NB01-hello-world

August 31, 2022

1 Hello World

1.0.1 Lesson Objectives

Upon completing this notebook you should be able to understand and apply the following concepts:

- How to set up your environment to use the Lucata toolchain to compile code
- Understand the different Lucata tools including *emu-cc* and *emusim.x*.
- Be able to run a simple Hello World script that spawns Emu threads and then syncs the result.
- Run a simulation with timing that generates statistics.
  - Compare some basic statistics for a naive and "Lucata-aware" memory layout.

1.0.2 Environment Setup

We first need to initialize our environment to use the Lucata toolchain. This toolchain allows you to compile Cilk code with x86, the Lucata simulator, and for hardware execution. Note that this notebook should load the toolchain using the included .env file, so this is just if you wanted to compile code on the command line.

```
In [1]: !set -x;. /tools/emu/pathfinder-sw/set-lucata-env.sh; set +x
```

```
+ . /tools/emu/pathfinder-sw/set-lucata-env.sh
+ LUCATA_TOOLCHAIN_BASE=/tools/emu/pathfinder-sw
+ LUCATA_VERSION=22.02
+ export LUCATA_TOOLCHAIN_DIR=/tools/emu/pathfinder-sw/22.02
+ export PATH=/tools/emu/pathfinder-sw/22.02/bin:/nethome/jyoung9/.cargo/bin:/opt/slurm/current
+ export LD_LIBRARY_PATH=/tools/emu/pathfinder-sw/22.02/lib:
+ export LD_LIBRARY_PATH=/tools/emu/pathfinder-sw/22.02/lib::/usr/lib/x86_64-linux-gnu/
+ echo Lucata tools are added to current path from /tools/emu/pathfinder-sw/22.02
Lucata tools are added to current path from /tools/emu/pathfinder-sw/22.02
+ set +x
```

For this and other notebooks, we will import the following environment variables - a pointer to the user's notebook code director and a pointer to the Lucata tools

```
In [2]: import os

        #Get the path to where all code samples are
        os.environ["USER_NOTEBOOK_CODE"]=os.path.dirname(os.getcwd())
        os.environ["PATH"]=os.pathsep.join(["/tools/emu/pathfinder-sw/22.02/bin",os.environ["PA
        os.environ["FLAGS"]="-I/tools/lucata/pathfinder-sw/22.02/include/memoryweb/ -L/tools/lu

        !echo $USER_NOTEBOOK_CODE
        #Print out which Emu compiler, emu-cc, we are using
        !which emu-cc
        #Print out the compiler flags we need to use the Lucata memoryweb headers and library
        !echo "Lucata compilation flags are $FLAGS"

/nethome/jyoung9/tmp/lucata-pathfinder-tutorial/code
/tools/emu/pathfinder-sw/22.02/bin/emu-cc
Lucata compilation flags are -I/tools/lucata/pathfinder-sw/22.02/include/memoryweb/ -L/tools/lu
```

1.1 Code Example 1 - Naive Hello World

Here is a "Hello, world" example to start showing aspects of writing for the Emu. However, your first question might be related to the use of the mw_malloc1dlong array with a distributed system.

Where does `ptr` itself live? Does computing `ptr[k]` cause a migration?

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cilk.h>

// These are Emu-specific.
#include <memoryweb.h>
#include <timing.h>

static const char str[] = "Hello, world!";

long * ptr;
char * str_out;

int main (void)
{
     // long is the reliable word length, 64-bits.
     const long n = strlen (str) + 1;

     ptr = mw_malloc1dlong (n); // striped across the nodelets
     str_out = malloc (n * sizeof (char))); // entirely on the first nodelet

     starttiming(); // For the simulator.  Start gathering stats here.
```

```
    for (long k = 0; k < n; ++k)
        ptr[k] = (long)str[k]; // Remote writes

    for (long k = 0; k < n; ++k)
        str_out[k] = (char)ptr[k]; // Migration and remote write...

    printf("%s\n", str_out);  // Migration back
}
```

1.1.1 Compilation and simulation for the Pathfinder

We'll test compiling this example to show the syntax and then move on to a more optimized example. Note that the .mwx output can be used for simulation and execution on the Pathfinder system.

We use `emu-cc` to compiler and `emusim.x` to run a System-C simulation of the application running on a Pathfinder with the specified memory and node paramters.

It is also important to understand the following details:
* We defined $FLAGS up above to include the Lucata headers and libraries
* emusim.x takes a few parameters including the memory size (-m 21 for 1 MB) and the number of nodes to simulate (--total_nodes). You can try changing the memory size (21 to 28) or number of nodes (1 to 8) to see how the simulation changes.

```
In [3]: %%bash
        emu-cc -o hello-world-naive.mwx $FLAGS hello-world-naive.c
        ls *.mwx
```

```
hello-world-naive.mwx
```

```
In [4]: %%bash
        #Run a basic simulation with memory size 2^21, one node, and the naive Hello World exe
        emusim.x -m 21 --total_nodes 1 -- hello-world-naive.mwx
```

```
Start untimed simulation with local date and time= Wed Aug 31 14:10:59 2022

Hello, world!
End untimed simulation with local date and time= Wed Aug 31 14:10:59 2022
```

```
        SystemC 2.3.3-Accellera --- Feb 22 2022 09:27:12
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
```

1.2 Code Example 2 - Hello World with Replication

With the Lucata architecture, we often want to avoid spurious migrations by replicating data across nodes so that each node has a copy of the relevant data it needs. This improved sample in `hello-world/hello-world.c`, demonstrates the usage of the `replicated` type:

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cilk.h>

// These are Emu-specific.
#include <memoryweb.h>
#include <timing.h>

static const char str[] = "Hello, world!";

replicated long * ptr;
replicated char * str_out;

int main (void)
{
     // long is the reliable word length, 64-bits.
     const long n = strlen (str) + 1;

     // Allocating a copy of data on each nodelet typically reduces migrations for commonly ac
     mw_replicated_init ((long*)&ptr, (long)mw_malloc1dlong (n));
     mw_replicated_init ((long*)&str_out, (long)malloc (n * sizeof (char)));

     starttiming(); // For the simulator.  Start gathering stats here.

     for (long k = 0; k < n; ++k)
          ptr[k] = (long)str[k]; // Remote writes

     for (long k = 0; k < n; ++k)
          str_out[k] = (char)ptr[k]; // Migration and remote write

     printf("%s\n", str_out);  // Migration back
}
```

1.2.1 Compiling and Simulating Hello World with Replication

Here we show how to compile the "Lucata-aware" hello world example and to run it with the simulator.

```
In [5]: %%bash
        emu-cc -o hello-world.mwx $FLAGS hello-world.c
        ls *.mwx
```

```
hello-world.mwx
hello-world-naive.mwx
```

```
In [6]: %%bash
        emusim.x -m 21 --total_nodes 1 -- hello-world.mwx
```

```
Start untimed simulation with local date and time= Wed Aug 31 14:11:01 2022

End untimed simulation with local date and time= Wed Aug 31 14:11:01 2022

SysC Enumeration done. Program launching...
Simulation @0 s with local date and time= Wed Aug 31 14:11:01 2022

Hello, world!

Info: /OSCI/SystemC: Simulation stopped by user.



        SystemC 2.3.3-Accellera --- Feb 22 2022 09:27:12
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
```

1.3 Hello World Spawn Example

That example kept one thread alive and migrating between nodelets. This next example, hello-world-spawn.c, uses Cilk's thread spawning intrinsic along with replicated memory.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cilk.h>

#include <memoryweb.h>
#include <timing.h>

const char str[] = "Hello, world!";

static inline void copy_ptr (char *pc, const long *pl) { *pc = (char)*pl; }

replicated long * ptr;
replicated char * str_out;

int main (void)
{
     long n = strlen (str) + 1;

     mw_replicated_init ((long*)&ptr, (long)mw_malloc1dlong (n));
     mw_replicated_init ((long*)&str_out, (long)malloc (n * sizeof (char)));

     starttiming();

     for (long k = 0; k < n; ++k)
```

```
        ptr[k] = (long)str[k]; // Remote writes

    for (long k = 0; k < n; ++k)
        cilk_spawn copy_ptr (&str_out[k], &ptr[k]);

    printf("%s\n", str_out);  // Migration back
}
```

```
In [7]: %%bash

        #Compile the code
        emu-cc -o hello-world-spawn.mwx $FLAGS hello-world-spawn.c
        #Note that we are simulating this with at least 4 nodes! This should give us different
        emusim.x -m 21 --total_nodes 4 -- hello-world-spawn.mwx
        #Then we can print out all the output files that were generated.
        ls hello-world-spawn.*
```

```
Start untimed simulation with local date and time= Wed Aug 31 14:11:04 2022

End untimed simulation with local date and time= Wed Aug 31 14:11:04 2022

SysC Enumeration done. Program launching...
Simulation @0 s with local date and time= Wed Aug 31 14:11:04 2022

Hello, world!

Info: /OSCI/SystemC: Simulation stopped by user.
hello-world-spawn.c
hello-world-spawn.cdc
hello-world-spawn.mps
hello-world-spawn.mwx
hello-world-spawn.vsf
```

```
        SystemC 2.3.3-Accellera --- Feb 22 2022 09:27:12
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
```

1.3.1 Simple Comparison

Then we can compare the output of the normal Hello World and the Spawn Hello World for the statistics that are different. The files labeled `*.cdc` have some basic statistics about the simulated system that will change with memory allocation type (naive or replicated) and system size (1 to 8 nodes).

```
In [8]: %%bash
        #Print out all the .cdc files we generated
        ls *.cdc
```

```
hello-world.cdc
hello-world-naive.cdc
hello-world-spawn.cdc
```

```
In [9]: %%bash
        less hello-world-spawn.cdc
```

```
***********************************************
Program Name/Arguments:
hello-world-spawn.mwx
***********************************************
Simulator Version: 22.2.22
***********************************************
Configuration Details:
Ring Model = Stratix: 3 GC Clusters, 8 MSPs
Number of Nodes=4
Total Memory (in MiB)=8
Logical MSPs per Node=1
Log2 Memory Size per MSP=21
GC Clusters per Node=3
GCs per Cluster=8
Capture queue depths=false
Core Clock=180 MHz, Pd=5.556
Memory DDR4-2133: Bandwidth = 1.886 GiB/s = 2.025 GB/s
SRIO SystemIC bandwidth=2.32 GiB/s (2.5GB/s)
***********************************************
PROGRAM ENDED.
Emu system run time 0.000624 sec==624494400 ps
System thread counts:
        active=1, created=15, died=14,
        max live=4 first occurred @47808552 ps with prog 7.66% complete
        and last occurred @47808552 ps with prog 11.6% complete
Num_Core_Cycles=112400
Num_SRIO_Cycles=390309
Num_Mem_Cycles=158099
***********************************************
***********************************************
Simulator wall clock time (seconds): 9
```

Note what changes in this file between the normal "replicated" Hello World and the "replicated+spawn" version of the code.
* We simulate with a different number of nodes so we used a different amount of memory * The larger simulation takes a bit longer to run and shows different statistics for the `active` threads and progession.

```
In [10]: %%bash
         diff hello-world.cdc hello-world-spawn.cdc
```

```
3c3
< hello-world.mwx
---
> hello-world-spawn.mwx
9,10c9,10
< Number of Nodes=1
< Total Memory (in MiB)=2
---
> Number of Nodes=4
> Total Memory (in MiB)=8
21c21
< Emu system run time 0.000623 sec==623383200 ps
---
> Emu system run time 0.000624 sec==624494400 ps
23,28c23,28
<         active=1, created=1, died=0,
<         max live=1 first occurred @0 s with prog 0% complete
<         and last occurred @0 s with prog 0% complete
< Num_Core_Cycles=112200
< Num_SRIO_Cycles=389614
< Num_Mem_Cycles=157818
---
>         active=1, created=15, died=14,
>         max live=4 first occurred @47808552 ps with prog 7.66% complete
>         and last occurred @47808552 ps with prog 11.6% complete
> Num_Core_Cycles=112400
> Num_SRIO_Cycles=390309
> Num_Mem_Cycles=158099
31c31
< Simulator wall clock time (seconds): 1
---
> Simulator wall clock time (seconds): 9
```

1.3.2 Cleanup

Finally we can clean up our code directory and the output files using the included Makefile in this directory.

```
In [11]: !make clean
```

```
rm -f *.mwx *.tqd *.cdc *.vsf *.mps; \
./helpers/backup_imgs.sh
```

1.3.3 Exercises

To further your understanding of this topic we encourage you to try the following:
1) Restart the notebook and change the memory size and numbers of nodes that are simulated.

How do the statistics change?
2) Investigate the other output files like the `.vsf` file and understand how they are different for different applications. More details on these files can be found in the