Shared Memory & Message Passing Programming on SCI-Connected Clusters

Lab Session

Joachim Worringen, RWTH Aachen

SCI Summer School 2000
Trinitiy College Dublin

Software Setup

- Frontend scripts are:
  - `smicc`, `smirun` for SMI
  - `mpicc`, `mpirun` for MPI
- Pathes should be working:

  verify with `type smicc`
- Copy examples to your home directory:

```
tar xf ~worringen2k/summer_school.tar
cd summer_school
```

- Open a shell window, from there: `xemacs &`

SMI Tutorial

- First steps towards SMI: `hellowordld`
  - Compiling & starting SMI applications
- Shared Memory Regions:
  - Establishing different layouts
  - Access characteristics
  - Passing pointers
  - Synchronization
- A common problem: `sort`
  - Direct approach
  - Optimized approach

helloworld

```
cd summer_school/SMI/helloworld
```

- Compiling the programm:

  ```
  smicc helloworld.c -o helloworld
  ```

- Starting the programm:

  ```
  smirun -np 2 helloworld
  ```

- Starting the programm & see what happens:

  ```
  smirun -np 2 -v helloworld
  ```
  (-verbose)

  ⇨no „machine file“ found!

- Create a machine file!

machine file

- List of nodes/hosts on which SMI programms can be run:

```
pc628
pc629
pc630
pc631
```

- Locations where `smirun` looks at:
  - `-machinefile` *`filename`*
  - `./machines`
  - `$(HOME)/.machines`
  - copy machine file to `~/.machines`
- If not found: All processes on the local host
- Special option: `-np N -nodes node0 ... nodeN-1`

More `smirun` Options

- Output to seperate terminal windows:

  `-xterm` (`xhosts` & `DISPLAY` need to be set correctly)
- Pipe output through a pager:

  `-pager` (can be configured via env. variables)
- Redirect input / output:

  `-stderr filename, -stdout filename, -stdin filename`

  Separate file generated for each process
- Verbose startup:

  `-v`

SHM Regions: Creation & Access

`cd summer_school/SMI/regions: make`

load `test_undivided.c`

Create different memory layouts:

- measure access time for each process.

Region types to be used:

- SMI_SHM_UNDIVIDED (test_undivided.c)
- SMI_SHM_BLOCKED (test_blocked.c)
- SMI_SHM_CUSTOMIZED (test_customized.c)

SHM Regions: UNDIVIDED

load `measure_undivided.c`

`make measure`

Common situation:

- each process exports a region
- then imports all other regions

Simple approach:

- For n process, create n UNDIVIDED regions

SHM Regions: FRAGMENTED

load `measure_fragmented.c`

Advanced approach:

- create a single FRAGMENTED region

⇨ Compare duration of creation!

Passing Pointers

Addresses of common shared memory regions:

- SISCI: different start address for each process
  - ⇨ passing of pointers not possible!
- SMI: identical start address for each process

load `pass_pointers.c`

- Sharing a linked list between processes
- Test SMI_SHM_NONFIXED - result ?
- Synchronization is required!

Synchronization Techniques

Locks:

- protection of objects (data structures)
- sequentialization of code segments
- Polling

Barriers:

- Collective synchronization (all processes need to participate)
- Polling

More Synchronization

Progress counters:

- Individual synchronization possible
- Polling

Signals:

- Individual synchronization possible
- Not polling
  - ⇨Callbacks possible

Copying memory

- Different types of memory copying techniques achieve best results in different situations
- `SMI_Memcpy()` takes care of chosing the right technique, BUT:
  - being smart costs time
  - hints by the user help to save time
- Utilizing DMA for asynchronous operation:
  - `SMI_Imemcpy()` starts/enqueues operation
  - `SMI_Memwait()` waits for completion
  - very low CPU load

membench

`cd summer_school/SMI/membench`

- use `membench` with 2 or 4 processes
- results in `membench.out`
- unidirectional write: no options
- bidirectional write: `-b`
  - results in `membench.out.x` (for each process)
- read instead of write: `-r`
- DMA instead of PIO: `-a`
  - use `top` to compare CPU load with/without `-a`

SCI-MPICH Tutorial

First steps towards SCI-MPICH: `hellowor ld`

- Compiling & starting SCI-MPICH applications

- Configuration of SCI-MPICH:
  - Protocols
  - General attributes
  - Startup information
- Performance impacts on buffer sizes
- Asynchronous Message Passing
- `sorting` via MPI

helloworld

`cd summer_school/SCI-MPICH/helloworld`

- Compiling the programm:
  `mpicc helloworld.c –o helloworld`
- Starting the programm:
  `mpirun –np 2 helloworld`
- Starting the programm & see what happens:
  `mpirun –np 2 –v helloworld` (-verbose)
  ⇨uses SMI machines file
- `mpirun`: same additional options as `smirun`

SCI-MPICH Configuration

- Configuration on startup via *device configuration file*
  - use default name `ch_smi.conf` in `.`
  - specify via `-devconf` option to mpirun
- Syntax: `keyword numeric_value`
- Different classes of keywords:
  - `SHORT_xy`: short protocol
  - `EAGER_xy`: eager protocol
  - `RNDV_xy`: rendez-vous protocol
  - `XY`: general variable

SHORT Protocol

- Variables:
  - `SHORT_nbrbufs`
    - number of slots for short/control messages
      "number of outstanding transactions"
  - `SHORT_bufsize`
    - determines maximum size for "inlined messages"
    - observe the latency for small messages!

EAGER Protocol

- Variables:
  - `EAGER_nbrbufs`
    - number of fixed allocated buffers to receive eager messages
    - disable eager protocol by setting to 0
  - `EAGER_bufsize`
    - size of these buffers
    - observer protocol switch to rendez-vous above this size!

RNDV Protocol

- Variables:
  - `RNDV_memorysize`
    - size of the dynamic memory pool to receive rendez-vous messages
    - observe bandwidth for messages above this size!
  - `RNDV_blocksize`
    - determines write-leave interleave blocksize
    - observe bandwidth for all rendez-vous messages

sort

Compare the performace of a parallel sort algorithm (split-merge):

- Shared-Memory implementation with SMI
  - different optimization stages
  - located in SMI/sort
- MPI implemenation
  - communication via TCP/IP
  - communication via SCI
  - located in SCI-MPICH/psort