An intoduction to QCDOC Computer

Chulwoo Jung, etc... for QCDOC collaboration

September 1, 2004

Contents

Chapter 1

Introduction

This is intended to be an introduction to using QCDOC computer. Currently, the author is using QCDOC Operating System (QOS) with CVS tag 2-1-2.

There are a couple more excellent notes on QOS, available on the web:

`http://www.ph.ed.ac.uk/~bj/QCDOC_quick_start.html`

by Baint Joo

`http://quark.phy.bnl.gov/~creutz/qcdoc/using.txt`

by Mike Creutz

The style and even some contents may be similar to an excellent manual for QCDSP by Balint Joo, whehter he is listed as an author of this manual or not...

Chapter 2

Tutorial: "Hello World"

In this chapter we show how to compile and run a simple program on a QCDOC computer.

2.1 Reserving partition

The first step any user should take is to reserve a machine partition Through QCDOC allocation web page (https://qcdoc.phys.columbia.edu). QCDOC allocation webpage shows which "partition", sets of nodes group together to run a single program. The user should remember the partition name (**ssbp2/01node/mach08** in the first entry) to start qdaemon or connect qcsh to running qdaemon 2.2

Users can click on non-allocated links and submit time to complete allocation Fig. 2.2 (Currently, the time is merely to be informative to other users only. Allocated partition will continue to be until the user de-allocate the partition.)

2.2 Front end

Similar to QCDSP, QCDOC is accessible through a front end machine, (qcdochost0.phys.columbia.edu at the moment). Users should have an account on the front end with appropriate read/write access to various software tools on the front end. This will be done by placing users in appropriate UNIX user groups, which we had not completely sorted out yet. Also, remote access is allowed only through SSH protocol.

2.3 Compiling for QCDOC

Once the user logged onto the front end, there will be installed QOS's in a public area. The sample session below was generated largely with qos-2-0-stable, but users should use the most up-to-date installed QOS (currently) /qcdoc/sfw/qos/qos-1-9-4/ unless you are developing/debugging QOS yourself.

The easiest way to set up compiling environment is to source (QOS directory)/(setup.csh or setup.sh, depending on your choice of shells). This sets up various environment variables and paths, including QOS environment variable which points to the top of QOS installation directory and GCC_EXEC_PREFIX, which set up compile environment for the cross-compiler(powerpc-gnu-elf-{gcc,g++}. Then, the cross-compiler can be used the same way as other GCC compilers, except it generates binary for QCDOC instead of the host machine.

```
# set the qos directory Environment variable (used by CPS)
setenv QOS /qcdoc/sfw/qos/qos-1-9-4/
setenv QOS_VERS 1.9.4
echo "QOS_VERS=$QOS_VERS"

# This will point gcc at the right
setenv GCC_EXEC_PREFIX ${QOS}/quser/gcc-lib-user///

# point the user to the correct GCC for the quser compile chain
# and the correct qshell
setenv PATH /qcdoc/local/gcc-3.2.2/bin/:${QOS}/qhost/bin/:${PATH}
```

IMPORTANT: Please beware that setting GCC_EXEC_PREFIX will make all the other compilers except the cross-compiler non-functional.

If you do not use setup script, (current Makefile in $QOS/quser does not require the setup script to be run beforehand) Users should user the same version of cross-compiler as the QOS is compiled for. For example, /qcdoc/sfw/qos/qos-1-9-4 is configured for GCC 3.2.2. The compilers are installed in /qcdoc/local/gcc-{version number}. Be sure to include the path for the correct version. If you are not sure, the version can be checked by using '–version' with the compiler:

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% powerpc-gnu-elf-g++
--version
3.2.2
```

Users can see how to compile for QCDOC by looking at $QOS/quser directory:

```
CVS/                      configure*            plaq/
Makefile                  configure.in*         qalloc/
Makefile.in*              gcc-lib-user/         qcd_api/
Makefile.user.rules       gint_print/           scu_simple/
Makefile.user.rules.in*   gint_test/            src/
config.log                glb_passthru/         usertest/
config.status*            include/              wilson_dslash/
```

Every directory except gcc-lib-user/, src/, qcd_api/ include/, and qalloc/ contains sample programs. Each directories includes a simple makefile which largely defers to $QOS/quser/Makefile.user.rules:

```
include ../Makefile.user.rules

all: usertest.x

clean:
	- rm -f $(CLEANFILES)

usertest.x: $(OBJECTS)
	env GCC_EXEC_PREFIX=../gcc-lib-user//// $(CXX) $(OBJECTS) -o usertest.x
```

$QOS/quser/Makefile.user.rules has the rules for compiling codes

```
#-----------------------------------------------------------------------------
# PAB Aug 2003
#-----------------------------------------------------------------------------

CXX = powerpc-gnu-elf-g++
CC  = powerpc-gnu-elf-gcc
LD  = powerpc-gnu-elf-ld
AR  = powerpc-gnu-elf-ar
AS  = powerpc-gnu-elf-as
TOPDIR=/qcdoc/sfw/qos/qos-1-9-4
CFLAGS=-Wa,-m405     -I$(TOPDIR)/quser/include/ -fno-exceptions -fno-hosted
CXXFLAGS=-Wa,-m405 -I$(TOPDIR)/quser/include/ -fno-exceptions -fno-hosted
CPPFLAGS= -I$(TOPDIR)/quser/include/
LDFLAGS="-Wa,\"-init _start\""
ASFLAGS=-m405

LINKLIBS=

#-----------------------------------------------------------------------------
# how to make an object file from a C++ source
#-----------------------------------------------------------------------------
%.o : %.C
env GCC_EXEC_PREFIX=$(TOPDIR)/quser/gcc-lib-user/// $(CXX) -c $(CXXFLAGS) $<

#-----------------------------------------------------------------------------
# how to make an object file from a c source
#-----------------------------------------------------------------------------
%.o : %.c
env GCC_EXEC_PREFIX=$(TOPDIR)/quser/gcc-lib-user/// $(CC) -c $(CFLAGS) $<

#-----------------------------------------------------------------------------
# how to make an object file from a .S source
```

```
#-----------------------------------------------------------------------------
%.o : %.S
env GCC_EXEC_PREFIX=$(TOPDIR)/quser/gcc-lib-user/// $(CC) -E $(CFLAGS) $< > $*.i
env GCC_EXEC_PREFIX=$(TOPDIR)/quser/gcc-lib-user/// $(AS) $(ASFLAGS) -o $*.o $*.i
rm $*.i
```

(continued)

2.4 Running QCDOC

User must start the qdaemon session for the allocated partition if it is not already present (**ssbp2/02node/mach00** in this example) in a separate window by, for example,

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% xterm -e $QOS/qhost/test/qdaemon
-m ssbp2/02node/mach00 &
```

A qcsh session is started by

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% $QOS/qhost/qcsh/qcsh
```

You can use $QOS/qhost/test/qtest script to do the same.

First, qcsh must connect to qdaemon. This is done by 'qinit' command.

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% qinit ssbp2/02node/mach00
..created system socket for commands.. ..authorization complete.
```

Now user must connect to one of the "partition"'s. This is done by 'qpartition_connect -p' command. User can, in principle, create/reserve/unreserve/destroy "user" partitions. Here we connect to the "root" partition(0).

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% qpartition_connect -p 0
executing... qpartition_connect -p 0
CWD is /home/chulwoo/qos-doc-sessions/
QD:TextParser::doqpartition_op.MESSAGE> Entered
QD:PartitionManager::process().MESSAGE> Operation 24
QD:PartitionManager::qpartition_connect.MESSAGE> Attempting to connect Client
clnt-0001 to Partition 0
QD:PartitionManager::qpartition_connect.MESSAGE> Connection Established
QD:PartitionManager::process().MESSAGE> Dispatching code 0 to client
QDaemonReturn 0
QKerReturn 0
AppReturn 0
```

Now we have to load the boot kernel which is loaded via Ethernet-JTAG and does the basic setup and testing before run kernel is loaded.

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% qreset
executing... qreset
CWD is /home/chulwoo/qos-doc-sessions/
QD:Partition::PrintState.MESSAGE> Boot kernel is NOT running
QD:Partition::PrintState.MESSAGE> Run kernel is NOT running
QD:Partition::PrintState.MESSAGE> Serial communications are NOT up
QD:Partition::PrintState.MESSAGE> Application axes are NOT mapped
RBC:Boot0::doBoot.MESSAGE> Building transporters
RBC:Boot0::doBoot.MESSAGE> Building packets
RBC:Boot0::BuildPackets.MESSAGE> Processing Elf boot0 kernel
RBC:Boot0::BuildPackets.MESSAGE> /home/chulwoo/QCDOC/qos-2-0-stable/qos/qker/b
ker/bker.x
RBC:Boot0::elf2jtag.MESSAGE> 6 P-sections
RBC:Boot0::BootNodes.MESSAGE> 1 Spooling threads
RBC:Boot0::BootNodes.MESSAGE> Thread 0 has 2 nodes
RBC:Boot0::NICBootNodes.MESSAGE> Thread 0: JTAG UpdateIp
RBC:Boot0::NICBootNodes.MESSAGE> Thread 0: JTAG ResetProc
RBC:Boot0::NICBootNodes.MESSAGE> Thread 0: JTAG HaltProc
RBC:Boot0::NICBootNodes.MESSAGE> Thread 0: JTAG Instruction Pipe Stuff
RBC:Boot0::NICBootNodes.MESSAGE> Thread 0: JTAG Kernel Write
RBC:Boot0::NICBootNodes.MESSAGE> Thread 0: JTAG Set IP/MCaddr
RBC:Boot0::NICBootNodes.MESSAGE> Thread 0: JTAG Exec Kernel
RBC:Boot0::doBoot.MESSAGE> BootNodes Succeeded
RBC:Boot0::doBoot.MESSAGE> Checking nodes claim success
RRBC:Boot0::AnalyzeBoot.MESSAGE> Contacted all nodes
RBC:Boot0::doBoot.MESSAGE> Destroying Packets
RBC:Boot0::doBoot.MESSAGE> Destroying Transporters
QD:Partition::qreset.MESSAGE> Reset was successful
QDaemonReturn 0
QKerReturn 0
AppReturn 0
```

'qboot' command loads the run kernel via Ethernet. The run kernel contains operating system for the user program.

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% qboot
executing... qboot
CWD is /home/chulwoo/qos-doc-sessions/
QD:Partition::PrintState.MESSAGE> Boot kernel is running
QD:Partition::PrintState.MESSAGE> Run kernel is NOT running
QD:Partition::PrintState.MESSAGE> Serial communications are NOT up
QD:Partition::PrintState.MESSAGE> Application axes are NOT mapped
QD:Partition::qboot.MESSAGE> Booting nodes
```

```
RBC:Boot1::doBoot.MESSAGE> Building transporters
RBC:Boot1::doBoot.MESSAGE> Building packets
RBC:Boot1::BuildPackets.MESSAGE> Processing elf kernel
RBC:Boot1::BuildPackets.MESSAGE> /home/chulwoo/QCDOC/qos-2-0-stable/qos/qker/r
ker/rker.x
RBC:Boot1::elf2bkerp.MESSAGE> 4 P-sections
RBC:Boot1::elf2bkerp.MESSAGE> Elf entry is 200298
RBC:Boot1::process_psect.MESSAGE> 345 packets
RBC:Boot1::process_psect.MESSAGE> 43 packets
RBC:Boot1::process_psect.MESSAGE> 3371 packets
RBC:Boot1::process_psect.MESSAGE> 2 packets
RBC:Boot1::BootNodes.MESSAGE> NIC 0 has 2 nodes
RBC:Boot1::NICBootNodes.MESSAGE> NICBootNodes: BKERP SYNC
RBC:Boot1::NICBootNodes.MESSAGE> NICBootNodes: BKERP Write
RBC:Boot1::NICBootNodes.MESSAGE> NICBootNodes: BKERP Execute
RBC:Boot1::doBoot.MESSAGE> Destroying Packets
RBC:Boot1::doBoot.MESSAGE> Destroying transporters
QD:Partition::qboot.MESSAGE> Boot was successful
QDaemonReturn 0
QKerReturn 0
AppReturn 0
```

Now.. we are ready to tell QCDOC to find the topology of the partition! 'qdiscover' command trains SCU and execute neighbor swap to figure out how many nodes there are in each of 6 dimensions.

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% qdiscover
executing... qdiscover
CWD is /home/chulwoo/qos-doc-sessions/
QD:Partition::PrintState.MESSAGE> Boot kernel is NOT running
QD:Partition::PrintState.MESSAGE> Run kernel is running
QD:Partition::PrintState.MESSAGE> Serial communications are NOT up
QD:Partition::PrintState.MESSAGE> Application axes are NOT mapped
QD:Partition::qscu_train.USER> Setting up SCU hardware
QD:Partition::qscu_train.USER> Starting training process
QD:Partition::qscu_train.USER> Completing training process
QD:Partition::qscu_train.USER> Switching to idle bytes
QD:Partition::qscu_train.USER> Releasing Dircom units
QD:Partition::DiscoverTopology.MESSAGE> Starting neighbour swap
QD:Partition::DiscoverTopology.MESSAGE> Completing neighbour swap
QD:Partition::DiscoverTopology.MESSAGE> Dimension 0 has length 1 nodes
QD:Partition::DiscoverTopology.MESSAGE> Dimension 1 has length 1 nodes
QD:Partition::DiscoverTopology.MESSAGE> Dimension 2 has length 1 nodes
```

```
QD:Partition::DiscoverTopology.MESSAGE> Dimension 3 has length 1 nodes
QD:Partition::DiscoverTopology.MESSAGE> Dimension 4 has length 1 nodes
QD:Partition::DiscoverTopology.MESSAGE> Dimension 5 has length 2 nodes
QD:Partition::DiscoverTopology.ERROR> FIXME: Not checked each individual link
neighbour
QD:Partition::DiscoverTopology.MESSAGE> Setting up MBTopology map
QD:Partition::DiscoverTopology.MESSAGE> Motherboard topology has been mapped s
uccessfully
QD:Partition::qscu_train.USER> Setting up SCU hardware
QD:Partition::qscu_train.USER> Starting training process
QD:Partition::qscu_train.USER> Completing training process
QD:Partition::qscu_train.USER> Switching to idle bytes
QD:Partition::qscu_train.USER> Releasing Dircom units
QD:Partition::DiscoverTopology.MESSAGE> QCDOC is ready
QDaemonReturn 0
QKerReturn 0
AppReturn 0
```

More often than not, the topology of the machine is different from what the user wants to use. In this example, only direction 5 (W) is 2 while most applications has communications in only 4 or 5 dimensions, causing QCDOC to run 2 sets of identical programs. The necessary re-mapping of machine axes to logical axes is done by 'qpartition_remap'. You can map more than 1 machine axes to one logical axis. For example' qpartition_remap -X01 -Y23 -Z4 -T5' will remap machine axes 0 and 1 to logical X axis, 2 and 3 to Y, X to 4, T to 5, turning 2^6 machine into $4^2 \times 2^2$. (Warning: qpartition_remap fails if the user tries to combine a loop-backed machine axis (dimension 1) with another. In this example, we have $1^5 \times 2$ partition with N_W=2, remapped to N_X=2.

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% qpartition_remap -X5 -Y0
-Z1 -T2 -S3 -W4
executing... qpartition_remap -X5 -Y0 -Z1 -T2 -S3 -W4
CWD is /home/chulwoo/qos-doc-sessions/
QD:TextParser::doqpartition_remap.MESSAGE> Machine axis 5 -> App 1
QD:TextParser::doqpartition_remap.MESSAGE> Machine axis 0 -> App 2
QD:TextParser::doqpartition_remap.MESSAGE> Machine axis 1 -> App 3
QD:TextParser::doqpartition_remap.MESSAGE> Machine axis 2 -> App 0
QD:TextParser::doqpartition_remap.MESSAGE> Machine axis 3 -> App 4
QD:TextParser::doqpartition_remap.MESSAGE> Machine axis 4 -> App 5
QD:TextParser::doqpartition_remap.MESSAGE> Setting up PartitionCtl object
QD:TextParser::doqpartition_remap.MESSAGE> Calling the partition
QD:Partition::PrintState.MESSAGE> Boot kernel is NOT running
QD:Partition::PrintState.MESSAGE> Run kernel is running
QD:Partition::PrintState.MESSAGE> Serial communications are up
```

```
QD:Partition::PrintState.MESSAGE> Application axes are mapped
QD:Partition::qpartition_remap.MESSAGE> AppAxis 2 has Mach Axis 0 length 1
QD:Partition::qpartition_remap.MESSAGE> AppAxis 3 has Mach Axis 1 length 1
QD:Partition::qpartition_remap.MESSAGE> AppAxis 0 has Mach Axis 2 length 1
QD:Partition::qpartition_remap.MESSAGE> AppAxis 4 has Mach Axis 3 length 1
QD:Partition::qpartition_remap.MESSAGE> AppAxis 5 has Mach Axis 4 length 1
QD:Partition::qpartition_remap.MESSAGE> AppAxis 1 has Mach Axis 5 length 2
QD:Partition::qpartition_remap.MESSAGE> Folding AppDim 1 over 1 MachDims
QD:Partition::Fold.MESSAGE> AppDim 1 length is 2 and is periodic
QD:Partition::qpartition_remap.MESSAGE> Folding AppDim 2 over 1 MachDims
QD:Partition::Fold.MESSAGE> AppDim 2 length is 1 and is periodic
QD:Partition::qpartition_remap.MESSAGE> Folding AppDim 3 over 1 MachDims
QD:Partition::Fold.MESSAGE> AppDim 3 length is 1 and is periodic
QD:Partition::qpartition_remap.MESSAGE> Folding AppDim 0 over 1 MachDims
QD:Partition::Fold.MESSAGE> AppDim 0 length is 1 and is periodic
QD:Partition::qpartition_remap.MESSAGE> Folding AppDim 4 over 1 MachDims
QD:Partition::Fold.MESSAGE> AppDim 4 length is 1 and is periodic
QD:Partition::qpartition_remap.MESSAGE> Folding AppDim 5 over 1 MachDims
QD:Partition::Fold.MESSAGE> AppDim 5 length is 1 and is periodic
QD:Partition::qpartition_remap.MESSAGE> Application dimension 1 is length 2
QD:Partition::qpartition_remap.MESSAGE> Application dimension 2 is length 1
QD:Partition::qpartition_remap.MESSAGE> Application dimension 3 is length 1
QD:Partition::qpartition_remap.MESSAGE> Application dimension 0 is length 1
QD:Partition::qpartition_remap.MESSAGE> Application dimension 4 is length 1
QD:Partition::qpartition_remap.MESSAGE> Application dimension 5 is length 1
QD:Partition::qscu_train.USER> Setting up SCU hardware
QD:Partition::qscu_train.USER> Starting training process
QD:Partition::qscu_train.USER> Completing training process
QD:Partition::qscu_train.USER> Switching to idle bytes
QD:Partition::qscu_train.USER> Releasing Dircom units
QDaemonReturn 0
QKerReturn 0
AppReturn 0
```

Now we are all set to run the code compiled for QCDOC. 'qrun' command loads the binary into each node and run it. By default, output from the node 0 is echoed to the qcsh session.

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% qrun $QOS/quser/usertest/usertest
.x
executing... qrun /home/chulwoo/QCDOC/qos-2-0-stable/qos//quser/usertest/usert
est.x
CWD is /home/chulwoo/qos-doc-sessions/
```

```
QD:Partition::PrintState.MESSAGE> Boot kernel is NOT running
QD:Partition::PrintState.MESSAGE> Run kernel is running
QD:Partition::PrintState.MESSAGE> Serial communications are up
QD:Partition::PrintState.MESSAGE> Application axes are mapped
QD:Partition::qrun.MESSAGE> Cmd: /home/chulwoo/QCDOC/qos-2-0-stable/qos//quser
/usertest/usertest.x
QD:RkerMgr::LoadElfFile.MESSAGE> Building packets
QD:RkerMgr::LoadElfFile.MESSAGE> Processing file /home/chulwoo/QCDOC/qos-2-0-s
table/qos//quser/usertest/usertest.x
QD:RkerMgr::Elf2Rpc.MESSAGE> file open succeeded /home/chulwoo/QCDOC/qos-2-0-s
table/qos//quser/usertest/usertest.x
QD:RkerMgr::Elf2Rpc.MESSAGE> 2 P-sections
QD:RkerMgr::LoadElfFile.MESSAGE> Sending 208 packets to 2 nodes
QD:RkerMgr::BroadCastPktList.MESSAGE> Thread 0 has 2 nodes
QD:RkerMgr::LoadElfFile.MESSAGE> Loaded binary
QD:Partition::qrun.MESSAGE> Sending /home/chulwoo/QCDOC/qos-2-0-stable/qos//qu
ser/usertest/usertest.x
QD:RkerMgr::CmdArgs.MESSAGE> Stuffing Command line arguments: 2 pkts length 67
QD:RkerMgr::BroadCastPktList.MESSAGE> Thread 0 has 2 nodes
QD:RkerMgr::Exec.MESSAGE> Setting return addr as 192.168.1.15:-21642
QD:RkerMgr::BroadCastPktList.MESSAGE> Thread 0 has 2 nodes
QD:Partition::qrun.MESSAGE> Loaded /home/chulwoo/QCDOC/qos-2-0-stable/qos//qus
er/usertest/usertest.x successfully
Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 0 : Global ctor called
Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 0 : Hello, world from QCDOC userland
Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 0 : 1 Command line args are
Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 0 :  0: /home/chulwoo/QCDOC/qos-2-0-stable/qos//qu
ser/usertest/usertest.x
QD:RkerMgr::AppExit.MESSAGE> Node Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 1 exited with app exit cod
QD:RkerMgr::AppExit.MESSAGE> Node Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 1 exited with rk
er exit code 0
QD:RkerMgr::AppExit.MESSAGE> Normal exit
QD:RkerMgr::AppExit.MESSAGE> Node Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 0 exited
with app exit code 0
QD:RkerMgr::AppExit.MESSAGE> Node Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 0 exited
with rker exit code 0
QD:RkerMgr::AppExit.MESSAGE> Normal exit
QD:RkerMgr::ThreadMain.MESSAGE> Exiting server thread
QD:RkerMgr::ApplicationFinish.MESSAGE> Retrieving status for Node Cb 78 Bp 0 Mbd 0 Dbd 0
QD:RkerMgr::ApplicationFinish.MESSAGE> Retrieving status for Node Cb 78 Bp 0 Mbd 0 Dbd 0
QD:RkerMgr::ApplicationFinish.MESSAGE> FIXME: sending back success regardless
QD:RkerMgr::ApplicationFinish.MESSAGE> Should I perform or of all nodes exit codes?
```

```
QDaemonReturn 0
QKerReturn 0
AppReturn 0
```

If you suspect your program is hanging, you can stop the program (equivalent of ctrl-C) by typing 'qkill' command (**??**) in **qdaemon window**. In fact, all the qcsh commands listed above, except qinit and qdetach which is not needed, can be run from qdaemon window too. Using qcsh does provide helpful features such as command history and scripting capability, so is more convenient for users.

User can check UNIX stdout, stderr and 'kernel' output, messages from QCDOC run kernel, after qrun by using 'qnodes_print' command.

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% qnodes_print -b stdout
executing... qnodes_print -b stdout
CWD is /home/chulwoo/qos-doc-sessions/
QD:Partition::PrintState.MESSAGE> Boot kernel is NOT running
QD:Partition::PrintState.MESSAGE> Run kernel is running
QD:Partition::PrintState.MESSAGE> Serial communications are up
QD:Partition::PrintState.MESSAGE> Application axes are mapped
QD:RkerMgr::ReadPrintBuf.USER> Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 0
Global ctor called
Hello, world from QCDOC userland
1 Command line args are
0: /home/chulwoo/QCDOC/qos-2-0-stable/qos//quser/usertest/usertest.x
QD:RkerMgr::ReadPrintBuf.USER> Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 1
Global ctor called
Hello, world from QCDOC userland
1 Command line args are
0: /home/chulwoo/QCDOC/qos-2-0-stable/qos//quser/usertest/usertest.x
QDaemonReturn 0
QKerReturn 0
AppReturn 0

(qcdochost0:/home/chulwoo/qos-doc-sessions)% qnodes_print -b stderr
executing... qnodes_print -b stderr
CWD is /home/chulwoo/qos-doc-sessions/
QD:Partition::PrintState.MESSAGE> Boot kernel is NOT running
QD:Partition::PrintState.MESSAGE> Run kernel is running
QD:Partition::PrintState.MESSAGE> Serial communications are up
QD:Partition::PrintState.MESSAGE> Application axes are mapped
QD:RkerMgr::ReadPrintBuf.USER> Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 0
QD:RkerMgr::ReadPrintBuf.USER> Cb 78 Bp 0 Mbd 0 Dbd 0 Asic 1
QDaemonReturn 0
```

```
QKerReturn 0
AppReturn 0

(qcdochost0:/home/chulwoo/qos-doc-sessions)% qnodes_print -b kernel
executing... qnodes_print -b kernel
```

Output is omitted here since it's too long! One of the occasions kernel output is very helpful is when the user program exited with errors such as SIG_SEGV. In this case, the contents of the registers when program generated errors will be printed to kernel output 2 specially relevant registers in this case are SRR0, which holds the address of the faulty instruction, and LR which shows the return address of the function which called the faulty routine.

The user can disconnect from the partition by using 'qdetach' command.

```
(qcdochost0:/home/chulwoo/qos-doc-sessions)% qdetach
disconnected
```

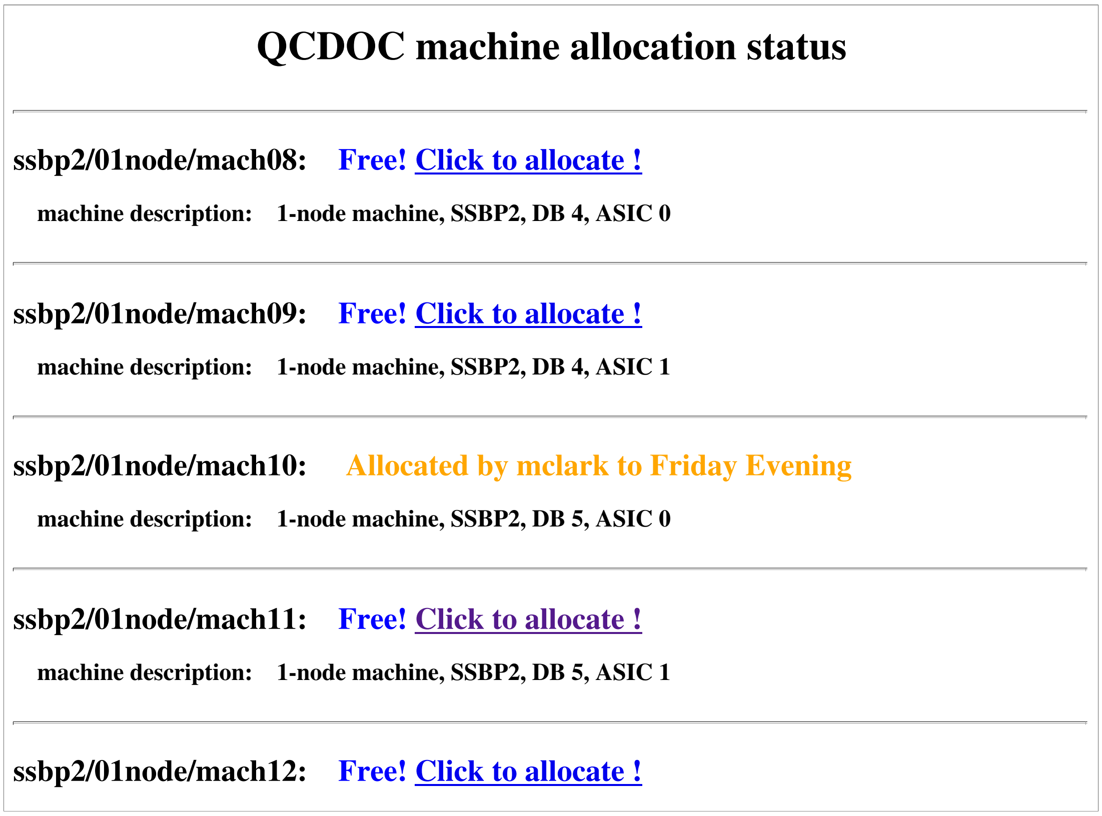

Figure 2.1: Screen capture of QCDOC allocation webpage (https://qcdoc.phys.columbia.edu)

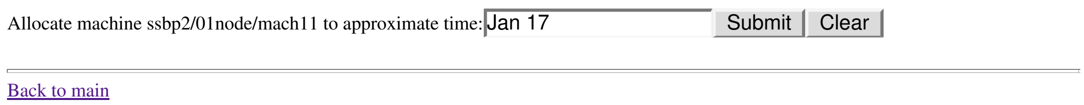

Figure 2.2: QCDOC allocation page

Chapter 3

Description of QCDOC OS routines

This chapter includes descriptions of routines available from QOS.

3.1 File I/O

QCDOC OS provides for NFS access from *all* nodes to a disk attached to the front end machine. However, only limited part (/space) is available, to guard against possible (very unlikely) OS error.

IMPORTANT: The mount point for QCDOC nodes and front end are different. While it is /host from QCDOC, /space is the mount point from the front end. For example, compile and running

```
#include <stdio.h>

int main (int argc, char ** argv){
  FILE *fp;
  char filename[200];
  sprintf(filename,"/host/chulwoo/test.out_%d%d%d%d",CoorX(),CoorY(),CoorZ(),CoorT());
  fp = fopen(filename,"w");
  fprintf(fp,"testing QCDOC OS\n");
  fclose(fp);
}
```

on QCDOC will generate a file /space/chulwoo/test.out_(0000,0001, ...) with

```
testing QCDOC OS
```

Since each QCDOC nodes can write to a file concurrently, users need to differentiate the filenames for each node. Size{XYZTSW}() (Section 3.2) can be useful for this.

3.2 sysfunc.h

Many of the functions available for QCDSP are also available for QCDOC. Descriptions for some of available functions are below.

- Size{X,Y,Z,T,S,W}(): returns the number of nodes in each dimension of n-dimensional mesh.
- Coor{X,Y,Z,T,S,W}(): returns the coordinate of the node calling the routine.

3.3 Memory allocation

QCDOC has 2 different kinds of RAM (Embedded DRAM(EDRAM) and external DDR) with 3 different caching behavior (non-caching,transient,normal).

While "normal" memory uses all 32kB processor cache, "Transient" memory uses only part (1kB) of it and "non-caching" memory does not use cache at all.

The allocation/deallocation from these different sections are done through qalloc(), qfree() function.

```
void * qalloc (int flags, size_t bytes)
enum {
  QNONCACHE = 0x01,
  QCOMMS    = 0x02,
  QFAST     = 0x04,
};
```

qalloc() allocates memory and returns the address. QCOMMS flag is for transient memory and QFAST flag is for EDRAM. For example, allocating 100 bytes in a transient EDRAM would be qalloc(QCOMMS—QFAST,100).

```
void qfree (void *par)
```

Deallocates allocated memory.

```
int qalloc_is_communicable(void *ptr);
int qalloc_is_noncache(void *ptr);
int qalloc_is_normal(void *ptr);
int qalloc_is_fast(void *ptr);
```

returns 1 if the memory pointed by ptr satisfies the condition.

While variable or data used frequently in a function should be in a normal memory, transient or non-caching memory can be optimal for communication since SCU memory access does not refer cache and user is responsible for the contents of the memory is consistent with the cache. In most cases, using transient memory for incoming data from SCU and noncaching one for outgoing data is optimal, since the memory bandwidth is larger from transient memory.

3.4 Communication calls

Communication routines for QCDOC is similar to those of QCDSP and the Class SCUDirArg is backward-compatible. A class SCUDirArgIR is added to take advantage of the hardware instruction registers(IR) in QCDOC.

```
class SCUDirArgIR
{
  public:

    SCUDirArgIR(){initted=FALSE;};
    SCUDirArgIR( void * addr, SCUDir dir, SCUXR sxr, int bsize,
      int nblk = 1 , int s  = 0, int sram_reg = UND, int pri= -1);
    SCUDirArgIR(SCUDir dir, SCUXR sxr, SCUDMAInst **, int n, int sreg
= UND, int pri= -1 );
    ~SCUDirArgIR();

    void Init( void * addr, SCUDir dir, SCUXR xr, int bsize,
      int nblk  = 1, int s = 0, int sram_reg = UND, int pri= -1);

    void Init(SCUDir dir, SCUXR sxr, SCUDMAInst **, int n,
      int sreg = UND , int pri= -1);
```

SCUDirArgIR needs instruction parameters (starting address, block, strides), direction, data direction (send/receive). If you do not specify the position of instruction register in which the first data parameters are stored, it will be automatically assigned and managed by the instruction manager (SCUIRMng). However, for multi-wire transfers(class SCUDirArg Multi), all the SCUDirArg in single multi-wire transfer should have the same starting IR's. sram_reg takes IR_0, IR_1 $\cdots$ IR_15, UND (for undefined/don't care) as input.

Instruction parameters can be directly set for a single block/stride transfer. For multiple block/strides, class DMASCUInst is needed

```
class SCUDMAInst{

  public:

    SCUDMAInst() {};
    SCUDMAInst(const void * a, int bsize, int nblk = 1, int s = 0);

    void Init(const void * a, int bsize, int nblk = 1, int s = 0);

    void *Addr(){return addr;};
    void *Addr(void * a);
```

```
    int Blklen(){return blklen;};
    int Numblk(){return numblk;};
    int Stride(){return stride;};
};
```

```
ClassSCUDirArgIR{

    void Addr( void ** a, int n);
    void Addr( void * a);
```

The starting address of each instruction can be modified after the class is initialized by using Addr() function.

```
    void SlowStartTrans();
    void StartTrans();
    void Assert();
    void TransComplete(int delay );
    int IsComplete();
}
```

Similar to QCDSP, StartTrans() starts the transfer and TransComplete() waits until the transfer is finished. IsComplete() returns 1 if the transfer has been completed and 0 if not. SlowStartTrans() flushes the data buffer before the data transfer to ensure the cache coherency.

A typical initialization and execution of SCU transfer might look like this:

```
{
  const int buf_len = 128;
  double *send_data = qalloc(QCOMMS|QFAST,sizeof(double)*buf_len);
  double *recv_data = qalloc(QCOMMS|QFAST,sizeof(double)*buf_len);

  SCUDMAInst *send_dma = new SCUDMAInst;
  send_dma->Init(send_data,sizeof(double)*buf_len);
  SCUDMAInst *recv_dma = new SCUDMAInst;
  recv_dma->Init(recv_data,sizeof(double)*buf_len);


  SCUDirArgIR *scu_arg[2];
  scu_arg[0] = new SCUDirArgIR;
  scu_arg[0]->Init(SCU_XP,SCU_SEND,&send_dma,1,IR_5);
```

```
  scu_arg[1] = new SCUDirArgIR;
  scu_arg[1]->Init(SCU_XP,SCU_SEND,&send_dma,1,IR_5);

  SCUDirArg Multi multi;
  multi.Init(scu_arg,2);
  multi.StartTrans();
  multi.TransComplete();

  delete scu_arg[0];
  delete scu_arg[1];
  delete send_dma;
  delete recv_dma;

}
```