ClearSpeed Software Tool Chain
Introduction

• Software tool chain used to develop code to target a specific microprocessor.
  – **Software Build tools**
    • Compiler, Assembler, Linker
  – **Code analysis tools**
    • Debugger, Profiler

• Embedded processors
  – Often use a hosted tool chain to allow for “cross-compilation” of executable code.
  – Code is loaded onto the embedded processor at run time.
  – In this model the debugger also has to run on the host processor but debug the target processor remotely.
  – Profiling is done by storing the trace data on the embedded system and then retrieving and post processing the data back on the host system.
Introduction

• ClearSpeed CSX600 is essentially an embedded microprocessor connected to an x86 host via a PCI bus.

• Tools have to be targeted to run in this ‘hosted’ environment across multiple operating systems.

• Even so the development environment is as fully featured as that of the host processor.

• Cross compilation tools are provided for x86.
  – Driver (**cscn**)
  – Compiler (**cncc**)
  – Linker (**cld**)
  – Assembler (**mass**)

• Tools for cross debugging and profiling are also provided.
  – Debugger (**csgdb**)
  – Profiler (**csvprof**)


The C^n Language

- C^n based on ANSI C
- Very closely coupled with the operation of the ClearSpeed processor architecture.
- Adds mono and poly variable type specifiers to language.
  - Mono variable: A variable that has one instance.
  - Poly variable: A variable that has many instances with, typically, different data values on each of the Processing Elements (PE).
- Support for both basic and derived poly types.
- Pointer support.
- Vector types for targeting SIMD within each PE.
- Full debugging support for the language.
The C\textsuperscript{n} Language – An example

A simple approximation of PI

\[ \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}. \]

Original host C code

```c
#include <stdio.h>
#include <math.h>
#include <reduction.h>
#include <lib_ext.h>

int main(void) {
    int times = 92160000;
    int i = 0;
    poly double series = 0.0;
    double pi = 0.0;
    int pe_slice = times/get_num_pes();
    poly int pe_start = pe_slice*get_penum();
    for (i = 0; i <= pe_slice; i++)
        series += 1.0 / ((i+1.0+pe_start) * (i+1.0+pe_start));
    pi = sqrt(6.0*cs_reduce_sum(series));
    printf("Value of Pi = %.7g\n", pi);
}
```

Using C\textsuperscript{n} to target the PE array

```c
#include <stdio.h>
#include <math.h>
#include <reduction.h>
#include <lib_ext.h>

int main(void) {
    int times = 92160000;
    int i = 0;
    poly double series = 0.0;
    double pi = 0.0;
    int pe_slice = times/get_num_pes();
    poly int pe_start = pe_slice*get_penum();
    for (i = 0; i <= pe_slice; i++)
        series += 1.0 / ((i+1.0+pe_start) * (i+1.0+pe_start));
    pi = sqrt(6.0*cs_reduce_sum(series));
    printf("Value of Pi = %.7g\n", pi);
}
```
C^n Compiler

• Optimization
  – Compiler optimizations for both mono and poly code from both –O0 to –O4.
  – Loop unrolling, register allocation, instruction scheduling.
  – Allowing performance to be accessed from C^n without falling back to assembly code.
  – Using existing application code for benchmarking and improving the compiler.

• Poly Vector types
  – Allows access to SIMD performance within each PE.
  – Operator overloading for standard C^n operators.
  – Intrinsic support for operations such as reduction.
The ClearSpeed Software Tool Chain

\textbf{C}^n \textbf{Compiler}

- **Assembly inserts**
  - Allows low level instructions or highly optimized assembly routines to be targeted from \( C^n \) source code.
  - Simple passing of parameter variables between \( C^n \) and assembly insert code.

- **Debugging (csgdb)**
  - Support for standard debug information generation at all optimization levels using the \(-g\) flag.
  - Runtime stack checking to detect stack overrun.

- **Profiling (csvprof)**
  - Function instrumentation for use with the hardware trace port on the CSX processor.
  - Automatically instrumented with the \(-pg\) flag.
The ClearSpeed Software Tool Chain

C^n Build Tools

• Assembler (mass)
  – Allows user defined assembly macros and use of custom instructions.
  – Generates debug information at the instruction level for assembly debugging in CSGDB.

• Linker (cld)
  – Support for dynamically linked executables.
  – Allows particular code sections to reside in specific memory locations.
  – Uses standard ELF file format for object files and executables.

• Unified front-end (cscn)
  – Standard compiler front-end model similar to gcc.
  – Drive all the command line build tools from one interface.
Standard library support

- **C^n Standard Library**
  - Standard GNU newlib library ported for both serial and parallel code.
  - Support for standard I/O routines, e.g. printf() and printfp().
  - Math functions, e.g. sin() and sinp().

- **C^n Extension Library**
  - Utility functions for processor for accessing CSX600 specific information.
  - PE id numbers, Number of PE’s, Enabled PE’s.
  - Access to the on chip DMA engine.
  - Semaphore and hardware threading functions.
Optimized specialist libraries

- **Vector Math Library**
  - Optimized set of high performance parallel vector math functions using compiler vector types.
  - Accelerated math functions to allow access to highly optimized PE vector code.

- **Random Number Generator**
  - Optimized set of parallel random number generating routines.

- **PE Variable Reduction**
  - Set of optimized reduction routines for getting parallel data types back from the PE array.
  - Four types of reduction: sum, product, min, max
Memory transfer libraries

• PE Array I/O Libraries
  – Used for targeting the on-chip I/O engine for data transfer between mono and poly memory spaces.
  – Simple library calls provide a memcpy() style interface to the data transfer hardware.
  – Synchronous routines provided which maintain cache coherency with mono processor.
  – Higher performance asynchronous routines provided to work both multi-threaded for large data sets and inline for smaller transfers.
  – Hardware I/O engine can be seen in the profiler to allow better understanding of data movement.
• Port of GDB debugger for CSX600 (csgdb)
  – Supports real-time hardware debugging of ClearSpeed processors.
  – Debugging done in place on the CSX600 processor allowing the real code on the real hardware to be investigated.
  – ClearSpeed-specific back end added to support hardware.
  – Symbolic debug support for C^n language added to gdb code base.
  – Support for large SIMD array data types added.
  – Additional commands added to support PE array debugging.
  – Support for configuring CSX600 real time hardware and simulator pipeline trace added to csgdb.
  – A familiar interface for most developers.
The ClearSpeed Software Tool Chain

CSGDB Demo

- Debugging a simple $C^n$ Application using csgdb
  - Connecting to the device.
  - Program control.
  - The ‘poly’ type in the debugger.
  - Low level device access.

```c
int main <void>
    test1(i);
    return;

int main <void> (poly int testval - 10;
    printf("CSX program 1 running ....... \n"); test(testval);
    printf("CSX program 1 terminating ....... \n");

cdb> test(testval);

cdb> stop
    test <i=(10 repeats 96 times)>> at simple.1.cn+9
    i++;

cdb> test1(i);

cdb> test1(i=(11 repeats 96 times)>> at simple.1.cn+4
    i++;
```
ClearSpeed Visual Profiler

Java based cross platform profiler GUI for displaying information relating to code performance on the CSX processor.

Displays resource utilization view of processor and relates information back to source code.

Allows users to iteratively improve code performance by tuning applications using the CSX trace hardware.
ClearSpeed Visual Profiler

- **CSX Processor**
  - Extended support for real time hardware trace port with new resource utilization view.
  - Compiler function instrumentation provided with simple build flag. Provides higher level trace view and can profile execution for 10’s of minutes.
  - Full visualization of CSX processor pipeline trace from the cycle accurate simulator.

- **x86 Processor**
  - Enhanced CSAPI profiling support allowing view of all CSAPI calls made.
  - x86 instrumentation library allowing users to profile any x86 source code.
Eclipse IDE

- Eclipse is an open source GUI development toolkit originally funded and developed by IBM.
- Becoming the de-facto standard for embedded processor development tools.
- Provides a cross platform IDE integrating both the build tools and the CSGDB debugger.
- Is extensible and already supports GDB for x86 making driving CSGDB a simple process.
- Allows us again as with CSGDB to leverage the work done in a large open source project.
- Attended EclipseCon 07 in March to accelerate port, gain basic understanding of Eclipse framework and meet with community developers.
Eclipse IDE – CSX Development Perspective

Integrated development environment supporting command line CSX development tools.

Based on the industry standard Eclipse platform.

Supports development of application code using the ClearSpeed C^n language.

Can be used as a plug-in with other existing eclipse based tools to provide an IDE for heterogeneous software development.
Eclipse IDE – CSX Debug Perspective

Standard Eclipse graphical debug interface for CSX processor debugging.

CSX processor provides full hardware debugging of application code.

Provides seamless view of all 96 processor cores and the associated state.

Allows full symbolic debug of the $C^n$ language.

Enhanced views for CSX specific information.
• Developing C\(^n\) Applications Within Eclipse
Release 3.00 Beta

- **Updated release of the ClearSpeed Tools**
  - Improved optimizing compiler for $C^n$.
  - Ease of programming issues in $C^n$ code.
  - Extended support in the vector math, random number and memory transfer libraries.
  - Upgrade to version of csgdb and simpler integration when debugging in conjunction with a host application.
  - Improved feature set in the visual profiler.
  - Eclipse IDE support.

- **Available to ClearSpeed developers.**
  - Download via the support website.
  - Feedback welcome and appreciated as we move towards product.
Conclusion

• What have we seen?

– Software development tools based around tools familiar to x86 developers to aid ease of adoption.

– Continued improvement in the development tools with all features discussed available in the release 3.00 beta.

– Software team actively focused on improving the performance of the compiler out of the box.

– Eclipse IDE provides developers with a cross platform GUI for the both the build tools and the debugger.

– Beta software available to developers enrolled in the beta program now.
Questions?

Matt.Fyles@clearspeed.com