Brown Deer Technology

COPRTHR-2 Overview

This document provides an overview of the COPRTHR-2 Software. Additional information is available in the following supporting documentation:

- *COPRTHR-2: Development Tools* provides information about development tools.
- *COPRTHR-2: API Reference* provides information about the supported APIs.
- *COPRTHR-2: Software Development Guide* provides detailed information about software development using COPRTHR-2.

Document revision b (20160629)

Contents

1 Overview 3
1.1 History 3
1.2 Components 3
1.3 Features 3
1.4 syscore: persistent proto-OS supporting fast coprocessor code execution 4
1.5 coprcc: cross-compiler front-end for Epiphany 5
1.6 coprcc-info: tool for detailed analysis of coprocessor binaries 5
1.7 coprcc-db: run-time integrated Epiphany core debugger 6

1 Overview

1.1 History

The CO-PRocessing THReads® SDK version 1.6 provides programming support for heterogeneous computing including support for the Epiphany architecture. COPRTHR® version 2 represents a significant redesign across the software stack with a focus on enabling greater performance and control for the application developer targeting scalable RISC array processors such as the Epiphany architecture.

The design and development of COPRTHR-2 has evolved from the internal code development project designated *Anthem* which focused on addressing the observed issues encountered in programming the Epiphany Parallella platform, and was strongly influenced by the work on threaded MPI.

The overarching focus guiding the Anthem project was enabling greater performance and providing the programmer with greater control over the compilation of Epiphany binaries, especially, in the area of managing the limited core-local memory that must be shared between instructions and data. Addressing these issues required a redesign of very low-level code in the software stack. As a result, COPRTHR-2 is not backwards compatible with many features of COPRTHR-1. However, the significant components of COPRTHR have thus far been migrated forward, including the COPRTHR API for Epiphany and support for threaded MPI.

1.2 Components

At the time of release the components of COPRTHR-2 include:

- **syscore**: a persistent proto-OS supporting fast coprocessor code execution
- **coprcc**: cross-compiler front-end for Epiphany
- **coprcc-info**: tool for detailed interrogation coprocessor binaries
- **coprcc-db**: run-time integrated coprocessor core debugger
- **libcoprthr**: support for the COPRTHR coprocessor API
- **libcoprthr_mpi**: support for threaded MPI

1.3 Features

Highlights of the COPRTHR-2 features include:

- Precise control over code placement in the compilation model
- Support for host-offload and direct execution run-time models
- Unified virtual address-space (UVA) between host and coprocessor
- Host call interoperability: make Linux host calls directly from coprocessor
- Threaded MPI programming model provides a familiar parallel programming API
- Support for ARL OpenSHMEM API
- Integrated coprocessor debugging tool for improved visibility of operation
- Experimental support for dynamic calls using an instruction cache

1.4 syscore: persistent proto-OS supporting fast coprocessor code execution

A significant development in COPRTHR-2 is the segmentation of system and user code within Epiphany binaries. This segmentation allows a bootable persistent proto-OS kernel to execute on the coprocessor cores in order to reduce the latency incurred in the launch of user application code. We describe this model as "hot loading" the Epiphany application binary. The design enables extremely fast response times between the host platform and the Epiphany coprocessor. Previous approaches used for Epiphany suffered from very high overhead in the startup of an application kernel. The development of syscore was challenging since the core-local memory in the Epiphany architecture is a precious commodity.

The nominal memory layout of an Epiphany binary under COPRTHR-2 is shown below:

```
Core-local memory (0x0000-0x80000):
       0x0000-0x0058           Reserved
       0x0058-0x0398           SYSCORE
       0x0400-                 USRCORE
       0x7f80-0x8000           Reserved

Shared global memory
       0x8e000000-0x8e002000   SYSMEM
       0x8e002000-             USRMEM
       0x8e100000-0x8f000000   Shared Heap
```

When the Epiphany coprocessor is opened for use it is booted into syscore which causes the execution of a persistent generic kernel compiled into the memory segments SYSCORE and SYSMEM. This memory is not available to the programmer. Since core-local memory is precious and must be utilized judiciously, syscore is designed to be compact in size while still providing essential functionality. Programmers using COPRTHR-2 will find the ability to create much smaller

program binaries and have greater control over their program layout when compared with COPRTHR-1 or the typical use of the eSDK directly.

The advantages brought by syscore are seen immediately with the loading of an application binary. A distributed tree loader is provided that reduces application kernel launch time by almost 30x over COPRTHR-1 or the eSDK. Not only is the loader fast, it is also scalable, and the design will exhibit logarithmic scaling with increased core count. Support for persistent threads, which was a very common feature request for COPRTHR-1, is now trivially supported. Greater host-coprocessor interoperability is also supported with the ability to make Linux host calls directly from the Epiphany cores using an RPC mechanism tightly integrated into syscore. The indispensable debugging tool *printf()* is now trivial and available to programmers along with most of the Linux system calls and calls provided by *stdio*. As an example, it is now possible to access the host file-system from the Epiphany cores.

1.5 coprcc: cross-compiler front-end for Epiphany

Under COPRTHR-1 a compiler front-end called *clcc* was used to compile Epiphany binaries and supported both a traditional compiler workflow or just-in-time compilation. This tool is deprecated under COPRTHR-2 and replaced with the compiler front-end called *coprcc*. The replacement was motivated by the goal of providing more flexibility in the compiler workflows that programmers may require. As a result, *coprcc* will behave much more like the native Epiphany compiler e-gcc insofar as it accepts all of the default compilation flags, and can be provided C source (.c), assembly (.s), or precompiled re-linkable object (.o) files as part of the compilation workflow. This allows greater flexibility in compilation steps and the conventional use of partial compilation steps. Support is provided for compiling coprocessor binaries suitable for explicit off-load from a host program. A new feature is support for compiling host executables that will implicitly off-load an embedded coprocessor binary for execution on the coprocessor device. This new feature is exemplified by the ability to compile a simple “hello, world” program which may be executed directly on any number of coprocessor cores.

1.6 coprcc-info: tool for detailed analysis of coprocessor binaries

One of the design objectives for COPRTHR-2 was enabling greater control over the layout of compiled coprocessor binaries, including the precise placement of executable code. In support of these capabilities, the tool *coprcc-info* is provided for extracting detailed information about the layout of a coprocessor binary. The tool is similar to the common *nm* and *readelf* commands provided by binutils. However, *coprcc-info* it is tailored to the COPRTHR-2 binaries generated by *coprcc* and provides information that has no equivalent in binutils.

1.7 coprcc-db: run-time integrated Epiphany core debugger

One of the most challenging issues encountered in using the Epiphany coprocessor was having to execute code "blindly" without any visibility into what might be happening (good or bad) as threads executed on the cores. Debugging was a challenge since the normal practice of using *printf()* statements for debugging was not properly supported. Although COPRTHR-2 provides interoperability with the Linux host and supports *printf()* trivially now, the need for an integrated debugger still exists. The *coprcc-db* tool was developed for this purpose.

With the run-time core debugger enabled, at any time during the execution of code on the Epiphany coprocessor (especially useful when your code is hanging due to a bug) hitting *ctrl-z* will cause the terminal to drop-down into the core debugger *coprcc-db*. This will provide a shell-like command-line that can be used to query the state of the coprocessor cores. The core debugger is designed to support many-core coprocessors and provides a simple prefix notation for applying any command to any number of selected cores. The use of a shell-based debugger extends its functionality to include any shell commands for a flexible and familiar debugging environment. As an example, the output of any debug command may be piped through UNIX commands like *grep* and also redirected to an output file for subsequent analysis.