



The Engine of SOC Design

Xtensa C/C++ Compiler

Advanced, optimizing compiler

v1.0, May 2007



Xtensa C/C++ Compiler (XCC)

- Powerful vectorizing compiler with advanced loop optimizations
 - ANSI C/C++ support
 - Advanced inter-procedural and alias analysis capability
 - Advanced loop optimizations like software pipelining
 - Automatic vectorization (mapping C code to SIMD vector instructions) for Vectra LX and XPRES-generated TIE
 - Automatic inference of XPRES-generated TIE
 - Feedback-directed optimization
- Improved performance over GCC
 - 15-20% average performance improvement on general C code
- Improved code size compared to GCC
 - Up to 10% code size improvement when compiling for optimal code size

- Based on Open64 from SGI
 - SGI's optimizer and code generator
 - gcc's front end
- Open64 is
 - Advanced compiler with huge research community and industry backing
 - Very reliable, stable code base
 - GCC compatible (same compiler flags)
 - We optimize/modify for Xtensa



Open64: High Performance Scalar Compiler

- Came with lots of modern optimizations

SSA-based scalar optimizer

PRE

RVI

Value numbering

Strength reduction

Induction variable elimination

Pointer alias analysis

Array dependence analysis

Control flow optimizations

Loop unrolling

Inner loop optimizations

If conversion

Recurrence breaking

Local and global instruction scheduling

Common subexpression elimination

MIN/MAX recognition

Memory optimizations

Global register allocation with

live range splitting

rematerialization

homing

Extended basic block optimizer

Inter-procedural analysis

Feedback directed compilation

- Software (loop) pipelining
 - VLIW support for dynamic/irregular formats
- Inter-procedural analysis (now also available in Open64)
 - Cross file inlining, dead function/variable elimination, alias analysis, etc
- Feedback directed optimization (now also available in Open64)
 - Compile once with instrumentation, run 'n' times and gather statistics, and recompile
 - Large improvement on size and speed
- Vectorization
 - Compiler automatically vectorizes standard C/C++ code to take advantage of SIMD function units



XCC Enhancements on Open64 .. more

- Code size improvements
- Zero-overhead loop support
- Updating load/store support
- Support for configurability
- 16-bit multiplications

- and the list goes on ...



XCC: TIE Support

- Automatically adds C datatype support for user defined register files in TIE
 - Use C variables to address user-defined register files
 - XCC does register allocation
- User-defined TIE instructions can be used in code as C intrinsics
 - No assembly coding required to use new instructions
 - Use C intrinsic instead (looks like a functional call to new instruction)
 - XCC does instruction scheduling
- XCC understands and treats TIE loads/stores like “normal” load/stores
 - Compiler can do smarter alias analysis and optimizations
 - Not possible if XCC treated TIE load/stores as intrinsics/function calls
- Result
 - Optimizer can operate on TIE operations similar to native instructions
 - Scheduler and register allocator understand TIE

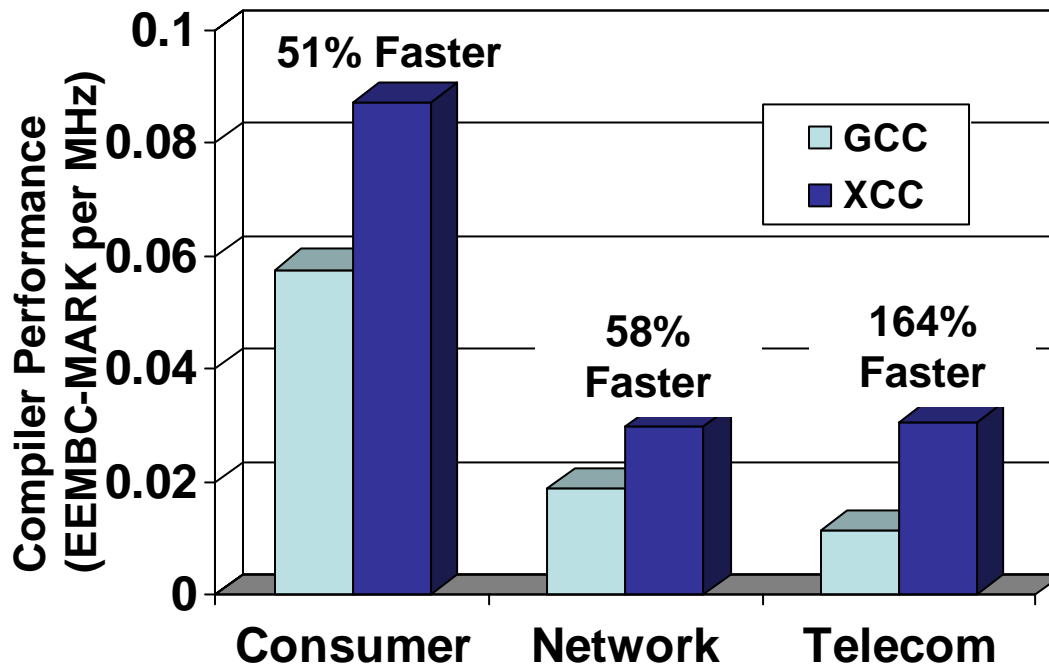
- XCC can infer user-defined TIE instructions
 - Instructions that are a chain (fusion) of other operations
 - Eg: New TIE instruction that is “add-shift” can be inferred wherever there is a add followed by shift (with the same input formats)
- XCC automatically finds instruction-level parallelism in C/C++ code and packs operations into VLIW (FLIX) bundles
 - Uses information about VLIW slots created in user-defined TIE
- XCC automatically vectorizes C/C++ code
 - Maps operations in C/C++ code to SIMD (Vector) functional units
 - Automatically infers XPRES-generated TIE and Vectra DSP TIE

- XPRES automatically generates TIE instructions based on application profiling
- Implemented as an extension of XCC
- Uses TIE inference, VLIW operation bundling, and SIMD/Vector mapping abilities of XCC
- Adds the ability to create a variant of an instruction with a narrow encoding
 - Specialized instruction has a limited immediate range

- All new Tensilica submissions to EEMBC use no assembly
 - TIE instructions are either inferred or called using C intrinsics
 - Compiler automatically extracts parallelism in C/C++ code for VLIW configurations
 - Eg: Diamonds 570T, 330HiFi, 545CK
- HiFi2 audio codecs are implemented completely in C
 - MP3 decoder on HiFi2 requires **5.7 Mhz** on average for 128kbps stereo MP3s
 - Without writing any assembly code!

Xtensa C/C++ Compiler (XCC): EEMBC Confirms XCC Benefits

- Comparison: EEMBC “out of box” scores
 - 2001: Xtensa Configuration w/ Xtensa-III GNU C Compiler (GCC)
 - 2002: Xtensa Configuration w/ Xtensa-V XCC Compiler

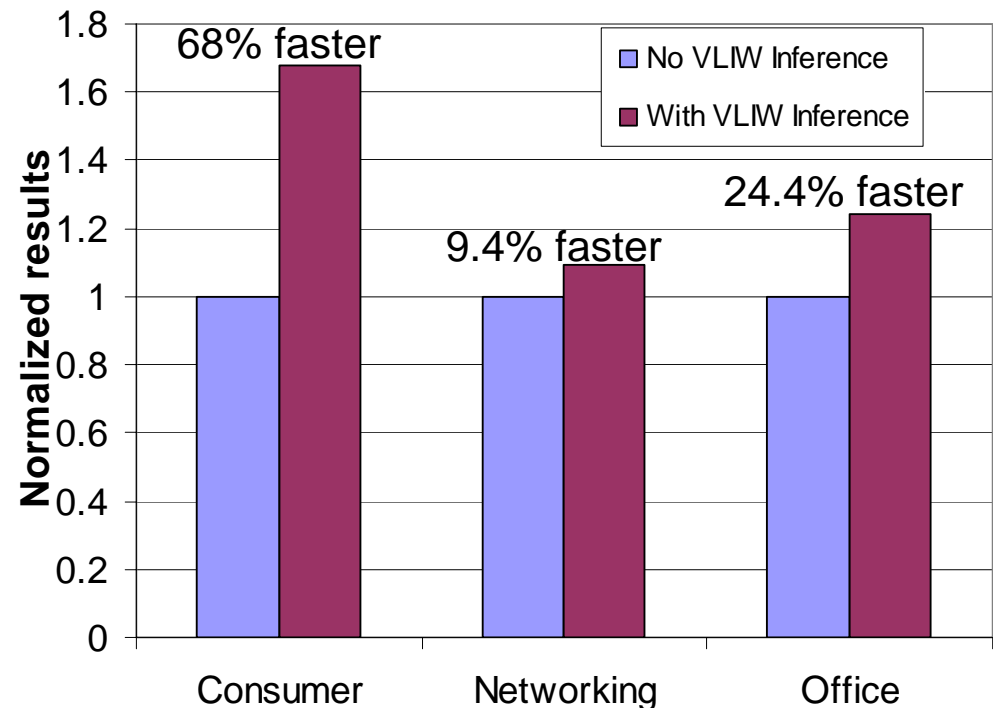


- Binaries compiled by XCC are >50% faster than gcc binaries
- Comparisons of Open64-based compilers by PathScale and HP OpenCC (for x86) against gcc confirm these results

XCC: Automatic Mapping of C/C++ Code to VLIW Instructions

- XCC extracts instruction-level parallelism from C/C++ code and maps to VLIW instructions
- No code modification required
- VLIW technology enables higher performance without increasing Mhz and power

Results for XCC's ability to automatically map C/C++ code to VLIW Instructions



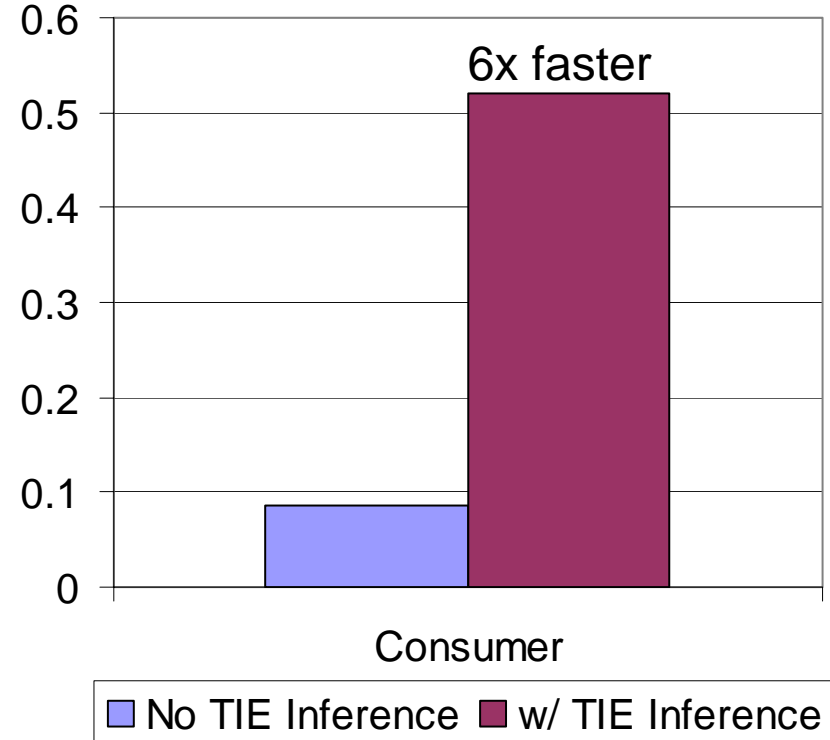
Comparisons on EEMBC Benchmarks



XCC Automatically Infers a Large Set of TIE Instructions

- Only one data point available for apples-to-apples comparison
- These results are a combination of XCC inferring
 - vector instructions
 - VLIW instructions
 - Operation fusions
- No intrinsics required
- No assembly coding required

Results for XCC's ability to automatically infer TIE instructions from C/C++ code



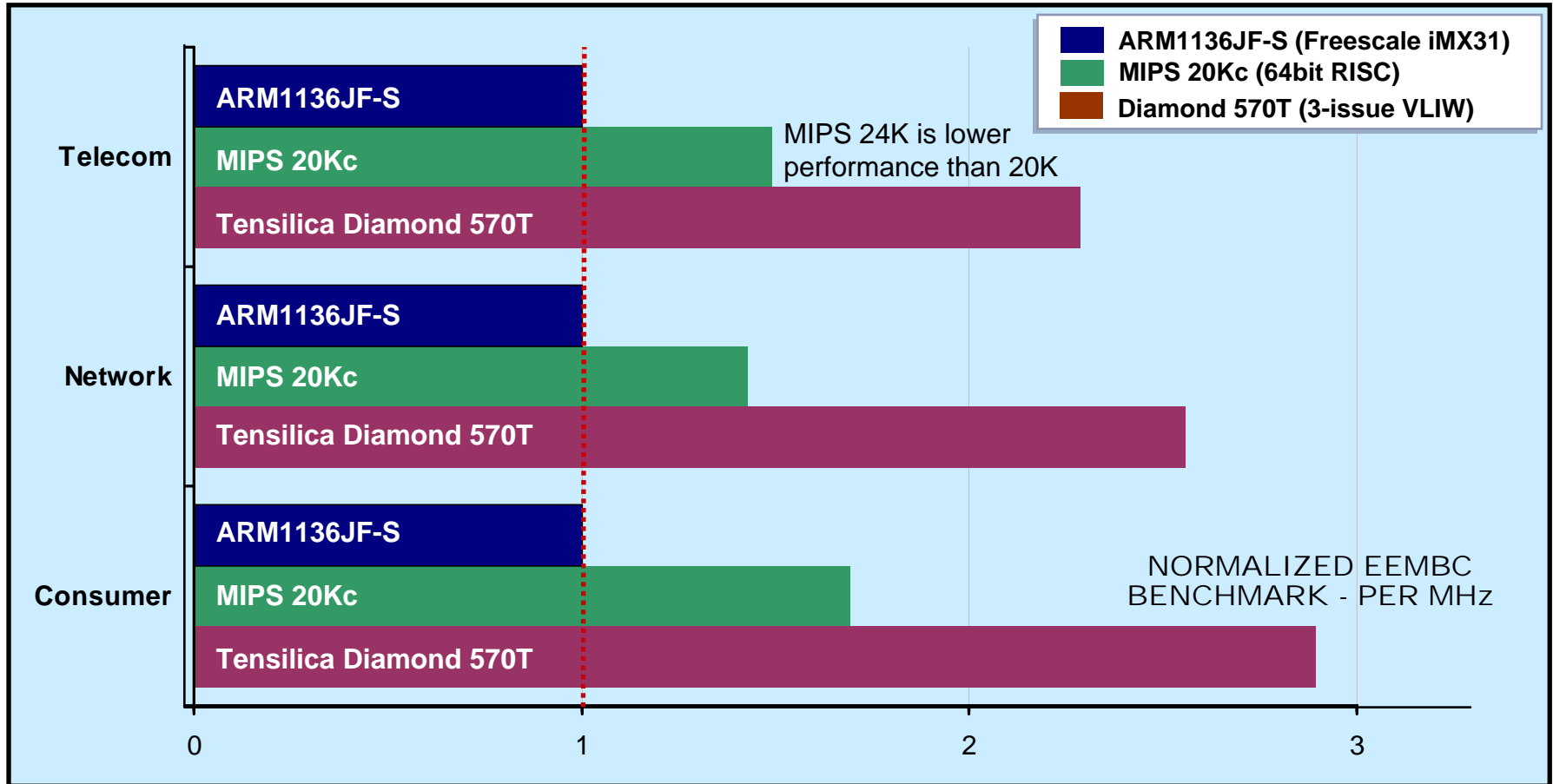
Comparisons on EEMBC Consumer Benchmark



Diamond 570T: 3-issue VLIW

EEMBC Benchmark Results: Out of the box results

XCC extracts parallelism from C code and maps to VLIW instructions



All scores are Simulations of Licensable cores.

All scores are EEMBC/ECL Certified. All scores "out of the box" except Xtensa with TIE (optimized)

Per-MHz certified benchmark scores normalized to ARM = unit score of 1 for suitability in graphing.

Competitive Data as of June 2006. Source: www.eembc.org



- XCC is based on the world-class Open64 compiler framework
- Optimizing compiler with advanced parallelizing and loop optimizations
- Benchmarks confirm XCC's superiority over GCC