WaveScalar Simulator Tutorial

Version 1.0
Revised Jan 26, 2006
Please send comments and questions to aputnam@cs.washington.edu

Install

1. Download ws_workloads.tar.gz and copy it into your home directory on your CSE 548 DEC Alpha account on jana.cs.washington.edu.

2. Unzip and untar the ws_workloads file:
    bash$ **gunzip ws_workloads.tar.gz**
    bash$ **tar -xf ws_workloads.tar**

3. Repeat the process on your regular department Unix account. You will need both accounts.

4. Download the ws_simulators.tar.gz file and copy it into your home directory on your department Unix machines.
    bash$ **gunzip ws_simulators.tar.gz**
    bash$ **tar -xf ws_simulators.tar**

5. This contains two simulators, *surfer*, and *kahuna*. Read the readme.txt file in the ws_simulators directory to get more info about the simulators and how to run them.

Compiling

1. Log onto your class account on jana.cs.washington.edu

2. Change to your ws_workloads directory

3. Source the 'setup' file (be sure to hit the '.' before './setup')
    bash$ **. ./setup**

4. Change to the Workloads directory
    bash$ **cd workloads**

5. Change to the simple directory
    bash$ **cd simple**

6. You should see 3 files in that directory:
    multiFunc13.c -- Source code
    multiFunc13.missingFuncs -- A list of certain Alpha library functions needed for simulation
    Makefile -- The makefile

7. Take a look at the multiFunc13.c file. You'll notice that there are 5 methods:
    main()

add()
add2()
add3()
add4()

These functions don't do anything in particular. They're simply random example functions.

8. Create a file called multiFunc13.compile. This file tells the compiler which functions you want to run as WaveScalar code and which functions you want to just run as native Alpha code. It also tells the compiler which function is main(). Make the file to look like:

```
main compile isMain
add compile
add2 compile
add3 compile
add4 compile
```

Figure 1 – multiFunc13.compile file

By putting *compile* after each function, we tell the compiler that we want those functions to be made into WaveScalar functions. We use the keyword *stub* to indicate that the function should just run in native Alpha mode. You should only use *stub* on your code if you have lots of functions that are irrelevant to the performance of your program, or when your code calls library routines (discussed in Step 24).

9. Create a file called multiFunc13.spec. This file has one line that tells the compiler what command line to use when running the program. The compiler automatically runs *gprof* to give you a profile of the program, so be sure to use the correct command line. The multiFunc13.c benchmark doesn't use any command line arguments, so the file is simply:

```
CMDLINE=""
```

Figure 2 – multiFunc13.spec file

10. Open the multiFunc13.missingFuncs file. These are Alpha library routines that commonly show up in our benchmarks. Hopefully this will be a complete list, and you will not have to edit this list. You should copy this and just rename when you compile your own benchmark.

11. Open your Makefile. The file should look like:

```
THREAD_LIB_DIR=$(LC_DEVEL_ROOT)/src/lib/threads
NAME=multiFunc13
SOURCES = multiFunc13.c

include ../Make.rules
```

Figure 3 – Makefile for the multiFunc13 benchmark

You will have to change the NAME and SOURCES lines to match your application.

12. At this point, you should have 4 files in your directory:
    multiFunc13.c -- Source code
    multiFunc13.compile -- Tells what to run as WaveScalar and what to run as Alpha code
    multiFunc13.missingFuncs -- A list of certain Alpha library functions needed for simulation
    Makefile -- The makefile

13. Run gmake to compile the Alpha binaries
    bash$ **gmake**

14. Make sure that the build completed without errors. If there were errors, go back and make sure that you have performed Step 3.

15. Your directory should now have the following files:

Makefile	multiFunc13.exe	multiFunc13.keyfuncs
junk	multiFunc13.gexe	multiFunc13.map
multiFunc13.axp	multiFunc13.gmon	multiFunc13.missingFuncs
multiFunc13.c	multiFunc13.go	multiFunc13.nm
multiFunc13.compile	multiFunc13.gprof	multiFunc13.o

Figure 4 - Files in the ~/ws_workloads/workloads/simple directory

The multiFunc13.gprof file is the output of gprof from profiling your application. This should match your results (roughly) as long as you remembered to set the command line correctly in step 9.

16. Copy the *.c, *.compile, *.exe, *.map, *.keyfuncs, and *.missingFuncs files to your account on the department Linux machines (barb, recycle, bicycle, your desktop Linux machine, etc…). The files should go in the same directory as they were on jana.
    bash$ **scp Makefile *.c *.compile *.exe *.map *.keyfuncs *.missingFuncs *.nm bicycle:~/ws_workloads/workloads/simple**

17. Make sure that you're running the bash shell. Just to make sure, start a new bash shell anyway.
    bicycle% **/bin/bash**

18. Change to the base WaveScalar directory
    bicycle% **cd ~/ws_workloads/**

19. Source the 'setup' file (be sure to hit the '.' before './setup')
    bash$ **. ./setup**

20. Change to the directory with your code
    bicycle% **cd ~/ws_workloads/workloads/simple**

21. Compile the code. Note: you use *make* here, NOT *gmake* like you do on jana.
    bicycle% **make**

22. Once again, the most common reason for an error is forgetting to do Step 19. Make sure that you have the multiFunc13.ws file. This file, along with multiFunc13.exe, are the only files needed for the simulator.

23. Now we can run the simulator(s). It may be a good idea to copy the .exe and .ws files to a different directory to run them. I suggest the following:

bicycle% **mkdir ~/ws_simulators/simple**
bicycle% **cp multiFunc13.ws multiFunc13.exe ~/ws_simulators/simple**
bicycle% **../surfer -- multiFunc13.ws multiFunc13.exe**

24. You'll notice that the simulator will end with the lines:

STATISTICS DONE
Regression FAILED: SIGABRT

This means that your program has a problem. Scroll up to the error. In this case (and in may cases when you're trying out a new program), you will have an error similar to the following:

CRASH[/nfsroot/4/swanson/lightCycle/lc-devel/src/surfer/WaveCache/AbstractProcessingElement.cxx:230]: 370 Sending to missing function @ 0x120009300

When you have an error about a "Missing Function", you need to look at the number. In this case, it is 120009300. This is the key to finding the missing function. (Yes, this is annoying, but you only have to chase these missing functions down once, so it's not that bad.)

25. Go back to the directory where you compiled your code. Find the file called multiFunc13.nm. This file maps function names to addresses. Grep for 120009300. (Note: don't have the leading 0x since that will make the grep fail).

bash$ **cd ~/ws_workloads/workloads/simple**
bash$ **grep 120009300 multiFunc13.nm**

The grep results in the following:

_OtsMove | 0x00000120009300 | T | 0x00000000000008

The function is one of the Alpha library functions. You need to add this to your .compile file as a stub.

26. Open your .compile file, and add the following line to the end of the file:

_OtsMove stub

27. Repeat Steps 17 through 21 until the simulator completes successfully. In the end, you will also have to add the following lines:

fprintf stub
__cf_fprintf stub

If you do another application (or if your application uses fprintf), then just copy these into your .compile file from the start). Remember, fprintf is different from printf, so these aren't the right lines for printf( ).

28. When the simulator runs successfully, the output will end with:

STATISTICS DONE
Regression PASSED

29. Now we want to get statistics from the simulators. Run the following:

bicycle% **../surfer -- multiFunc13.ws multiFunc13.exe >& surfer.out**
bicycle% **../kahuna -- multiFunc13.ws multiFunc13.exe >& kahuna.out**
bicycle% **../kahuna -sim/rows 1 -sim/rows 1 -- multiFunc13.ws multiFunc13.exe >& k1x1.out**

The output will be stored in surfer.out, kahuna.out, and k1x1.out respectively. As described in the readme.txt file in the ws_simulators directory, these three commands correspond to:

1) Surfer – A functional simulation whose statistics are "perfect case" statistics
2) Kahuna – Default cycle-accurate simulation of an 8x8 (64 cluster) WaveScalar processor
3) Kahuna – Parameterized cycle-accurate simulation of an 1x1 (1 cluster) processor

30. Look in the output for the statistics sim/ipc and sim/aipc. These show how many instructions per cycle WaveScalar was able to execute. This equates to performance. (Note that there will be two entries for each stat. The first is either 0 or NaN, and indicates the status of the statistic before running. You can ignore these). The IPC number is the raw number of WaveScalar instructions executed. Since WaveScalar (and dataflow machines in general) tend to require more instructions than a von Neumann machine, the AIPC number represents the Alpha-equivalent IPC. This can be directly compared to the performance on a traditional von Neumann machine like the DEC Alpha.

31. Since this is a small program, you won't notice much difference in the results between kahuna.out and k1x1.out. Try using the lcs.exe and lcs.ws files provided in ws_simulators/lcs_example to see the effect of the number of clusters on performance for larger applications. Read the readme.txt file in ws_simulators/lcs_example for more info about running the lcs example.