROCID=5:
                PROCID=6:
                PROCID=7:
#Subcomplex Id 4
SC=gsm16:
        UID=0: GID=3: PERM=0777:
        POLICY=1:
        NODEID=2:
                GMEM=256:
                PROCID=0:
                PROCID=1:
                PROCID=2:
                PROCID=3:
                PROCID=4:
                PROCID=5:
                PROCID=6:
                PROCID=7:
        NODEID=3:
                GMEM=256:
                PROCID=0:
                PROCID=1:
                PROCID=2:
                PROCID=3:
                PROCID=4:
                PROCID=5:
                PROCID=6:
                PROCID=7:
```

This type of configuration gives the users a variety of ways in which to execute code. You have 6 way, 8 way, and 16 way parallel SC's in which to run code. Again, these are just suggestions, if the customer wants a 24 CPU SC, then make them one.

Don't put everything in `System`

I wouldn't suggest a configuration where every CPU is in the `System` SC. There are several reasons for this:

- If the `System` SC span nodes then processes may be started on any node, so `inetd` is allowed to create a `rlogin` session on node 1. If this happens and the user wants the machine to be reconfigured so that the `System` SC is only on node zero, the re-config will fail, because there is an active process on node 1, and processes/threads can't migrate across nodes.
- If a process like `rlogind` gets started on a node that doesn't have the networking interface attached to it, then messages must be sent from one node to another for everything that the user does.
- Since the only way to deallocate GMEM is destroy the SC that that GMEM lives in, you can not get rid of any GMEM allocated to the `System` SC, because you can't destroy `System`

This type of configuration makes it almost impossible to reconfigure without a reboot.

Specific Codes

The system was purchased for a specific job. Other jobs will be run on this machine, but it's main purpose is to run only a few codes.

Everything above applies here, the only difference is there's usually one large SC.

How many CPU's should be in the `System` SC

There is no clear cut definition of how many CPU's should be in the `System` SC. I would recommend at least 2 for a system with several users. Some users may ask why so much horse power is required for a bunch of daemons, but you have to remember that every character that a user types, or requests to be printed goes through one of these daemons, and the overhead for system calls is quite substantial at this point.

Global Memory

GMEM is physical memory, and is placed into a SC in 16M chunks. In order for the memory to be placed into the SC, it must be un-used at the time of allocation. This is why it is suggested that SC's be configured at boot time, before the system has a chance to pollute physical memory.

There are 4 ways in which an application can require GMEM.

Going Multinode
: If the subcomplex in which a multithreaded application run's is a multinode subcomplex, you will need GMEM, and if the application is not taylored to just put the required data in GMEM, lots of GMEM will be needed.

System V Shared Memory
: All System V Shared Memory goes into GMEM. If an application uses System V Shared Memory via shmget(2), then it will require GMEM.

memory_class_malloc
: If the application explicitly calls memory_class_malloc, with an argument of far_shared, or near_shared, then it will require GMEM.

mmap(2)
: If the application uses mmap, and asks for MAP_SHARED, CNX_MAP_FAR_SHARED, CNX_MAP_NEAR_SHARED, then it will require GMEM.

For multinode subcomplexes you should configure as much GMEM as you can. For those sites that use parallelism through MPI or PVM, you won't need as much GMEM as those that go parallel via -O3.

If a code is compiled at -O3, or uses CPS, almost all of the space that the program uses goes into GMEM, this includes text, data, and bss, this is the default for the compiler. There are some parameters that mpa(1) and the loader will allow you to specify that change this behavior.

If the SC doesn't span nodes, and the code to be run doesn't do one of the four things above, then you don't need much GMEM. Versions of CPS before 3.5.5.5 explicitly map some data to GMEM, so a little bit of GMEM is required for systems that don't have ALL 3.5.5.5, then some GMEM

needs to be configured in each SC.

You shouldn't configure GMEM in the `System` SC. The reason for this is to discourage users from running parallel code in the SC, causing paging problems.

Reconfiguring Sub-Complexes

There are a few gotcha's that a sysadmin should be aware of when reconfiguring SC's

- To reconfigure GMEM you must first destroy the SC, and then re-create it. Beware, that in the deletion of a SC memory may become poluted and thus unavailable to put back into your SC.
- If all CPU's are removed from a SC, any threads in that SC will suspend until a CPU is allocated to the SC. Also, if a SC spans nodes, and all of the CPU's in one node of the SC are removed, all threads running on that node will suspend, but the threads on the remaining node will continue to run.
- In order to delete a SC, there must be no threads in that SC, so the sysadmin must kill any processes in that SC before deleting the SC.
- A System SC must exist, and it can not be deleted, you also can not remove all processors from the System SC.

Send mail to Shane

Shane Holder <holder@convex.com>
Last modified: Tue Jun 27 11:16:30 CDT 1995