NB01-hello-world

August 2, 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*, *emusim.x*, and various plotting helper scripts.
- 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 for plotting.
- Look at and understand a simple Cilk spawn operation for the Lucata architecture.

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]: !. /tools/emu/pathfinder-sw/set-lucata-env.sh

Lucata tools are added to current path from /tools/emu/pathfinder-sw/22.02
```

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/tutorial/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.

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

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

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

```
Start untimed simulation with local date and time= Tue Aug  2 10:43:45 2022

Hello, world!
TID0 NODE 0 DIED NODE 0 FUNC @_Exit @cycle 7319
End untimed simulation with local date and time= Tue Aug  2 10:43:45 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
}
```

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

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

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

```
Start untimed simulation with local date and time= Tue Aug  2 10:43:47 2022

End untimed simulation with local date and time= Tue Aug  2 10:43:47 2022

SysC Enumeration done. Program launching...
Simulation @0 s with local date and time= Tue Aug  2 10:43:47 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 one, hello-world-spawn.c, uses Cilk's thread spawning intrinsic:

```
#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

        emu-cc -o hello-world-spawn.mwx $FLAGS hello-world-spawn.c
        emusim.x --untimed -m 21 --total_nodes 1 -- hello-world-spawn.mwx
        ls hello-world-spawn*
```

```
Start untimed simulation with local date and time= Tue Aug  2 10:43:50 2022

End untimed simulation with local date and time= Tue Aug  2 10:43:50 2022

SysC Enumeration done. Program launching...
Simulation @0 s with local date and time= Tue Aug  2 10:43:50 2022

Hello, world!

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

Then we can compare the output of the normal Hello World and the Spawn Hello World for the statistics that are different.

```
In [8]: !make clean

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