CGT C6x v8.0
- OpenCL and OpenMP performance improvements
- OpenMP production accelerator model
- Open-source-friendly compiler tools
- ARM-linux hosted version to enable compilation on target
- C99 complex type performance improvements
- Native vector type support for C

CGT C6x v8.1
- Mac OS-hosted compiler tools
- Additional OpenMP SIMD pragma support
- Improved OpenMP & OpenCL debug and performance
- Native vector type support for C and C++

Longer Term
- OpenCL and OpenMP improvements
C6000 Compiler Overview

• Industry’s best highly-optimizing C/C++ VLIW compiler
• Optimized support for all C6000 variants
• Compiler exploits performance capability of C6000 by automatically software pipelining inner loops
• Extensive set of SIMD operations to speedup algorithms by up to 16x
  – Automatically exploited by compiler, automatic unrolling
  – Also accessible with intrinsics
• Large range of performance vs. code size options
• Many GCC extensions are supported
• Compiler is tested with all commercial compiler test suites
• Daily regression runs to assure correctness and performance
C6000 v7.2 Compiler Tools

• Performance entitlement for C6600 devices
  – Compiler exploits many new C6600 instructions automatically; up to 4x performance improvement on inner loops
• Improved performance stability
• ELF support/new EABI: Added 40-bit native type (int40_t)
• Multicore application deployment support
• Improved GCC compatibility
C6000 v7.3 Compiler Tools

- Irregular Control Loop Optimization
  - 10 of 14 Layer 2 (MAC layer) loops have significantly improved performance
- C6600 Optimization
  - 7.3.x improves 19% of our suite of C66x loops by an average 46% and a maximum of 143% vs 7.2.x
  - C6600 floating point routine optimizations (floating pt divide, sqrt)
- C6x Linux Support/GCC interlinking
  - Enables use of TI C6x compiler on DSP-critical code in C6x Linux systems
  - TI C6x compiled objects can be linked into Linux dynamic objects with GCC linker
  - TI C6x built DSOs can be interlinked with other Linux EXEs/DSOs
- OpenMP (Early Adopter Capability)
- Ability to build specific runtime support libraries on demand

Irregular Control Code Loop Optimization Overview:
- Improved certain “control code” loops
  - Loop has compound conditions, both in the loop and in the exit condition.
  - Iteration counts are not known when the loop is entered.
  - Involves structure references
- Significant gains in performance on TI C6x DSP can be achieved through:
  - Improved memory ability to utilize speculative loads
  - Improved alias analysis of structures
C6000 v7.4 Compiler Tools

- **OpenMP 3.0 support**
  - Shared memory parallel programming paradigm -- Run multiple threads on multiple cores
- **Thread-safe run-time support libraries (RTS)**
  - When concurrently executing multiple threads, calls to RTS functions are safe
- **Thread local storage support**
  - Each thread has its own copy of any object/variable declared with `__thread`
- **Performance advice for non-DSP programmers**
  - Compiler identifies common and basic performance issues with user's compiler options and C code and suggests specific changes. Use the compiler option `--advice:performance`
- **C99 Complex type support**
  - `float _Complex, double _Complex, long double _Complex`
- **Prelinker improvements to help with debug when using the prelinker**
- **Automatic load speculation (auto –mh)**
  - Compiler/linker will automatically pad memory ranges with –mh=auto compiler option; Enables better irregular loop performance and lower code size.

- **CCS integrates C6000 compiler documentation**
- **Hyperlinks in CCS to supplement information from performance advice**
- **Available now; v7.4 is the recommended release branch for existing development**
C6000 v7.6 Compiler Tools

- v7.6 is a preview of the v8.0 release and is a limited audience release (MCSDK/OpenCL/MP)
- v7.6 is focused on facilitating multicore support of OpenCL and OpenMP
- v7.6 tools are used in the OpenCL production release
- ARM/Linux host support (for OpenCL)
- OpenCL kernel efficiency improvements
- OpenMP accelerator model support (early adopter)
- OpenMP language fixes
- GCC, TI, C99 extensions accepted by default
  - Strict mode available to avoid features conflicting with a strictly conforming program
- Parser upgrade
- Variable-length arrays support (unoptimized)
- v7.6 available now; *v7.6 will not be supported after v8.0 is released*
C6000 v8.0 Compiler Tools -- Details

- Limited-audience release (MCSDK-HPC/OpenCL/OpenMP users)
- Less restrictive licensing terms (Open-source-friendly, non-export controlled)
  - PPA (Debian) packages added (ARM and Linux x86 CGT)
- Object code compiled with <= v7.4 is compatible with v8.0 for C, not C++
  - New C++ RTS; is not object compatible with C++ object compiled with v7.4 and earlier
- OpenMP accelerator model support
- OpenCL kernel efficiency improvements
- Latest compiler infrastructure
  - Facilitates wider variety of architectures, long-term
  - Lays the foundation for future optimization
- ELF EABI-only, 32-bit long only (COFF & 40-bit long type discontinued)
- Discontinues support for legacy processors (6200, 6400, 6700, 6700+)
- Native vector type support (C only)
  - Eases use of SIMD instructions in C
- C99 complex type performance improvements
- v8.0 available in August, 2014

7/22/2014
C6000 Compiler Tools

Into the Future

June, 2014
C6000 Compiler – Retrospective

• First C6x compiler released in 1997 – Two decades of development
• Industry’s best optimizing VLIW compiler for 17 years
• The C6000 Compiler has evolved and adapted:
  – Providing optimized support for all C6000 processor variants
    • Exploitation of new instructions, SPLOOP, compact instructions
  – To support industry-standard object & debug formats: COFF/STABS -> ELF/DWARF
  – To support many GCC extensions

• Many performance and feature advancements:
  – Advanced scheduling heuristics
    • Software pipelining; Loop stage collapsing; Loop unrolling; Automatic SIMD;
      Software pipeline prolog/epilog scheduling; Nested loop scheduling; Inter-block
      scheduling
  – Wide spectrum of performance and code size options
  – Dynamic loading compatibility
  – In most cases, hand-coded assembly performance with C code
C6000 Compiler – Foundations for the Future

• Recent compiler development has focused on enabling multicore users and a wider audience with different needs and applications:
  – OpenCL and OpenMP support
  – Irregular control loop optimization
  – Thread-safe run time support libraries
  – Dynamic loading compatibility
  – C99 complex type support
  – Performance remarks for non-DSP programmers

• In order to further prepare for the next two decades, we’re making significant changes to our compiler toolset. The v8.0 C6000 Compiler is a new toolset that enables us to adapt and respond quickly to the needs of the next generation of multicore and general purpose C6000 applications.
C6000 Compiler – What’s Changing

• The v7.4.x C6000 Compiler will be supported for the long-term
• We’ve added new features to v8.0
  – Native vector types available in all C code, OpenCL-like intrinsics
  – Open-source-friendly distributions of the compiler & runtime support library
  – Additional OpenMP and OpenCL features supported
  – Non-intrusive debug information
• To create a leaner, more adaptable toolset, v8.0 has significant changes
  – Contains significantly re-designed infrastructure. These improvements:
    • Enable future optimizations
    • Allow us to rapidly explore a wider range of ISA/CPU options
  – Does not support legacy features
    • The following legacy features are no longer supported in v8.0 and beyond. These features will continue to be supported in v7.4:
      – COFF
      – 6200, 6400, C6700, C6700+, and Tesla
      – Other seldom used features
C6000 Compiler – Making the Transition

• Preserving investments:
  – **We are committed to long-term support of the v7.4.x compiler**
  – New or existing projects that depend on legacy processors or features will continue to be supported with the v7.4.x compiler
  – New or existing projects that have no need for new features in v8.0 can continue to use the v7.4.x compiler

• Transition when it makes sense to do so. Consider using v8.0 when:
  – Starting a new project
  – A new feature like *native vector types* is needed
  – Acceptable performance of v8.0 on their application has been verified
C6000 Compiler – Processor Support Model for v8.0 and Beyond

The C6000 CGT v8.0 is a **NEW** compiler:

- v8.0 will support C6400+, C6740, and C6600 in ELF EABI mode only
- v7.4.x will continue to support (long-term) all processor variants in ELF EABI or COFF ABI mode
- C++ obj code generated by older compilers is **not** compatible with v8.0 RTS obj libraries
- v8.0 offers comparable performance to v7.4. There will be some performance variation and it will vary based on the application.

**Customers should use CGT v8.0 if they are:**

- Developing new applications using OpenCL, OpenMP, or HPC-MCSDK
- Developing new applications that utilize new compiler features only in v8.0 and above
  - Native Vector Types, for example

**Customers should use CGT v7.4.x if they are:**

- Maintaining an existing code base that they don’t want or need to transition to v8.0 in the near-term
- Developing new applications or maintaining existing applications that use the COFF ABI
- Developing new applications or maintaining existing applications on C6200, C6400, C6700, C6700+, or Tesla
C6000 Compiler – Processor Support Model for v8.0 and Beyond

C6400+, C6740, C6600:
• Supported in v8.0 and beyond – ELF-only
• Supported in v7.4.x – COFF or ELF

C6200, C6400, C6700, C6700+, Tesla:
• NOT supported in v8.0 and beyond
• Supported in v7.4.x (long-term)
C6000 Compiler – Infrastructure Changes

• The underlying representation of the architecture in the compiler has fundamentally changed. There have been other infrastructure changes as well.

• The compiler infrastructure changes:
  1. Allow the compiler to represent a wider variety of DSP and VLIW architecture variations
     • Allow TI to perform faster, wider-ranging architecture exploration
  2. Create a foundation for future optimizations, allowing more
     • Loop optimization
     • Automatic SIMD optimization
     • Effective handling of vector types, resulting in faster code