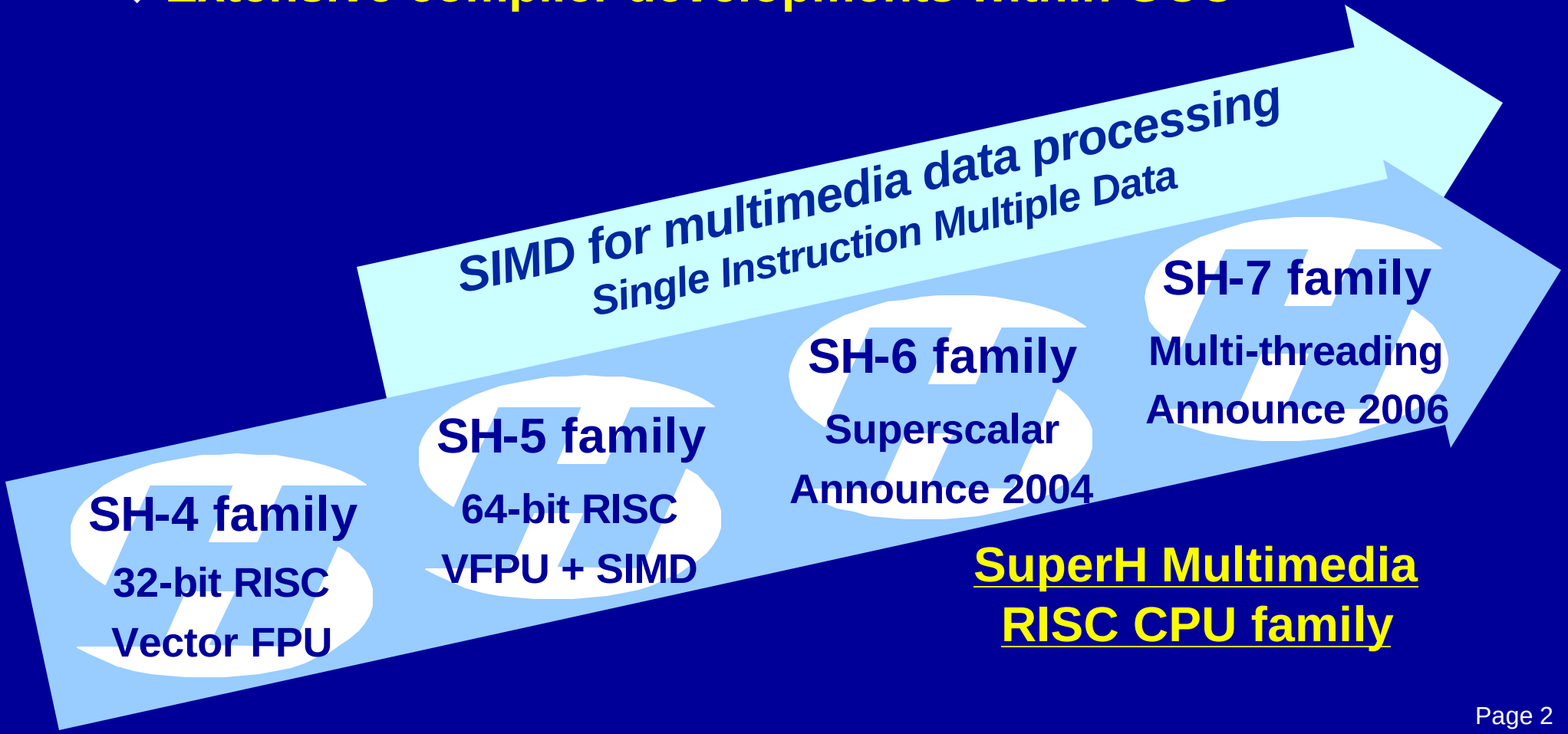
Using Intrinsics to Exploit the Multimedia 8-way SIMD Instructions in SH-5

Speaker: Andy Sturges, Director of IP, SuperH, Inc.

andy.sturges@superh.com
www.superh.com

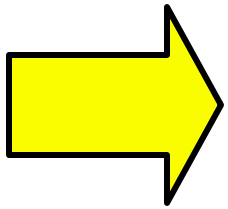
SuperH, Inc. Introduction

- Licenses RISC CPU cores and development tools
  - Extensive compiler developments within GCC



Building Applications with SIMD

- SIMD instructions:
  - Not well supported by standard industry compilers
  - Not automatically generated
  - Assembly language inserts
- No type checking
- No portability
- Overhead caused by compiler



SIMD support required for mainstream application use

'*Ideal*' SIMD Compiler

- Assembler level performance with no overhead
- Full language support for targeting SIMD
- Rapid optimization of existing 'C' code
  - Minimal changes to original code
  - Full benefit of compiler optimizations
- As portable as standard 'C' code

- Ultimately: automatic vector generation from C

What are Intrinsics?

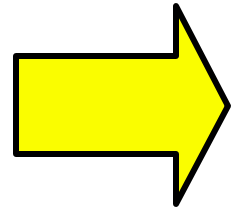
Functions built-in to the compiler to describe instructions

- **SIMD instruction (8-way unsigned 8-bit add with saturation)**
  madds.ub source1, source2, result
- **Intrinsic definition**
  unsigned int64 madds_ub (unsigned int64 source1, unsigned int64 source2)
- **Knowledge of intrinsic via GCC instruction pattern**
  (define_insn "usaddv8qi3"[…match_operands…]" madds_ub %1,%2,%0"..)
- **Vector definition**
  vector unsigned char madds_ub (vector unsigned char,vector unsigned char)

GCC Vector Extensions

- Introduced in 1998 to GCC technology
  - Extends C language to describe vector objects
  - Initially supported ALTIVEC (PowerPC) SIMD architecture
  - Other architectures now supported
    - e.g. TI C6X, SuperH and MMX / x86

Vector C ideal for SIMD



- But still no automatic vectorizations!

GCC Vector Extensions Deliver

- Performance
  - SIMD instructions fully described to the compiler as intrinsics
  - Results in no run-time overhead in use of intrinsics
- Language
  - Interface to SIMD instructions forces use of correct vector types
  - Vector types extended through all of the C language
  - Provides natural interface to SIMD instructions
- Portability
  - SIMD programs can be ported across architectures with compiler support (e.g. detection of different vector widths)
  - First portable language for SIMD programming
  - SuperH investigating MMX to SH-5 translation
  - Can define standard SIMD API

GCC Vector SH-5 SIMD Intrinsics

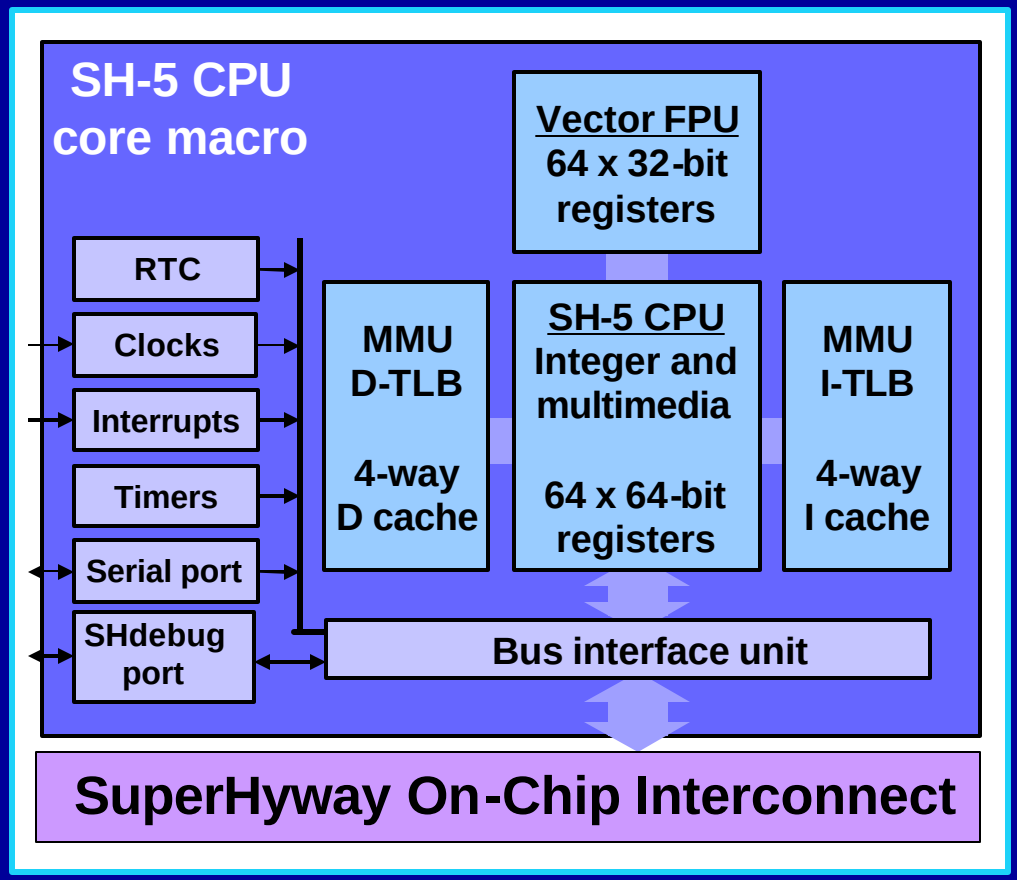
- SH-5 fully implements the GCC vector extensions
- Intrinsic provided for each SIMD operation
- ABI defined functional interface to SIMD instructions
- Forces use of correctly typed operands to SIMD operations
  - Important where SIMD instructions perform subtle type conversions
- Fully type checked and optimized by compiler
  - Compiler performs unrolling, in-lining and scheduling
  - Compiler takes care of register allocations and load/store
- Returned to open source
  - Other SIMD architectures can exploit technology

Detailed Example: MPEG4 Encode

- Objective
  - Using public source code demonstrate the ability to run MPEG4 encode in software on the SH-5 CPU core
- SuperH have used the ‘Xvid’ publicly available codec (see http://www.xvid.org)
- SuperH optimized the encoder using SHmedia SIMD intrinsics for the SH-5 64-bit CPU

SH-5: 64-bit Multimedia CPU

- Integrated RISC and DSP
  - 1.5 DMIPS / MHz
  - 7 MFLOPS / MHz
- Exploits 64-bit registers for
  - 2,4,8-way SIMD delivers
    - 24 MOPS / MHz
- CPU core 3mm^2 in 0.13u
  - Complete hard macro 12mm^2
  - 400MHz



1st Embedded 64-bit SIMD CPU

- SH-5 support for video and DSP data types:
  - 8x8 / 4x16 / 2x32-bit, signed/unsigned/fractional, saturate/modulo
  - 64 high accuracy accumulators (64-bit general purpose registers)
- SH-5 set of SIMD operators:
  - Multiplies (four 16x16=32 per cycle): Mul, MulAcc, MulAdd, MulSub
  - Arithmetic: Add, Sub, Abs, Compare, Convert
  - Rearrangement: Shift, Shuffle, Permute, Bitwise Cmove, Extract
  - Motion estimation: Sum of Absolute Differences 9.6GOPS at 400MHz
- Ideally suited to exploiting GCC vector intrinsics

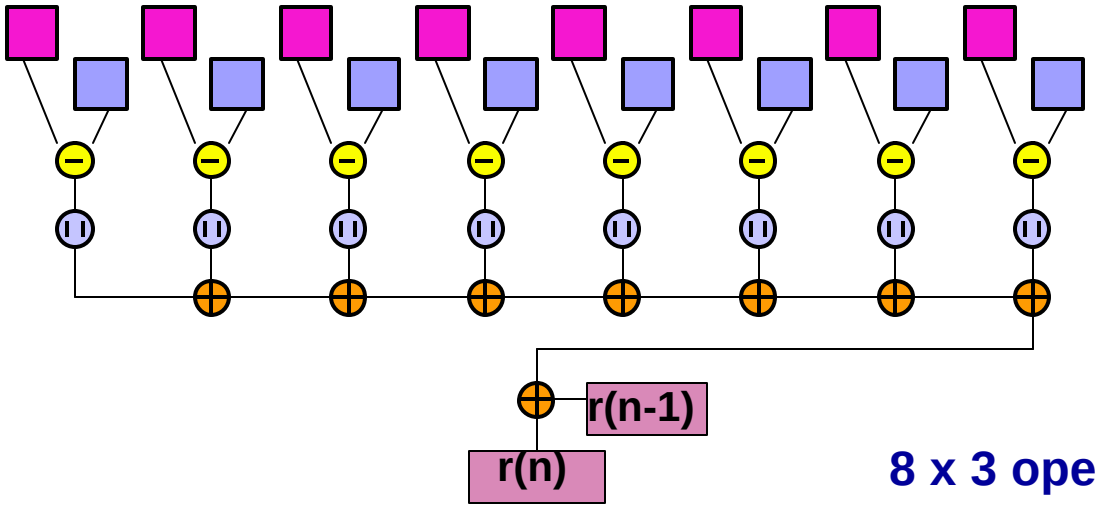
MPEG4: Motion Estimation C Code

Main inner loop of XVID MPEG4 motion estimation:

```
uint32_t sad8_c(const uint8_t * const cur,  const uint8_t
   * const ref, const uint32_t stride)
{
   uint32_t sad = 0, i, j;
   uint8_t const *ptr_cur = cur;*ptr_ref = ref;

   for (j = 0; j < 8; j++) {
       for (i = 0; i < 8; i++)
             sad += ABS(*(ptr_cur + i) - *(ptr_ref + i));
       ptr_cur += stride; ptr_ref += stride;
   }
   return sad;
}
```

Motion Estimation Optimized

```
for (j = 0; j < 8; j++) {
   vector char cur = *(vector char*)ptr_ref;
   vector char ref =
      (vector char)sh_media_unaligned_LD_Q((uint8_t*)ptr_ref,0);

   sad = sh_media_MSAD_UBQ(cur, ref, sad);

   ptr_cur +=stride; ptr_ref +=stride;
}
```



Optimized using SH-5 MSAD instruction

MSAD.UBQ Rm, Rn, Rw
Sum of absolute differences of unsigned 8-bit

8 x 3 operation/cycle x 400MHz = 9.6 GOPS

Comparison of Compiler Output

- Original
  - Generates 122 instructions
  - Takes 976 cycles

```
...
ld.ub    r41,0,r38
ld.ub    r40,0,r39
add.l    r19,r21,r19
add.l    r17,r21,r17
sub.l    r38,r39,r37
add.l    r37,r63,r7
sub      r63,r7,r8
cmpgt    r63,r7,r9
cmvne    r9,r8,r7
...
```

X 64

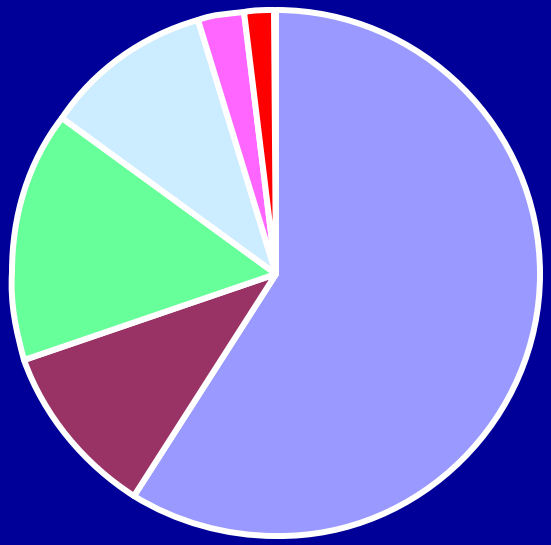
- Optimized
  - Generates 52 instructions
  - Takes 59 cycles

```
...
or         r60,r61,r21
ldhi.q     r43,0,r6
ldlo.q     r20,0,r7
addi.l     r8,7,r9
ld.q       r20,0,r23
msad.ubq r22,r21,r36
...
```

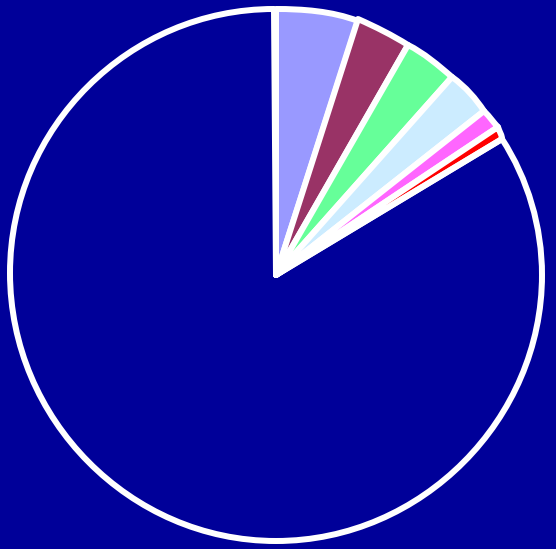
X 8

MPEG-4 Encode Result

Unoptimized



SIMD Optimized
>80% improvement

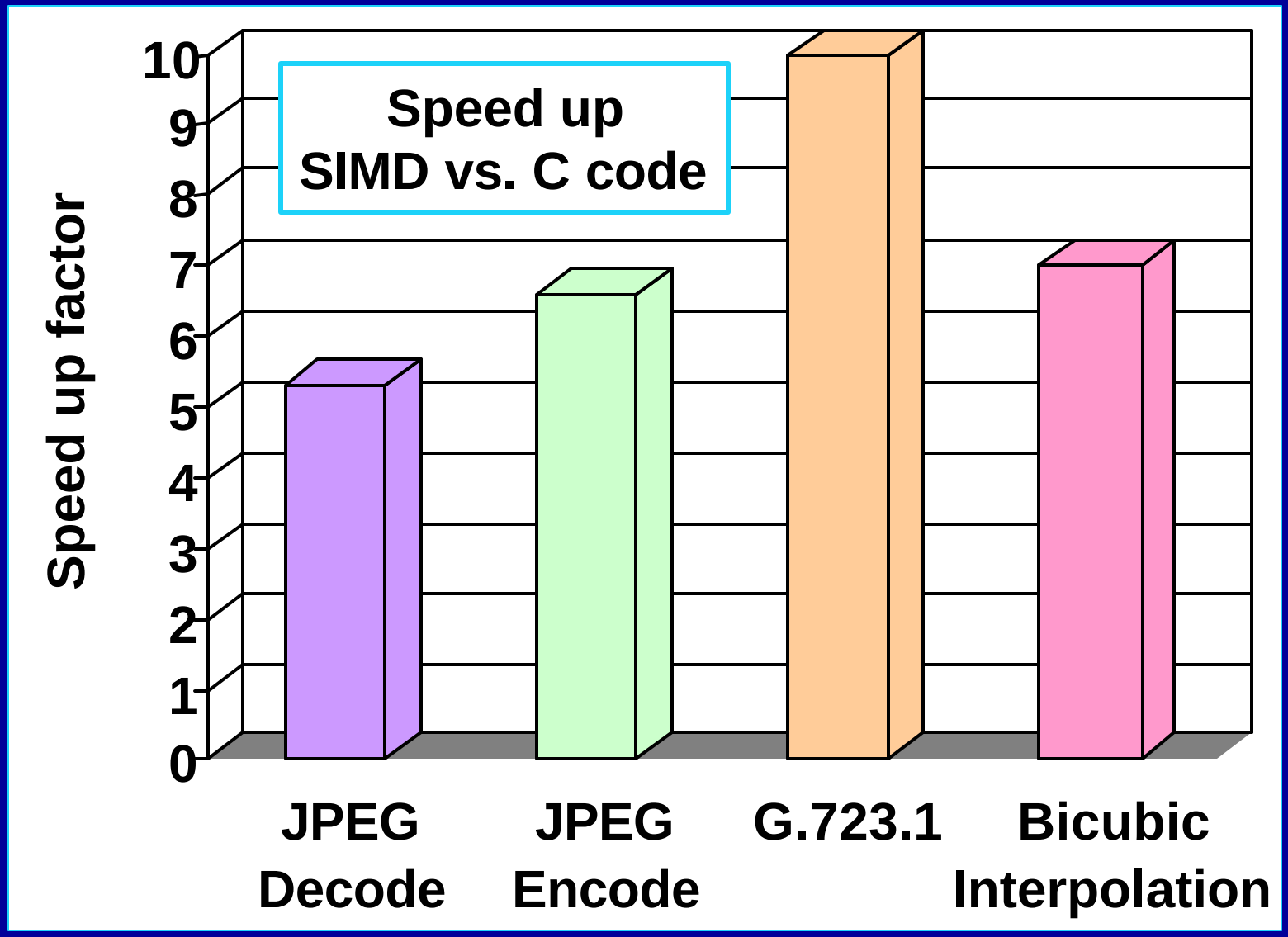


SH-5 Performance	QCIF 64kbps, 15fps	CIF 384bps, 25fps	VGA 1mbps, 25fps
Unoptimized	97.9MHz	771.1MHz	n/a
Optimized	18.3MHz	146MHz	398MHz

Benchmarking

- CPU performance is combination of
  - Architecture and compiler
- EEMBC provides a mechanism for evaluating SIMD
  - ‘Out of box’ and ‘full fury’
- SH-5 ‘full fury’ (optimized) benchmark will be published later this year using GCC vector intrinsics

Examples of 64-bit SIMD Intrinsic



JPEG using EEMBC® benchmarks. To be certified in 2H 2003.

Optimizing SH-5 Cores

- Future SH-5 targeting handheld multimedia devices
  - Dynamic power: clock gating and low power datapath elements
  - Power management: dynamic frequency and voltage control
  - Static power: mixed Vth technology and ‘unit’ shutdown
  - Target: 0.30mW/MHz at 300MHz

- Delivers very low power consumption for mobile multimedia

Xvid MPEG-4 encode power requirements		
Profile	QCIF, 64kbs, 15fps	CIF, 384kbs, 25fps
Power	5.5mW	44mW

SH-6 Plans

- SH-6 will deliver even higher performance:
  - With GCC Vector extensions SIMD code will be portable to SH-6
    - Used existing applications to optimize microarchitecture
  - Super-scalar optimized for the embedded market
    - Dual issue 8-way SIMD
    - Deeper pipeline for higher clock frequency
  - New SIMD instructions for specific applications, e.g.
    - H.264
    - JPEG 2000
    - Cryptography
    - Networking

Summary

- GCC vector extensions enable effective programming model for SIMD
  - This has been demonstrated on a variety of applications
  - Proven in EEMBC benchmarks
- SH-5 8-way SIMD delivers high architecture performance
  - Replaces dual CPU and DSP solutions
- SH-6 can fully exploit all existing SH-5 optimized code:
  - Full binary compatibility
  - Re-compile C code for maximum performance