SGI® Altix™ NumaTools

Reiner Vogelsang
SGI GmbH
reiner@sgi.com

January 19, 2005

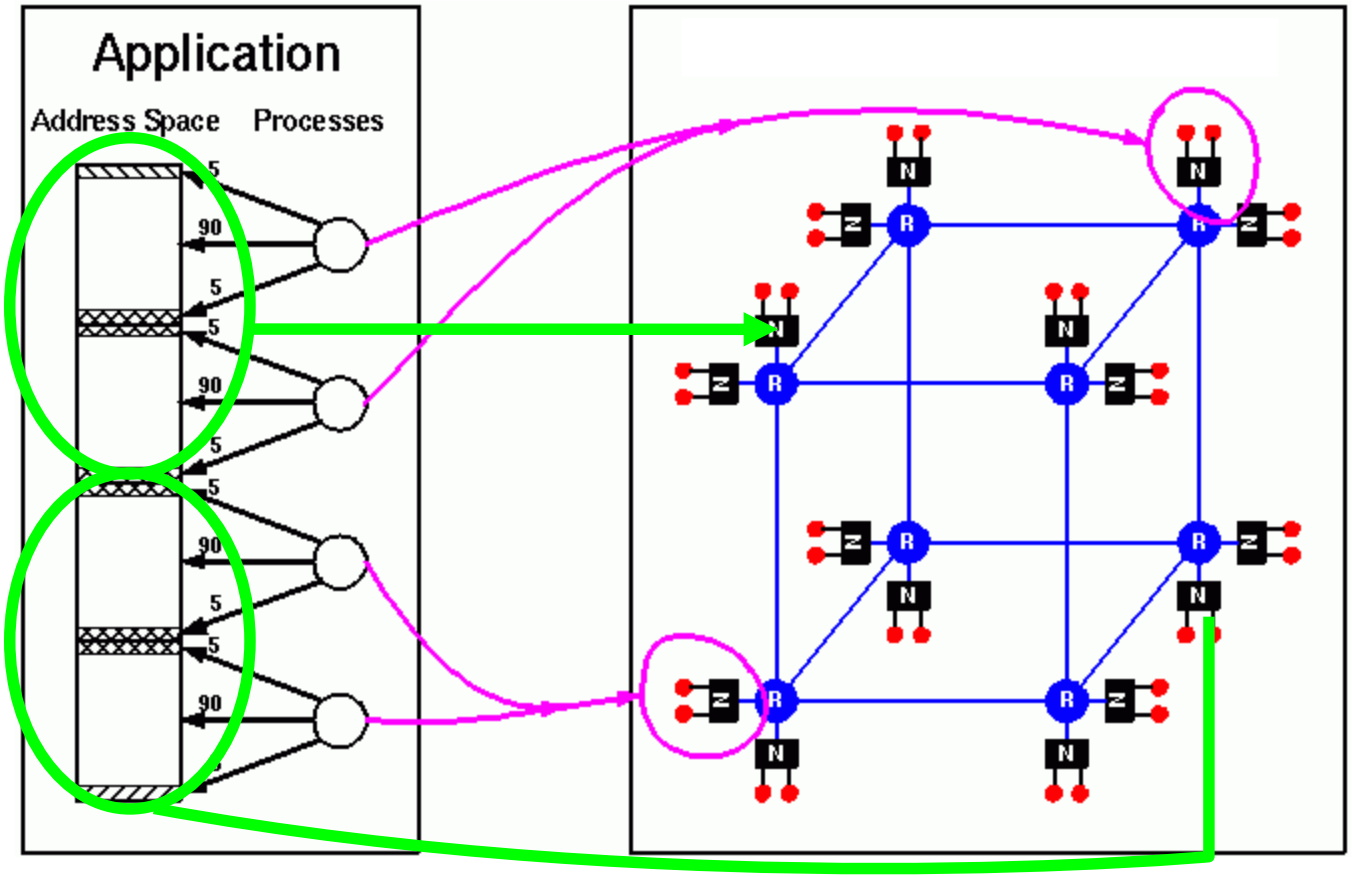
Module Objectives

After completing the module you will understand

- Memory Placement Policies
- Control Placement of Processes
- Analyze the memory placement of processes for better performance of your application

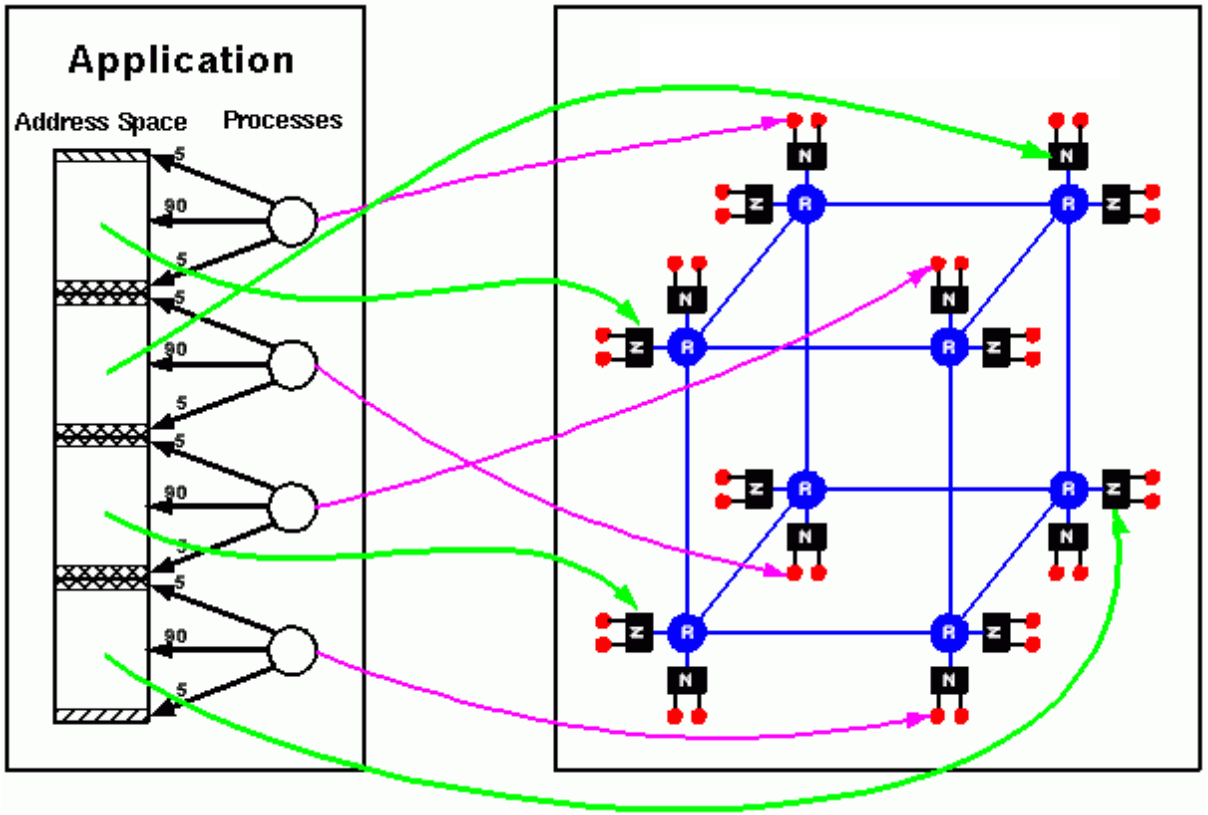
Memory placement

- Non optimal placement



Memory Placement

- Random placement

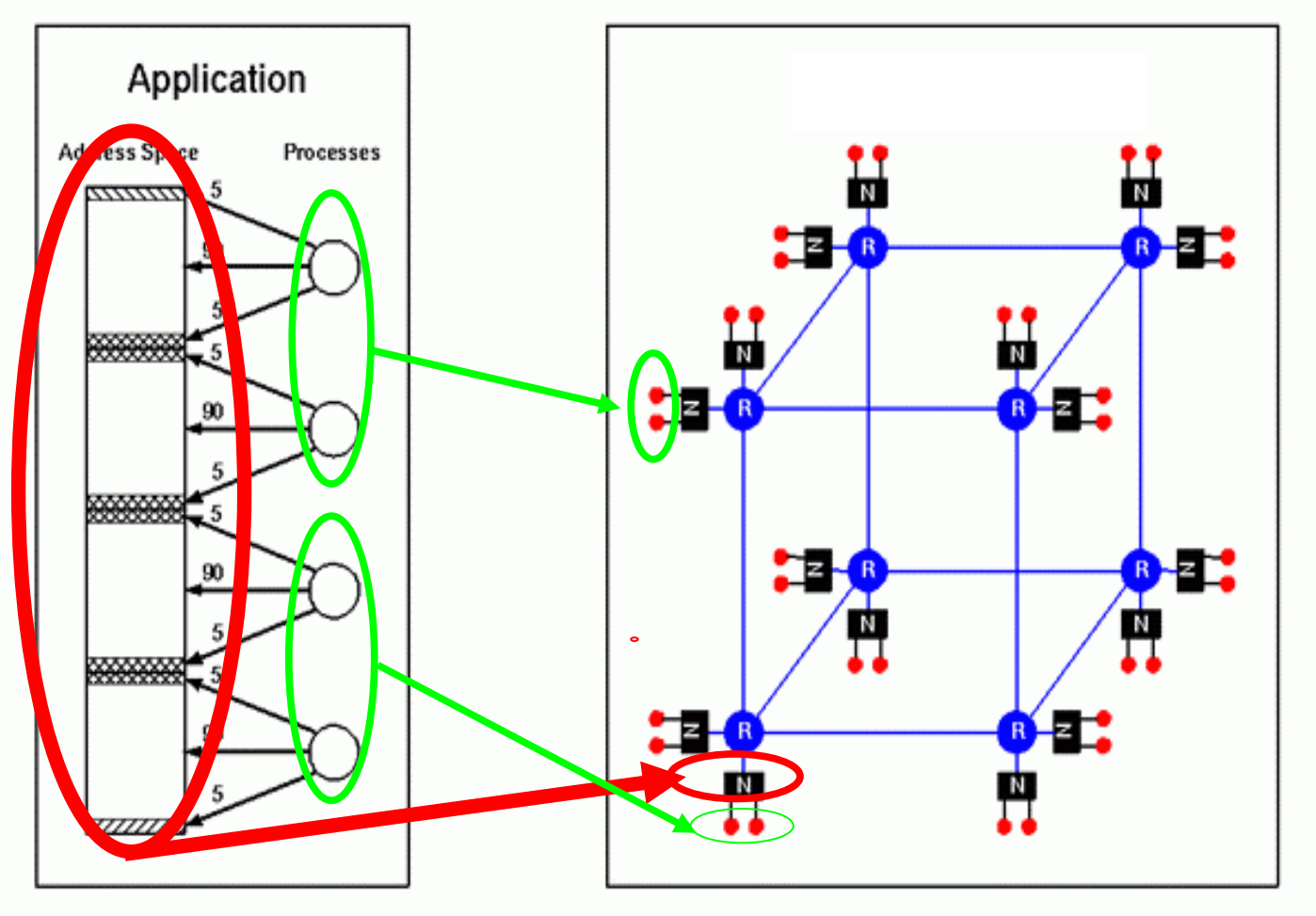


Data placement policy

- First touch
  - Default policy
  - First processor to touch a page of memory causes it to be allocated from its local memory
  - Works well for fully parallelized programs, but serial initialization cause non-local accesses and bottlenecks.

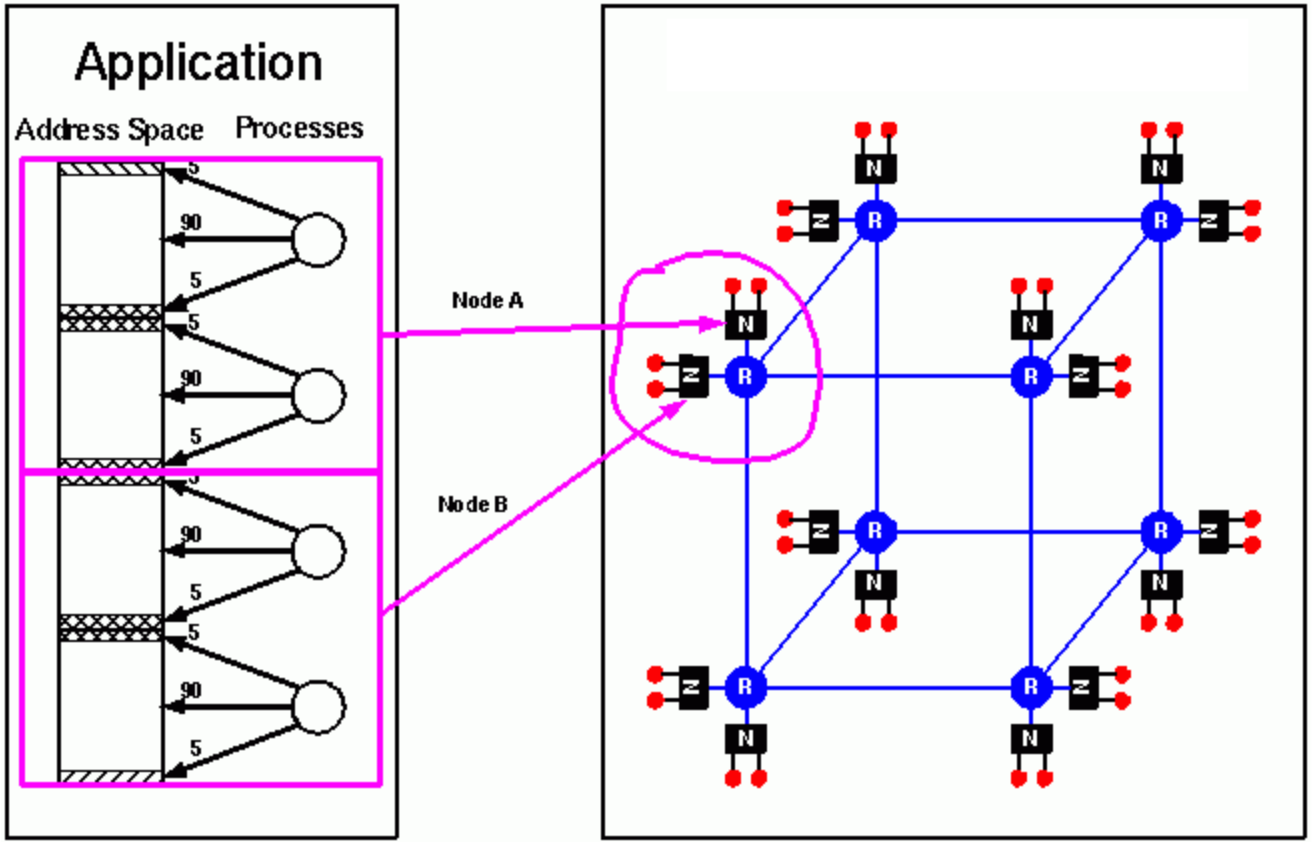
Data Placement Policy: First touch bottleneck

- Non-parallel initialization



Memory placement

- Optimal memory placement



Running on Specific CPUs -- taskset

- Known as '`runon`' under Irix and Redhat ProPack 3.
  - Syntax slightly different. Consult the man page.
- The `taskset` command executes a command on a CPU or list of CPUs:

```
taskset -c 1,3-5 ./a.out
```

- Find idle CPUs using `top`
- `taskset` restricts the process and its children to run on the set of listed CPUs, but does not prevent them from moving around within it.
- Numbering scheme of logical CPUs is relative to the corresponding cpuset.

Running on Specific CPUs -- cpuset

- `cpuset(1)` allows you to run your programs on a restricted subset of processors and associated memory, called a cpumemset
- It requires the prior creation of a cpumemset to run the program in
- cpumemsets may be created:
  - Manually by root : `cpuset -c <name> -f <file name>`
  - Automatically by batch schedulers such as Platform's LSF or Altair Engineering's PBSPro
- Run something in a cpuset: `cpuset -i my_cpuset -I a.out`
- Determine existing cpumemsets with `ls -d /dev/cpuset`
  - You get a list of created cpusets.
  - `cat /dev/cpuset/<name>/cpus` lists CPUs belonging to the set.
- As usual, read the `man` pages!

Running on Specific CPUs -- cpuset

- The command `cpuset` provides a CPU and memory container for your application.
- Example of a cpuset description file:

```
#cpuset configuration file
cpu_exclusive
mem_exclusive
cpus 17-19, 21, 23
mems 8-11
```

- `cpuset -c matafubatu  -f this_file`
  - Creates a directory /dev/cpuset/`matafubatu`
- Permissions of files of directory /dev/cpuset/<name> controls who can run on a cpuset.
- SGI work on cpusets are accepted by the Linux community.

Binding Threads on CPUs -- dplace

- Important flags:

  `-c <cpulist>` CPU numbers are logical numbers relative to current cpumemset.

  `-x <mask>` A bitmask for specifying threads to skip placing. [See following examples.]

  `-s <count>` Skip placement of the first <count> threads. Use `-s1` to skip placing the shepherd thread in MPI programs.

  `-q` Displays static load information. `dplace` without arguments will avoid loaded cpus.

  `-e` Exact placement

- Note: numbering scheme ist cpuset relative.

Binding Threads on CPUs -- dplace

- **Beware of glibc pthreads changes!**
  - **ProPack 2.4: glibc 2.2.4 / linuxthreads**
  - **ProPack 3: glibc 2.3.2 / Native Posix Thread Library (NPTL)**
- **NPTL does not have a pthread monitor thread--impacts placement of pthreads and OpenMP codes**
  - ProPack 2.4 OpenMP: **`dplace -x6`** …
  - ProPack 3 OpenMP: `dplace -x2 -c 0-3`…
    (still need to skip OpenMP shepherd thread)
  - **ProPack 4 OpenMP: `dplace -x 2 -c 0-3`**
  - '`setenv LD_ASSUME_KERNEL 2.4.19`' in ProPack 3 to revert to old Linuxthreads behavior

Binding Threads on CPUs -- dplace

- Use of profiling tools may require modification of placement flags. E.g., OpenMP program on ProPack 4.0:

```
dplace -x5 -c0-15 histx -o prof a.out
```

`histx`	skip	(1)
`a.out` master	place	(0)
OpenMP monitor	skip	(1)
`a.out` slave1	place	(0)
`a.out` slave2	place	(0)
…	place	(0)

$101_2 = 5_{10}$

Or use explicit placement:

```
dplace -e -c x,0,x,1-15 histx …
```

- Always use dplace in conjunction with OpenMP programs!

Determining Data Access Patterns -- `dlook(1)`

- **`dlook(1)` allows you to display the memory map and CPU usage for a specified process**

  `dlook [-a] [-c] [-h] [-l] [-o` *`outfile`*`] [-s` *`secs`*`]`
  ` `*`command`*` [` *`command-args`*`]`

  `dlook [-a] [-c] [-h] [-l] [-o` *`outfile`*`] [-s` *`secs`*`]` *`pid`*

- **For each page in the virtual address space of the process, `dlook(1)` prints the following information:**
  - **The object that owns the page, such as a file, SysV shared memory, a device driver, etc.**
  - **The type of page, such as random access memory (RAM), FETCHOP, IOSPACE, etc.**

Determining Data Access Patterns -- `dlook(1)`

- For RAM pages, the following are also listed:
  - memory attributes (SHARED, DIRTY, etc.)
  - node that the page is located on
  - physical address of page, if option `-a` is specified
- With option `-c`, `dlook(1)` also prints the amount of elapsed CPU time that the process has executed on each physical CPU in the system.

%

Determining Data Access Patterns -- dlook(1)

```
 dlook ls
anaconda-ks.cfg  install.log  install.log.syslog
```

```
Exit:  ls
Pid: 12905      Thu Aug 22 10:45:34 2002

Process memory map:
  […]
  4000000000000000-4000000000024000 r-xp 0000000000000000 04:13 58723137   /bin/ls
  [4000000000000000-4000000000004000]            1 page  on node   2  MEMORY|SHARED
  [4000000000004000-4000000000008000]            1 page  on node   0  MEMORY|SHARED
  [4000000000008000-400000000000c000]            1 page  on node   1  MEMORY|SHARED
  [400000000000c000-4000000000010000]            1 page  on node   2  MEMORY|SHARED
  [4000000000014000-4000000000018000]            1 page  on node   1  MEMORY|SHARED
  [4000000000018000-400000000001c000]            1 page  on node   2  MEMORY|SHARED
  [400000000001c000-4000000000020000]            1 page  on node   3  MEMORY|SHARED
[…]
```

Data Locality

- If working in a non-batch environment, be aware of per-node memory availability:

```
% cp bigbigfile /fastfs
% numactl --hardware
reiner@dcm27 108> numactl --hardware
available: 8 nodes (0-7)
node 0 size: 7874 MB
node 0 free: 5467 MB
node 1 size: 7888 MB
node 1 free: 5705 MB
node 2 size: 7888 MB
node 2 free: 5814 MB
```

- In I/O intensive environment, Linux gladly eats up available memory for I/O buffer cache

Data Locality

- `bcfree` utility frees non-dirty buffer (and optionally slab) cache pages to the system (requires root privileges):

```
% bcfree -as
% numactl --hardware
node 0 size: 7874 MB
node 0 free: 7364 MB
node 1 size: 7888 MB
node 1 free: 7607 MB
node 2 size: 7888 MB
node 2 free: 7717 MB
```

- Under ProPack 4.2 bcfree only in rare cases needed. Candidate buffer cache pages will be tossed automatically in low-memory situations

Data Locality

- Alternatively one can use `posix_fadvise` with the advice `POSIX_FADV_DONTNEED`

```
#define _GNU_SOURCE
...
#include <fcntl.h>
#include <asm/unistd.h>

main (int argc, char *argv[]) {
    long i,fd,result;
    for(i=1;i<argc;i++) {
        fd = open(argv[i],O_RDWR);
....
 result=posix_fadvise(fd,0,0x3777777777777,POSIX_FADV_DONTNEED);
        close(fd);
    }
    return;
}
```

Recommendations for Achieving Good Performance

- Remember: It's just shared memory
- CPUmemsets, `dplace` and `dlook` are there to help them run well
- Memory is allocated and distributed in pages
- First tune single-processor performance
- Cache friendly programs will run well on the NUMA architecture
- Memory-intensive, cache unfriendly programs
- If scaling is less than expected, data placement may be a problem
- Make sure the data are uniformly distributed
- When developing new code use first-touch and program with it in mind


sgi®