Outline

This presentation covers the following topics

- Lucata Workflow
  - *X86 Debugging*
  - *Simulation*
  - *Hardware*

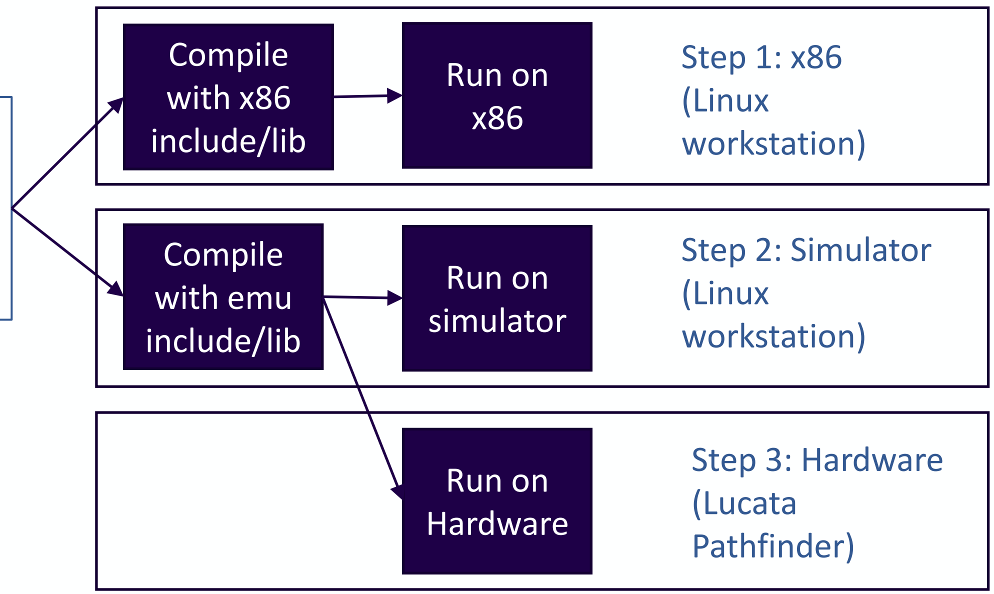
Slides originally developed by Janice McMahon, Lucata Corporation

Debugging

Workflow step 1: x86 execution for verification of correctness

Single program

```
#ifdef X86
#include "memoryweb_x86.h"
#else
#include "memoryweb.h"
#endif
// rest of C/C++ Cilk program
```

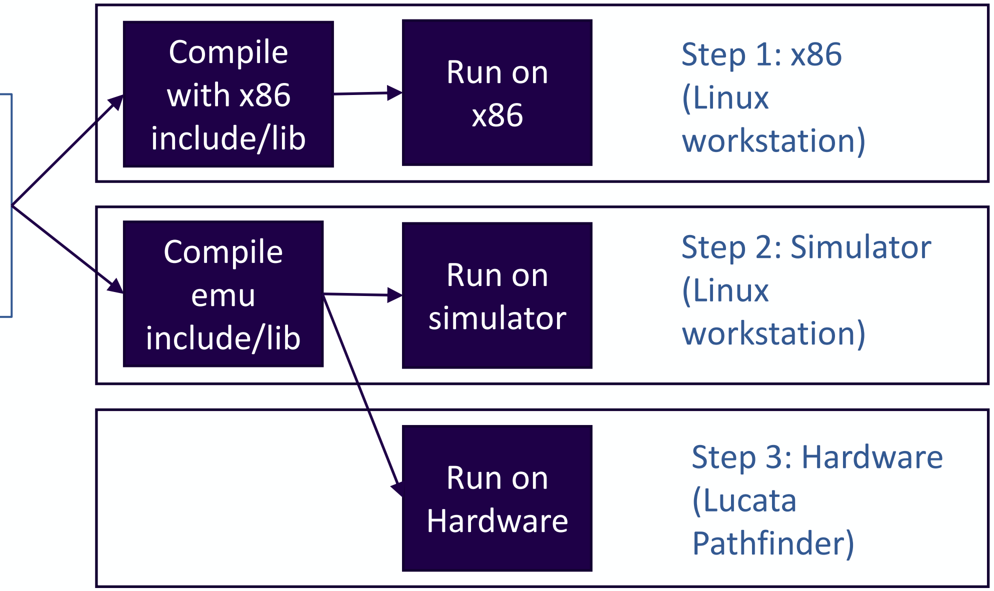


Software Development Workflow

Single program

```
#ifdef X86
#include "memoryweb_x86.h"
#else
#include "memoryweb.h"
#endif
// rest of C/C++ Cilk program
```

- Only difference is include file
- Program uses intrinsics, mw_malloc functions for distributed data
- X86 version mimics single node with single core



Development steps should be done in sequence
Lucata hardware platform used only to run final code
All tools run on Linux workstation

Workflow Step 1

Single program

```
#ifdef X86
#include "memoryweb_x86.h"
#else
#include "memoryweb.h"
#endif
// rest of C/C++ Cilk program
```

Compile with x86 include/lib → "exe" file → Run on x86

Step 1: x86

- Uses standard Linux compiler requiring Cilk support (GCC v5-v7 or OpenCilk)
- Compile with special paths (flags to compiler)
- Could use standard Linux tools (editor, debugger, profiler)
- Program is built and run the same manner as any C program

Example Linux commands:

```
> clang -DX86 -I /usr/local/emu/x86/include main.c -o
    main -L /usr/local/emu/x86/lib -lemu_c_utils
> main
```

Produces executable file

Runs program on x86

Verify correct operation of parallel Lucata program

Debug in x86 mode

- Provides cross-compilation for Emu codes on x86
- Use for rapid building and testing of codes before deployment to Emu architecture
- Treats system as single node with multiple Cilk threads
- Requires Cilk support in x86 compiler
- Use x86 library and include paths when building

Sample Program Execution: intrs_hook.c

```
#include <cilk/cilk.h>
#ifdef X86
#include <stdio.h>
#include <memoryweb_x86.h>
#include <emu_c_utils.h>
#else
#include <memoryweb.h>
#include <emu_c_utils/emu_c_utils.h>
#endif
#include <emu_c_utils/emu_c_utils.h>
…
```

- Uses different include files
- Uses different directories for includes and libraries

```
$ opencilk-2.0.1/bin/clang -I /usr/local/emu/x86/include/emu_c_utils
-DX86 intrs.c -L /usr/local/emu/x86/lib -l emu_c_utils  -o intrs

$ ./intrs
cycles = 13056

$ opencilk-2.0.1/bin/clang  -I
/usr/local/emu/x86/include/emu_c_utils -DX86 intrs_hook.c -L
/usr/local/emu/x86/lib -l emu_c_utils  -o intrs_hook

$ ./intrs_hook
{"region_name":"example","time_ms":0.69,"ticks":347008}
time (ms) = 0.694016
```

We recommend using OpenCilk

Program is executed as in standard Linux manner

Unit Summary: Debugging

- Code modifications for x86 execution
- Building and running in x86 mode

<u>Exercises:</u>

- Re-build all examples for x86
- Use standard Linux tools (gprof, gdb, etc.) on x86 executable

Simulation

Workflow step 2: implementation study using simulator

Workflow Step 2

Single program

```
#ifdef X86
#include "memoryweb_x86.h"
#else
#include "memoryweb.h"
#endif
// rest of C/C++ Cilk program
```

Compile with emu include/lib → "mwx" file → Run on simulator

Step 2: Simulator

- Use Lucata compiler, linker and simulator on Linux platform
- Simulator produces thread migration and memory usage statistics
- Used with profiler for visualization of those statistics

Example Linux commands:

```
> emu-cc -o main.mwx main.c -lemu_c_utils
> emusim.x -- main.mwx
> emusim_profile dir - main.mwx
```

Produces "mwx" executable file

Simulator mimics execution of mwx and produces output files with detailed statistics

Runs simulator and produces image files from statistics that can be viewed in a browser

Verify correctness of Emu Cilk program
Architecture modeling and study
Understand parallel performance characteristics for *small data sets*

Simulation Modes

- Untimed mode produces summary statistics
  - Runs a functional rather than a timed simulation
- Timed mode produces detailed statistics
  - Wealth of detail included in output files (*.vsf, *.cdc, *.msp)
  - Entered via function call in code (used in hooks functions)
  - Specifies that ALL code AFTER the call will be included in detailed timing measurements
  - Required for CLOCK intrinsic in the simulator; produces NOOP on hardware
  - Used to generate raw data for profiler images

Simulator Options for Program Control

- Simulator Control
  - Initialization of memory — `--initialize_memory`
  - Run untimed (functional) — `--profile functional`
  - Maximum simulation time — `--max_sim_time`
  - Sampling interval — `--timing_sample_interval`
  - Return values — `--*return_value*`
  - Output monitor period — `--output_monitor_period`
  - Help — `-h, --help`
  - Output file base — `-o, --base_ofile`
- Machine Configation
  - Memory per Node — `-m, --log2_memory_per_node`
  - Total Nodes — `-n, --total_nodes`

Simulator Options for Profiling Execution

- Execution information printed to screen
  - List all spawn, quit, migrate operations to screen

    --{untimed_}short_trace
  - Thread ID, Node ID, TPC, Memory and Register effects for every instruction executed

    --verbose_isa
  - Verbose thread information

    --verbose_tid
- Instruction counts per function (.uis file)

  --output_instruction_count
- Queue statistics (.tqd file)

  --capture_timing_queues

Used by profiler to generate data for images

Example: Short Trace

```
>>>>>>>> /usr/local/emu/bin/emusim.x --total_nodes 4 --untimed_short_trace -- saxpy_1d.mwx 4 32 5
Start untimed simulation with local date and time= Thu Apr  1 17:16:11 2021

TID0 NODE 0 SPAWN CHILD TID1 FUNC @main @cycle 2634
TID1 NODE 0 MIGRATE DEST 1 FUNC @main.outline_.otd1 @cycle 2673  TPC 0x800024ef Inst LDE
TID0 NODE 0 SPAWN CHILD TID2 FUNC @main @cycle 2687
TID1 NODE 1 MIGRATE DEST 2 FUNC @main.outline_.otd1 @cycle 2693  TPC 0x800024ef Inst LDE
TID1 NODE 2 MIGRATE DEST 3 FUNC @main.outline_.otd1 @cycle 2713  TPC 0x800024ef Inst LDE
TID2 NODE 0 MIGRATE DEST 1 FUNC @main.outline_.otd1 @cycle 2726  TPC 0x800024ef Inst LDE
TID1 NODE 3 MIGRATE DEST 0 FUNC @main.outline_.otd1 @cycle 2733  TPC 0x800024ef Inst LDE
TID0 NODE 0 SPAWN CHILD TID3 FUNC @main @cycle 2740
TID2 NODE 1 MIGRATE DEST 2 FUNC @main.outline_.otd1 @cycle 2746  TPC 0x800024ef Inst LDE
TID1 NODE 0 MIGRATE DEST 1 FUNC @main.outline_.otd1 @cycle 2753  TPC 0x800024ef Inst LDE
TID2 NODE 2 MIGRATE DEST 3 FUNC @main.outline_.otd1 @cycle 2766  TPC 0x800024ef Inst LDE
TID1 NODE 1 MIGRATE DEST 2 FUNC @main.outline_.otd1 @cycle 2773  TPC 0x800024ef Inst LDE
TID3 NODE 0 MIGRATE DEST 1 FUNC @main.outline_.otd1 @cycle 2779  TPC 0x800024ef Inst LDE
TID2 NODE 3 MIGRATE DEST 0 FUNC @main.outline_.otd1 @cycle 2786  TPC 0x800024ef Inst LDE
TID0 NODE 0 SPAWN CHILD TID4 FUNC @main @cycle 2793
TID1 NODE 2 MIGRATE DEST 3 FUNC @main.outline_.otd1 @cycle 2793  TPC 0x800024ef Inst LDE
TID3 NODE 1 MIGRATE DEST 2 FUNC @main.outline_.otd1 @cycle 2799  TPC 0x800024ef Inst LDE
TID2 NODE 0 MIGRATE DEST 1 FUNC @main.outline_.otd1 @cycle 2806  TPC 0x800024ef Inst LDE
TID1 NODE 3 MIGRATE DEST 0 FUNC @main @cycle 2813  TPC 0x8000238d Inst ADDM
TID1 NODE 0 DIED NODE 0 FUNC @main @cycle 2817
TID3 NODE 2 MIGRATE DEST 3 FUNC @main.outline_.otd1 @cycle 2819  TPC 0x800024ef Inst LDE
TID2 NODE 1 MIGRATE DEST 2 FUNC @main.outline_.otd1 @cycle 2826  TPC 0x800024ef Inst LDE
TID4 NODE 0 MIGRATE DEST 1 FUNC @main.outline_.otd1 @cycle 2832  TPC 0x800024ef Inst LDE
TID3 NODE 3 MIGRATE DEST 0 FUNC @main.outline_.otd1 @cycle 2839  TPC 0x800024ef Inst LDE
TID2 NODE 2 MIGRATE DEST 3 FUNC @main.outline_.otd1 @cycle 2846  TPC 0x800024ef Inst LDE
TID4 NODE 1 MIGRATE DEST 2 FUNC @main.outline_.otd1 @cycle 2852  TPC 0x800024ef Inst LDE
TID3 NODE 0 MIGRATE DEST 1 FUNC @main.outline_.otd1 @cycle 2859  TPC 0x800024ef Inst LDE
TID2 NODE 3 MIGRATE DEST 0 FUNC @main @cycle 2866  TPC 0x8000238d Inst ADDM
TID2 NODE 0 DIED NODE 0 FUNC @main @cycle 2870
TID4 NODE 2 MIGRATE DEST 3 FUNC @main.outline_.otd1 @cycle 2872  TPC 0x800024ef Inst LDE
TID3 NODE 1 MIGRATE DEST 2 FUNC @main.outline_.otd1 @cycle 2879  TPC 0x800024ef Inst LDE
TID4 NODE 3 MIGRATE DEST 0 FUNC @main.outline_.otd1 @cycle 2892  TPC 0x800024ef Inst LDE
TID3 NODE 2 MIGRATE DEST 3 FUNC @main.outline_.otd1 @cycle 2899  TPC 0x800024ef Inst LDE
TID4 NODE 0 MIGRATE DEST 1 FUNC @main.outline_.otd1 @cycle 2912  TPC 0x800024ef Inst LDE
TID0 NODE 0 DIED NODE 0 FUNC @main @cycle 2918
TID3 NODE 3 MIGRATE DEST 0 FUNC @main @cycle 2919  TPC 0x8000238d Inst ADDM
TID3 NODE 0 DIED NODE 0 FUNC @main @cycle 2923
TID4 NODE 1 MIGRATE DEST 2 FUNC @main.outline_.otd1 @cycle 2932  TPC 0x800024ef Inst LDE
TID4 NODE 2 MIGRATE DEST 3 FUNC @main.outline_.otd1 @cycle 2952  TPC 0x800024ef Inst LDE
TID4 NODE 3 MIGRATE DEST 0 FUNC @main @cycle 2972  TPC 0x8000238d Inst ADDM
TID4 NODE 0 DIED NODE 0 FUNC @_Exit @cycle 3623
End untimed simulation with local date and time= Thu Apr  1 17:16:11 2021
```

Trace thread movement throughout program execution to verify expected migration patterns

Object Dump for Debugging

```
>>>>>>>> /usr/local/emu/bin/emu-cc  saxpy_1d.c -o saxpy_1d.mwx
/usr/local/emu/bin/gossamer64-objdump -D saxpy_1d.mwx

saxpy_1d.mwx:     file format elf64-gossamer64


Disassembly of section .text:

0000000040001000 <@saxpy>:
    40001000:   80002000:       ETD     2
    40001001:   80002002:       BCTGT   0x8000200a
    40001003:   80002006:       JMP     0x8000203f
    40001005:   8000200a:       LSR0
    40001007:   8000200d:       DTE     6
    40001008:   8000200f:       ETD     6
    40001009:   80002011:       SLLC    3
    4000100b:   80002015:       DPETA   4
    4000100c:   80002018:       LDE     7
    4000100e:   8000201b:       ETD     6
    4000100f:   8000201d:       SLLC    3
    40001011:   80002021:       DPETA   5
    40001012:   80002024:       ETD     3
    40001013:   80002026:       MULTE   7
    40001015:   80002029:       ADDM
    40001016:   8000202b:       ETA     6
    40001017:   8000202d:       AAIMB   1
    40001018:   80002030:       ATE     6
    40001019:   80002032:       ETD     2
    4000101a:   80002034:       XORE    6
    4000101c:   80002037:       BCTDZ   0x8000203f
    4000101e:   8000203b:       JMP     0x8000200f
    40001020:   8000203f:       JMPE    1

0000000040001021 <@main>:
    40001021:   80002042:       ETD     0
    40001022:   80002044:       DTD2
    40001023:   80002046:       LSR3
    40001025:   80002049:       DTE     0
    40001026:   8000204b:       LIT16   128=0x0080
    40001029:   80002051:       SILL    0x0
    4000102c:   80002057:       SILL    0x0
```

Uses Emu version of standard Linux objdump utility

Thread program counter can be correlated with simulator output or simulator exception message to pinpoint errors

Simulator Error Exception

```
>>>>>>>> /usr/local/emu/bin/emusim.x --total_nodes 4 -- saxpy_1d.mwx

        SystemC 2.3.3-Accellera --- Mar 24 2021 16:05:40
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
Start untimed simulation with local date and time= Thu Apr  1 17:38:35 2021

Translating address == 0
UNTIMED SIMULATION on NODE[0] TID[0] generated exception
ThreadID=0
HW ThreadID=0x1
Thread using HW ThreadID
ThreadletState=Active
ThreadletException=1=Address
ExecutionType=7
Current Instruction:
80004eb2 LD8A: iToken=162 iLength=3 nibbles=b3e000
Threadlet TCB Data:
TCB.(TPC)=(0x80004eb2) (32 bits each)
TCB.(D,D2)=(1,1) (one bit each)
TCB.(A,A2)=(1,1) (one bit each)
TCB.(TS,TSDATA)=(0,0x0) (two bits, four bits)
TCB.AID=0x1 (8 bits)
TCB.MaxDepth=0xff (8 bits)
TCB.Priority=0x1 (8 bits)
TCB.(NaN,U,V,CB,N,Z)=(0, 0, 0, 0, 0, 0)
TCB.M=0 (one bit)
TCB.DB=1 (one bit)

Threadlet Data Registers
A: 0x0=0
A2: 0x180000080000a90=108086393204378256
Format: signed decimal, unsigned decimal, hex
D: 2147503792,  2147503792, 0x80004eb0
D2: 0, 0, 0x0
```

```
E[0]  (Live): 108086393204378256, 108086393204378256, 0x180000080000a90
E[1]  (Live): 2147492262, 2147492262, 0x800021a6
E[2]  (Live): 0, 0, 0x0
E[3]  (Live): 108086393204375568, 108086393204375568, 0x180000080000010
E[4]  (Live): 1152921504606848016, 1152921504606848016, 0x1000000000000410
E[5]  (Live): 36028797018978840, 36028797018978840, 0x80000000003a18
E[6]  (Live): 36028797018977280, 36028797018977280, 0x80000000003400
E[7]  (Live): 36028797018977288, 36028797018977288, 0x80000000003408
E[8]  (Live): 1, 1, 0x1
E[9]  (Live): 1040, 1040, 0x410
E[10] (Live): 108086395351859176, 108086395351859176, 0x1800000fffffe8
E[11] (Live): 108086393204378248, 108086393204378248, 0x180000080000a88
E[12] (Live): 0, 0, 0x0
E[13] (Live): 108086393204375568, 108086393204375568, 0x180000080000010
E[14] (Live): 0, 0, 0x0
E[15] (Live): 0, 0, 0x0

Other Useful Data
Fence Counter=0
Source Node=0
Dest Node=0
End untimed simulation with local date and time= Thu Apr  1 17:38:35 2021

halt_data_dump function has been removed
.halt_data_dump function has been removed
.halt_data_dump function has been removed
.halt_data_dump function has been removed
.GENERATED EXCEPTION
```

Manufactured error (arguments missing)

Find TPC in object dump to locate error

Unit Summary: Simulation

- Many options for controlling simulator to exercise a variety of machine configurations
- Many options for producing detailed information about program execution to aide in understanding performance
- Used in conjunction with Linux utilities to pinpoint errors at low level of code

Exercises:

- Run previous examples with timed/untimed mode, examine data file contents, study migration patterns via short-trace.

Hardware

Workflow step 3: execution for timing measurement

Workflow Step 3

Single program

```
#ifdef X86
#include "memoryweb_x86.h"
#else
#include "memoryweb.h"
#endif
// rest of C/C++ Cilk program
```

- Use Lucata compiler and linker on Linux platform
- Executable must be copied to Lucata machine over LAN and run on that machine (i.e., cross-compiled)
- Execution time can be measured but no statistics gathered

Example Linux commands:

```
> emu-cc -o main.mwx main.c -lemu_c_utils
> scp main.mwx LUCATA:
> ssh LUCATA
LUCATA> emu_multimode_exec main.mwx
```

Produces "mwx" executable file

Copy mwx to Lucata machine over network

Run command executed on Lucata machine

Run and measure program on Lucata machine

Pathfinder Configuration

- Single-node Execution
  - Program runs on a single node
  - Uses all Gossamer cores on that node
  - Users can work independently on different nodes
- Multi-node Execution
  - Program runs on full system
  - Can access all nodes (all Gossamer cores)
  - Single user

Pathfinder Hardware Execution

- Compile programs on Host then scp to Pathfinder
- Single-node Execution
  - Launched on node using emu_handler_and_loader
- Multi-node Execution
  - Launched on node 0 using emu_multinode_exec

Program Execution Utilities

- Load program and data to all nodes
- Launch initial thread into the system
- Monitor the system exception queue and handle system services until a thread quits or an exception occurs
- Terminate by issuing a checkpoint to clear the system and dump any remaining threads
- Print information to log files for each thread that quits, exits, generates an exception, or is checkpointed
- Return the program's return value

Slurm Extensions for the Pathfinder

- ➢Georgia Tech has developed a Slurm workflow for the Pathfinder to simplify this process
- ➢No need to scp data/programs!
  - All data is shared via network-mounted storage
- ➢Use sbatch to launch jobs for single-node, single-chassis, or multi-chassis jobs

```
#Show the real-time status of the Pathfinder nodes.
%sinfo --federation -M pathfinder
```

				CLUSTER:	pathfinder
PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
rg-pathfinder*	up	180-00:00:	1	alloc	c2n1
			31	idle	c0n[0-7],c1n[0-7],c2n[0,2-7],c3n[0-7]

```
#Run the Saxpy command with Slurmemu_handler_and_loader saxpy-1d-workflow.mwx 8 128 5.0
sbatch -M pathfinder -q single-node --wrap "emu_handler_and_loader 0 0 -- saxpy-1d-workflow.mwx 8 128 5.0"
```