ALMOS distribution for the TSAR many-core

1. 1. Summary
   2. Compatibility
   3. Setup
   4. Start the simulation
   5. Your first application
   6. Running without the interactive mode

Summary

Running ALMOS on the TSAR virtual prototype requires the installation of several open-source technologies like a GCC cross-compiler for Mips (el), ?SocLib virtual prototyping library, ?TSAR related components, and ?SystemCASS. To simplify the task of building and configuring a correct development environment, ALMOS comes with a stand-alone and ready-to-use distribution.

Mainly, this distribution enables you to:

- Port your own applications and libraries to ALMOS.
- Run these applications on TSAR using several configurations ranging from 4 to 1024 cores.
- Analyse the performance of your applications or the totality of the software-stack on a large-scale many-core.

Upon your needs you can also use this distribution to:

- Validate and evaluate any kernel new features or updates.
- Experiment and develop new parallel programming libraries and run-times for a large-scale single-chip many-core.
- Consolidate/build your own educational materials in the field of operating systems and parallel programming.
- Validate and evaluate any hardware evolution/development in a TSAR based architecture.

Compatibility

The distribution package contains some binary executables (e.g. GCC cross-compiler, TSAR simulator). These programs can run on a Linux based distribution. They were tested on Ubuntu 10.04/12.04 and on Scientific Linux 6.2. Both 64 and 32 bits host machines are supported.

Setup

Download the latest stable distribution from this link. Decompress the .tbz2 file:

```
$ tar jxf almos-tsar-mipsel.tbz2
```

Now you have a sub-directory named *almos-tsar-mipsel*. In the reset of this page, we refer to the absolute path to this directory as *DISTRIB*.

Start the simulation

Before **any use** of the distribution package you need to set some environment variables:

```
$ cd DISTRIB
$ source ./SourceMe
```

Note: by default you have to source this file from its local directory, that is, you have to be in the *DISTRIB* directory.

Now, lets go to *DISTRIB/test/pf1* and run *make sim1*:

```
$ cd DISTRIB/test/pf1
$ make sim1
```

This will launch the TSAR full-system simulator with its 4 tty terminals and one frame-buffer window. You will end by having a shell prompt on the tty1.

That is it ... you are done !!

The tty0 and tty3 are reserved for the kernel trace and information messages. For a user application, the tty1 represent the stdin and stdout while the tty2 represent the stderr.

Sim1 rule means the simulator of TSAR is configured to one cluster, that is, 4 cores. Using *sim4*, *sim16*, *sim64* or *sim128* lets you start the simulator with (respectively) 4, 16, 64 or 128 clusters of 4 cores each.

Note: although the hardware configuration can be changed at each simulation, there is no need to recompile or regenerate the kernel. The kernel of ALMOS detects the hardware resources at each boot. A user application can get the number of online cores using the standard *sysconf()* call (man sysconf).

Now lets take a look inside the *DISTRIB/test/pf1* directory.

```
$ ls DISTRIB/test/pf1
arch-info.bin bootloader.bin hdd-img.bin kernel-soclib.bin Makefile
```

After each *make simN* command (where N is 1, 4, 16, 64 or 128) you will have at least 4 .bin files. These bins are required by the simulator. In the TSAR simulated platform, there is a ROM component and a H.D.D controller. The ROM will be filled with the contents of *arch-info.bin*, *bootloader.bin* and *kernel-soclib.bin* before starting the simulation. The *hdd-img.bin* is a FAT32 file system image sought by the H.D.D controller at each sector transfer.

The *kernel-soclib.bin* file is the ALMOS kernel while the *bootloader.bin* file is the ALMOS boot-loader for TSAR. The *arch-info.bin* file contains the description of the TSAR hardware resources. This file is regenerated (copied from *DISTRIB/test/arch-bins*) at each *make simN* command.

You can take a look to *DISTRIB/test/pf1/Makefile*. This file lets you customize some parameters like the used ALMOS kernel revision and the number of simulator threads. If you want to run simultaneously several configurations, lets say 4, you can create 3 additional *DISTRIB/test/pf[2-4]* directories each of which with its own customized Makefile. Than in each of these *pf* directories you can type your *make simN* command.

Note: by default, the location from where the corresponding kernel revision is looked for is *DISTRIB/test/kern-bins*. The number of simulator's threads should not exceed the number of the physical CPUs (not the logical one) of your host machine. Finally, the hdd-imge.bin is a symbolic link to *DISTRIB/test/misc/hdd-img.bin* so be careful to provide an *hdd-img.bin* regular file per *pf* directory when you run simultaneously a multiple instances of the simulator.

Your first application

Lets change the cap and look to how we can compile and run a user application for ALMOS/TSAR target. This task can be resumed in 4 steps:

1. Write your application
2. Cross-compile it using ALMOS headers and libraries
3. Install it on the hdd-img.bin file system image
4. Go to your *DISTRIB/test/pfX* directory and run the TSAR simulator

Some example of applications source code can be found in *DISTRIB/apps*. Lets try the *hello_world* one. The *main()* function of this application asks the system for the number of online cores (*count*) and then it fires *count* threads before it synchronizes on their end. Each fired thread executes the *thread_func()* function which prints a "Hello World" message.

To Cross-compile the application using ALMOS headers and static libraries:

```
$ cd DISTRIB/apps/hello_world
$ make TARGET=tsar
```

The result is the *./hello* program for ALMOS. To install this program into ALMOS's file system (*hdd-img.bin*). Just type:

```
$ make install
```

This will copy the *hello* program located in your current directory on your host machine (Linux) to */bin/hello* inside the *hdd-img.bin* file system image.

Now, lets go to the *DISTRIB/test/pf1* directory and launch the simulator:

```
$ cd ''DISTRIB/test/pf1''
$ make sim1
```

Wait until you get a prompt on the *tty1* console, then type:

```
[/HOME/ROOT]>ls /bin
```

The contents of */bin* directory will be listed and the *hello* program must figure in the list. To run */bin/hello* program, type the following command on the *tty1* console:

```
[/HOME/ROOT]>exec /bin/hello
```

You can see some kernel log messages on the *tty0* and *tty3* until you see the output of your *hello* program arrives on the *tty1*.

That is it ... you are done !!

Note: the kernel of ALMOS takes its own dynamic decisions about the management of the hardware resources including threads placement and cores load-balancing. Nevertheless, you can explicitly place the new process on a given core by using the *-p core_number* of the *exec* command (e.g. *exec -p 3 /bin/hello*). This concerns only the first thread which executes the *main()* function. If you want to control the placement of the next threads, you can use the *pthread_attr_setcpuid_np()* function to set the core number on which the new thread will be pinned (e.g. *DISTRIB/apps/bcv/main.c*).

In a normal situation your application may have some bugs and you need to validate it. The recommended and most convenient method is to test and validate your application on Linux before corss-compiling it for ALMOS. Lets go back to *DISTRIB/apps/hello_world* and clean the directory before compiling the same application for Linux:

```
$ cd ''DISTRIB/apps/hello_world''
$ make realclean
$ make TARGET=linux
```

This will generate the *./hello* program for Linux. To run it, just type:

```
$ ./hello
```

Now, lets take a look to the Makefile:

```
$ cat ./Makefile

FILES=main
BIN=hello
ADD-CFLAGS=-O3

HDD=$(ALMOS_TEST)/pf1/hdd-img.bin

include $(ALMOS_TOP)/include/appli.mk

install:
        mcopy -i $(HDD) $(BIN) ::bin/.
```

The *ALMOS_TEST* and *ALMOS_TOP* are an environment variables exported by sourcing the *DISTRIB/SourceMe* file from the *DISTRIB* directory. The fist one is equivalent to *DISTRIB/test* directory while the second is equivalent to *DISTRIB/almos*.

The *mcopy* command enables you to copy a file from your host machine file system to a FAT file system image and in particular to the *hdd-image.bin*. *mdir* and *mdel* are another two useful utilities from the *mtools* package (see man mtools).

Running without the interactive mode

As you may certainly noticed, the simulation speed is slow. This because the simulator is cycle and bit accurate, that is, the simulation includes the evolution of all hardware FSMs (Finite State Machine), data movements and contentions on all hardware resources at a cycle and bit precision.

For a large TSAR configuration starting from *sim16* (i.e. 16 x 4 = 64 cores) or even if you just want to launch several instances of the simulator with different configurations, you need to disable the interactive mode in favor of a batch execution mode. This can be done by following 2 steps before running a simulation:

1. Redirecting the simulator ttys output to files
2. Write a mini-script to let the shell automatically executes your programs

To do the first step you have to set an environment variable *SOCLIB_TTY* as fallows:

```
$ export SOCLIB_TTY=FILES
```

Now, lets go to *DISTRIB/test/pf1* directory and edit a new *shrc* script with the following contents:

```
echo started
echo exec /bin/hello
exec /bin/hello
echo ended
```

This can be done as follows:

```
$ cd DISTRIB/test/pf1
$ cat > ./shrc
echo started
echo going to exec /bin/hello
exec /bin/hello
echo ended
^D
```

Lets show the *./shrc* contents before coping it to */etc/shrc* on ALMOS file system image *hdd-img.bin*:

```
$ cat ./shrc
echo started
echo going to exec /bin/hello
exec /bin/hello
echo ended
$ mcopy -i hdd-img.bin ./shrc ::/etc/.
```

Note: the *shrc* file name and the */etc* destination directory are required by the shell of ALMOS. The first user process executed by the kernel of ALMOS is */bin/init* which forks and execs the shell */bin/sh*. The *sh* will systematically look for */etc/shrc* configuration file and executes all of its commands.

Now, we are ready to run the TSAR simulator for a while before killing it and inspecting the generated files:

```
$ make sim1
^C
```

That is it ... you are done !!

Lets list the contents of *pf1* directory:

```
$ ls
arch-info.bin  bootloader.bin  hdd-img.bin  kernel-soclib.bin  Makefile  shrc  tty0  tty1  tty2
```

As you can see, there is *tty[0-3]* new files. They are the result of the redirection of simulator's ttys output into these files.

```
$ cat tty0
init: System Revision Info [Almos almix-v1.2 2011 tsar mipsel mips32]
init: Listening to signals 15, 1
init: Connexion-Shell has been created [pid 2]
started
going to exec /bin/hello
ended
[/HOME/ROOT]>pid 3, tid 0, arg 0: Hello World !!
pid 3, tid 2, arg 1: Hello World !!
pid 3, tid 3, arg 2: Hello World !!
Main: thread #0 has finished with return value 0
Main: thread #1 has finished with return value 1
pid 3, tid 4, arg 3: Hello World !!
Main: thread #2 has finished with return value 2
Main: thread #3 has finished with return value 3
```