Northeastern
UNIVERSITY

Multimedia Macros for Portable Optimized Programs

Juan Carlos Rojas
Miriam Leeser
Northeastern University
Boston, MA

Programming Multimedia Architectures

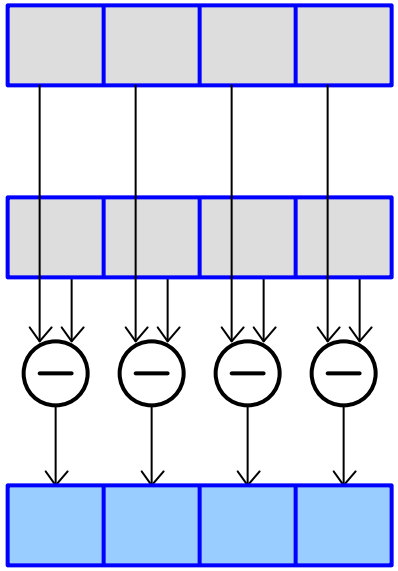
- Many different Multimedia Architectures
  - TriMedia®, AltiVec™, MMX™, SSE, SSE2
  - Different complex parallel instructions
  - Different register lengths, data types, ...
- Goals for programming
  - Portability
    - Write application once
    - Translate to any architecture
  - High Performance
    - Make best use of each architecture

Multimedia Architectures

- Partitioned registers

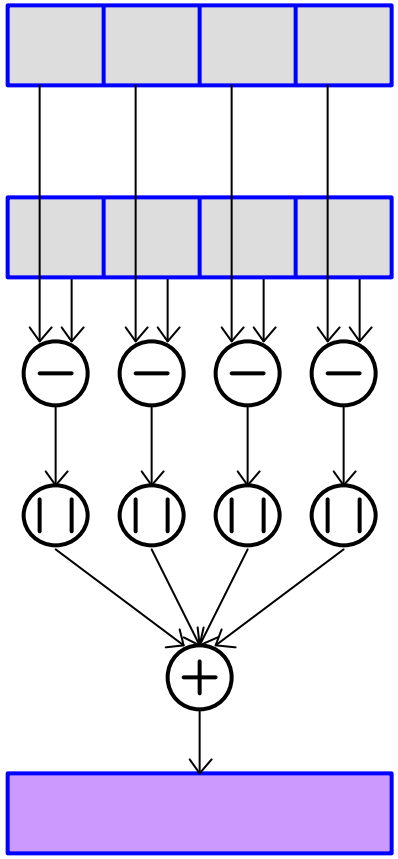
Multimedia Architectures

- Partitioned registers
- Parallel operations



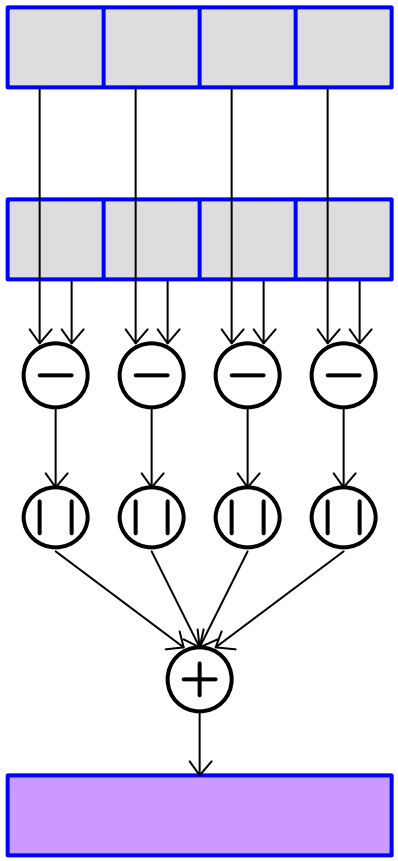
Multimedia Architectures

- Partitioned registers
- Parallel operations
- Complex instructions

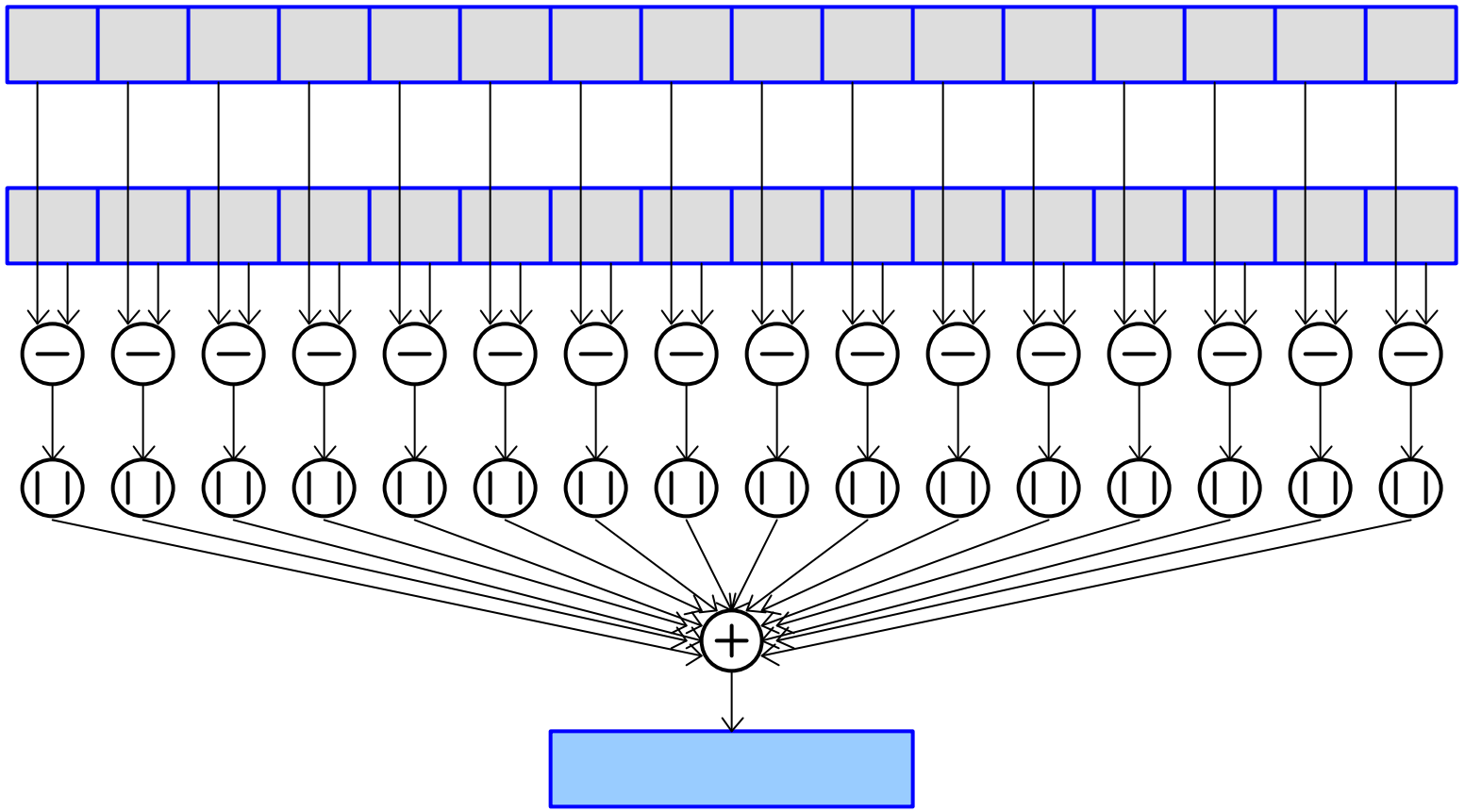


Multimedia Architectures

- Partitioned registers
- Parallel operations
- Complex instructions
- Architectures differ in
  - Register lengths
  - Instruction sets
  - Alignment requirements
  - Programming styles



Example: Vector SAD

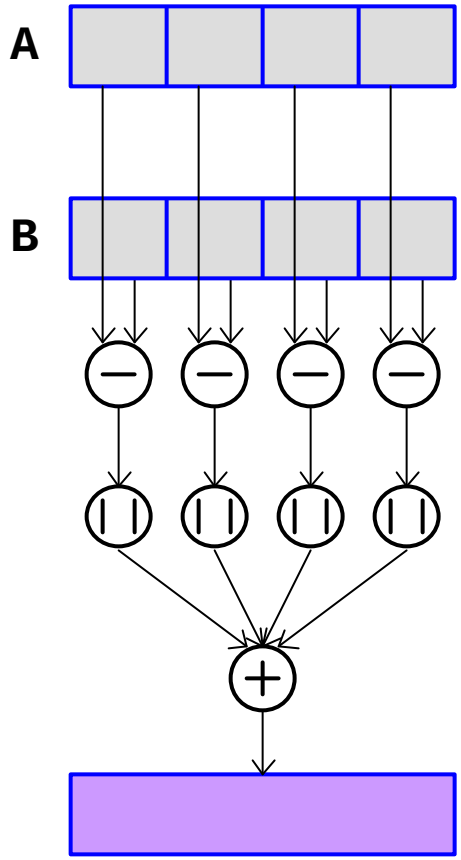


Vector SAD: Scalar Implementation

```
uint8 *a, *b;
int diff, sad;

sad = 0;
for (i=0; i<16; i++)
{
    diff = a[i] – b[i];
    sad += diff > 0 ? diff : -diff;
}
```

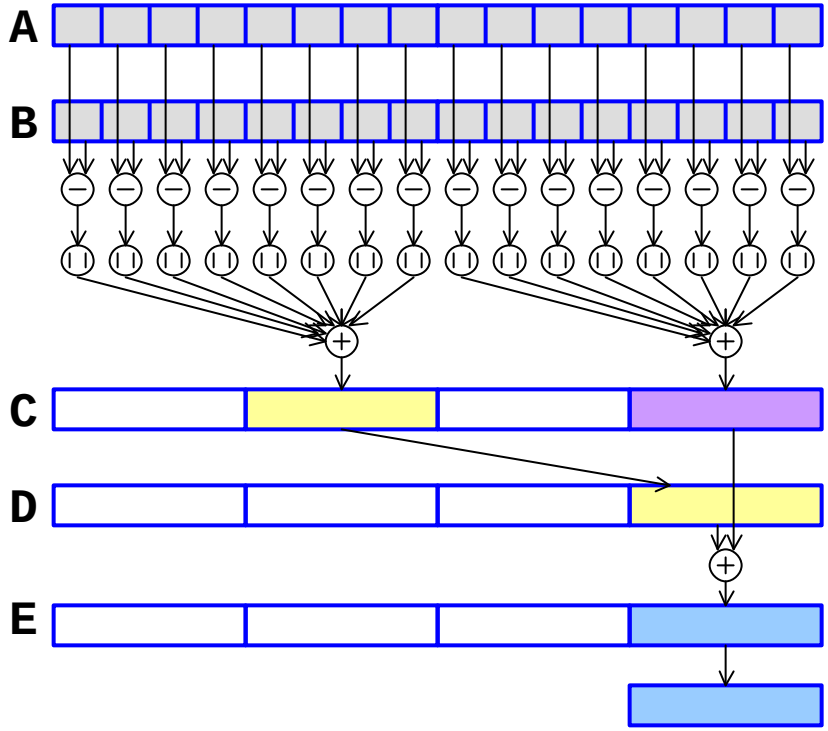
Vector SAD: Optimized for TriMedia®

```
uint8 *a, *b;
int A, B, sad;

A = *((int *) a);
B = *((int *) b);
sad = UME8UU(A, B);
A = *((int *)(a+4));
B = *((int *)(b+4));
sad += UME8UU(A, B);
A = *((int *)(a+8));
B = *((int *)(b+8));
sad += UME8UU(A, B);
A = *((int *)(a+12));
B = *((int *)(b+12));
sad += UME8UU(A, B);
```



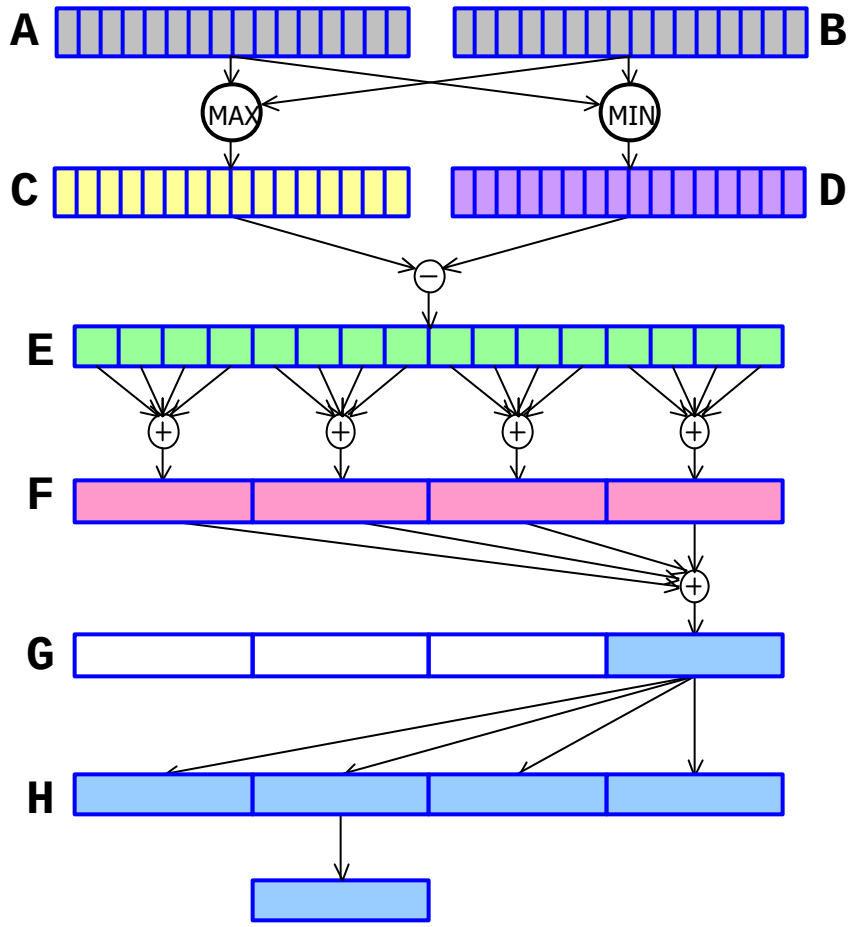
Vector SAD: Optimized for SSE2

```
uint8 *a, *b;
__m128i A, B, C, D, E;
int sad;

A = _mm_load_si128(
        (__m128i *) a);
B = _mm_load_si128(
        (__m128i *) b);
C = _mm_sad_epu8(A, B);

D = _mm_srli_si128(C, 8);
E = _mm_add_epi32(C, D);
sad = mm_cvtsi128_si32(E);
```



Vector SAD: Optimized for AltiVec™

```
uint8 *a, *b;
vector uint8 A, B, C, D;
vector uint8 E, F, G, H;
int sad;

A = vec_ld((vector uint8 *) a);
B = vec_ld((vector uint8 *) b);
C = vec_min(A, B);
D = vec_max(A, B);
E = vec_sub(C, D);

F = vec_sum4s(E);
G = vec_sums(F);

H = vec_splat(G, 3);
vec_ste(H, &sad);
```

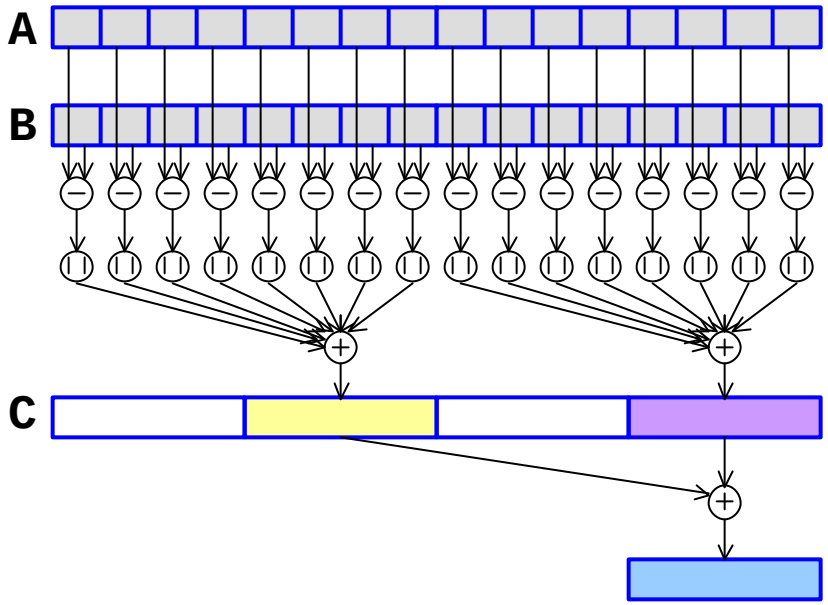


Solution: MMM

- MMM: MultiMedia Macros
- Instruction-level macro library
- Common virtual instruction set
- Emulation of long registers
- Emulation of instructions

Vector SAD: MMM Version

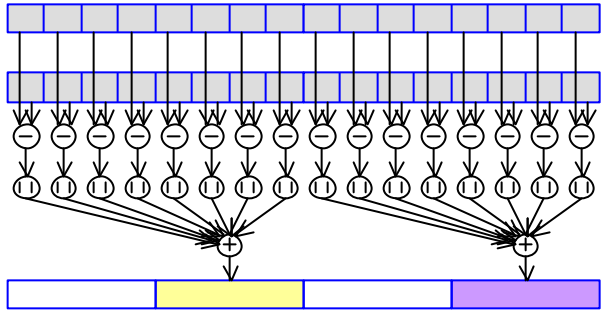
```
uint8 *a, *b;
DECLARE_U8x16(A)
DECLARE_U8x16(B)
DECLARE_U32x4(C)
int sad;

LOAD_A_U8x16(A, a)
LOAD_A_U8x16(B, b)
SAD2_U8x16(C, A, B)
SUM2_U32x4(sad, C)
```



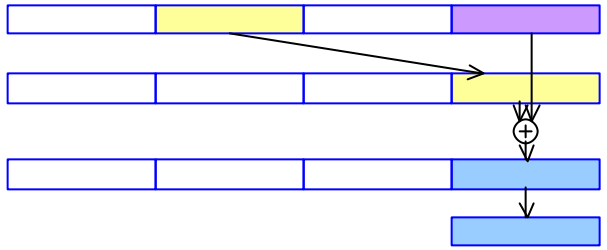
MMM definitions for SSE2

```
#define DECLARE_U8x16(var) \
    __m128i var;

#define LOAD_A_U8x16(var, ptr) \
    var = _mm_load_si128((__m128i *) (ptr));
```

```
#define SAD2_U8x16(dst, src1, src2) \
    dst = _mm_sad_epu8(src1, src2);
```



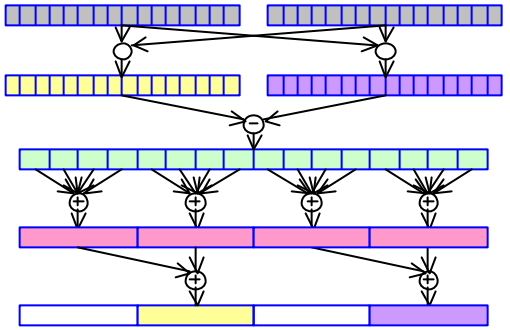
```
#define SUM2_U32x4(dst, src) \
    dst = _mm_cvtsi128_si32( \
          _mm_add_epi32(src, \
          _mm_srli_si128(src, 8)));
```



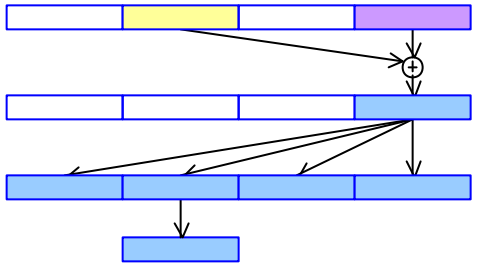
MMM definitions for AltiVec™

```
#define DECLARE_U8x16(var) \
    vector UINT8 var;

#define LOAD_A_U8x16(var, ptr) \
    var = vec_ld((vector UINT8 *)(ptr));
```

```
#define SAD2_U8x16(dst, src1, src2)    \
    dst = vec_sum2s(vec_sum4s(         \
          vec_sub(vec_max(src1, src2), \
                  vec_min(src1, src2))));
```

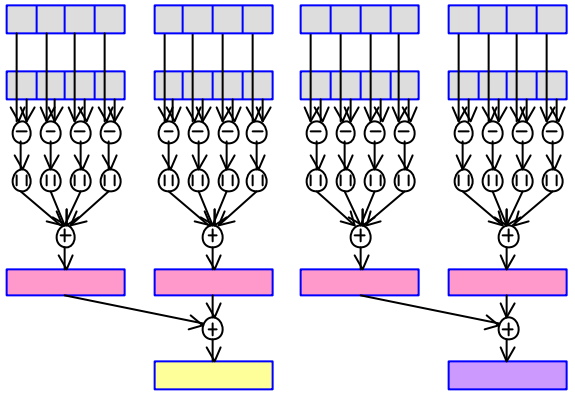


```
#define SUM2_U32x4(dst, src) \
    vec_ste(vec_splat(       \
            vec_sums(src), 3), &dst);
```



MMM definitions for TriMedia®

```
#define DECLARE_U8x16(var) \
    unsigned int var##_0;  \
    unsigned int var##_1;  \
    unsigned int var##_2;  \
    unsigned int var##_3;

#define LOAD_A_U8x16(var, ptr)      \
    var##_0 = *((int *) (ptr));     \
    var##_1 = *(((int *)(ptr))+1); \
    var##_2 = *(((int *)(ptr))+2); \
    var##_3 = *(((int *)(ptr))+3);

#define SAD2_U8x16(dst, src1, src2)         \
    dst##_0 = UME8UU(src1##_0, src2##_0)+ \
              UME8UU(src1##_1, src2##_1); \
    dst##_2 = UME8UU(src1##_2, src2##_2)+ \
              UME8UU(src1##_3, src2##_3);

#define SUM2_U32x4(dst, src) \
    dst = src##_0 + src##_2;
```



Other Approaches

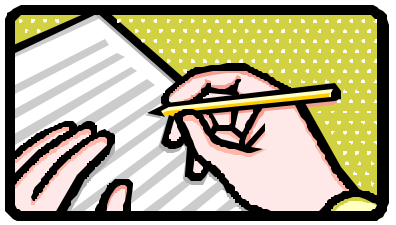
- Parallelizing compilers
- Optimized kernel libraries
  - BLAS, Intel® IPP, VSIPL
- Data-parallel languages
  - Fortran 90, SWARC, Vector Pascal
  - C++ SIMD classes
- Automatic code generators
  - SPIRAL, FFTW, ATLAS
- None of these approaches achieves performance and flexibility of MMM
  - MMM makes use of complex instructions in each ISA

MMM Advantages

- General solution

- Complex applications

- Complex partitioned instructions
- Hand-coded performance



Our Approach

- Study representative architectures:
  - TriMedia® TM1300 – 32-bit registers
  - MMX™+SSE – 64-bit integer registers
  - SSE2 – 128-bit integer registers
  - AltiVec™ – 128-bit registers
- Define common virtual instruction set
- Implement MMM libraries for each target
- Implement portable applications in MMM
- Measure performance, compare to reference implementations

Common Virtual Instruction Set

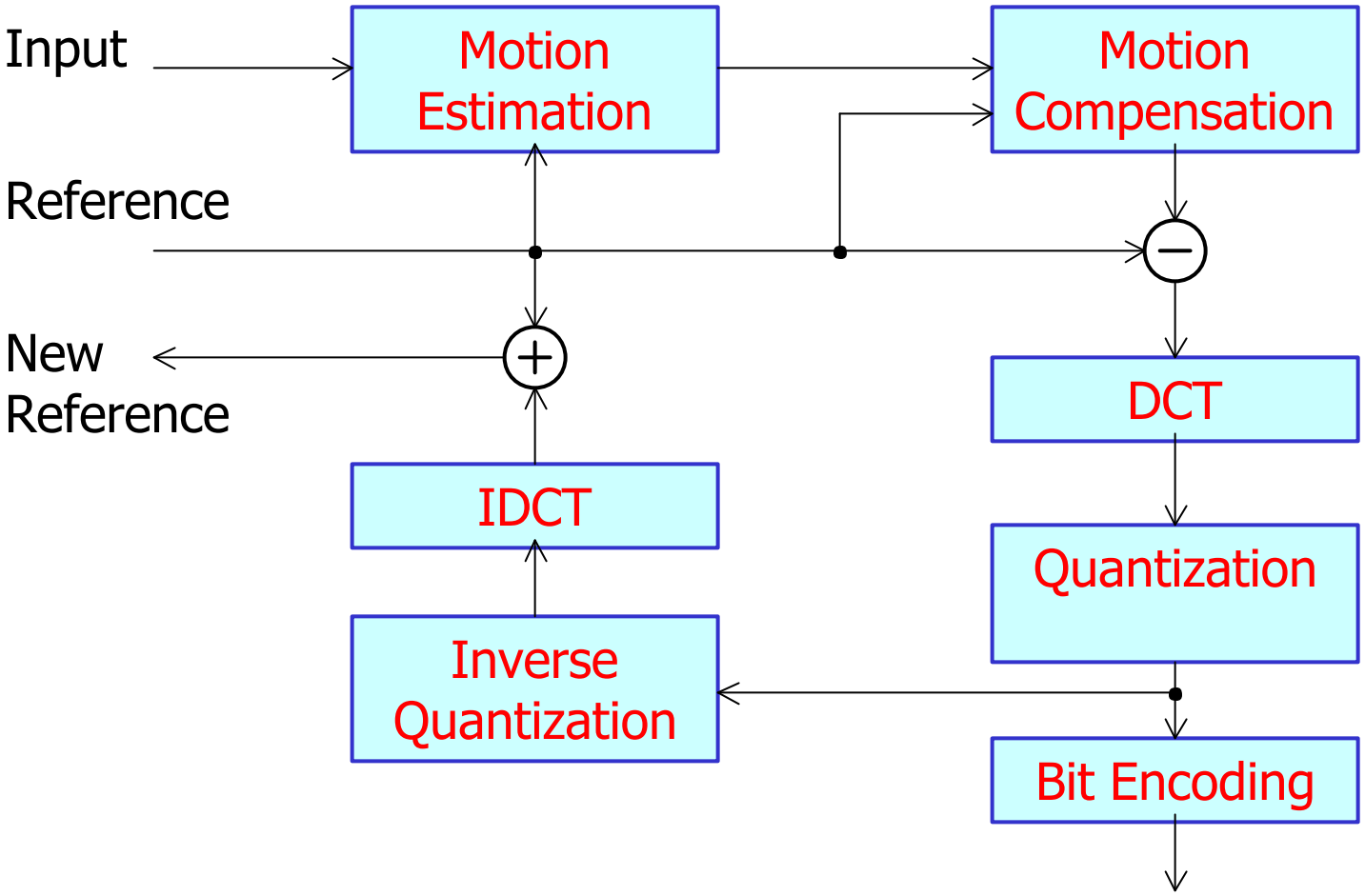
Vector types	**I8x16 U8x16 I16x8 U16x8** **I32x4 U32x4 F32x4**
Vector declarations	**DECLARE DECLARE_CONST**
Set	**SET SET1 CLEAR COPY**
Load and store	✍ Aligned ✍ Unaligned
Rearrangement	**INTERLEAVE BROADCAST PERMUTE**
Type conversion	**PACK EXTEND CVT** ✍ Integer <-> float ✍ Vector <-> scalar

Common Virtual Instruction Set

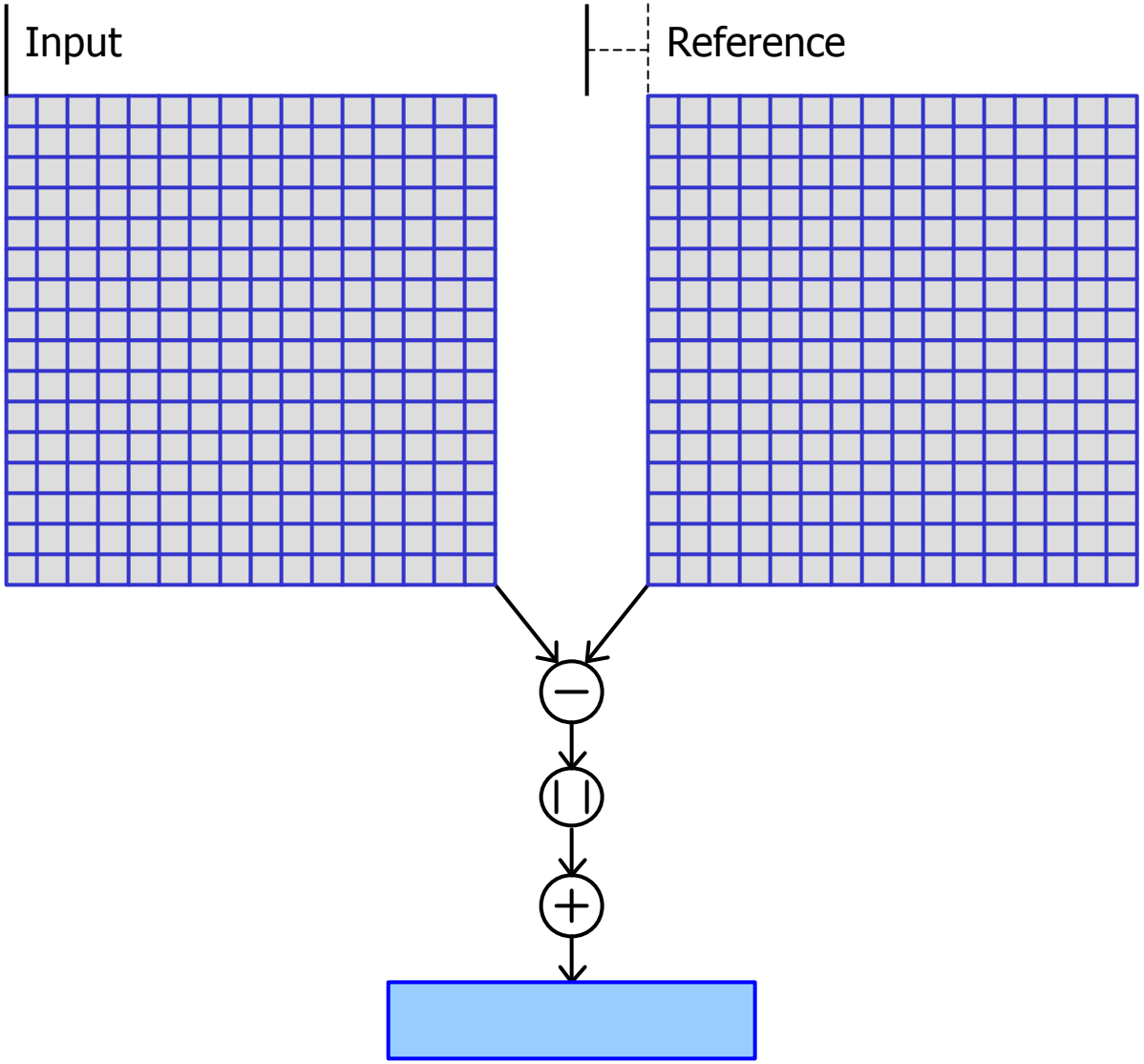
Shift	**SLL SRL SRA ROL** **SLL_I SRL_I SRA_I ROL_I**
Bit-wise logic	**AND ANDN OR XOR SEL**
Comparison	**CMP_EQ CMP_LT CMP_GT**
Float arithmetic	**ADD SUB MULT MULT_ADD** **DIV MIN MAX SQRT**
Integer arithmetic	**ADD SUB AVG MIN MAX** **MULT_H MULT_L MULT_ADDPAIRS** **SAD2 SUM2** Handling of overflow: - Modulo - Saturation - Unspecified

Example Programs

MPEG2 Video Encoder



16x16 Block L_1-Distance



16x16 Block L_1-Distance

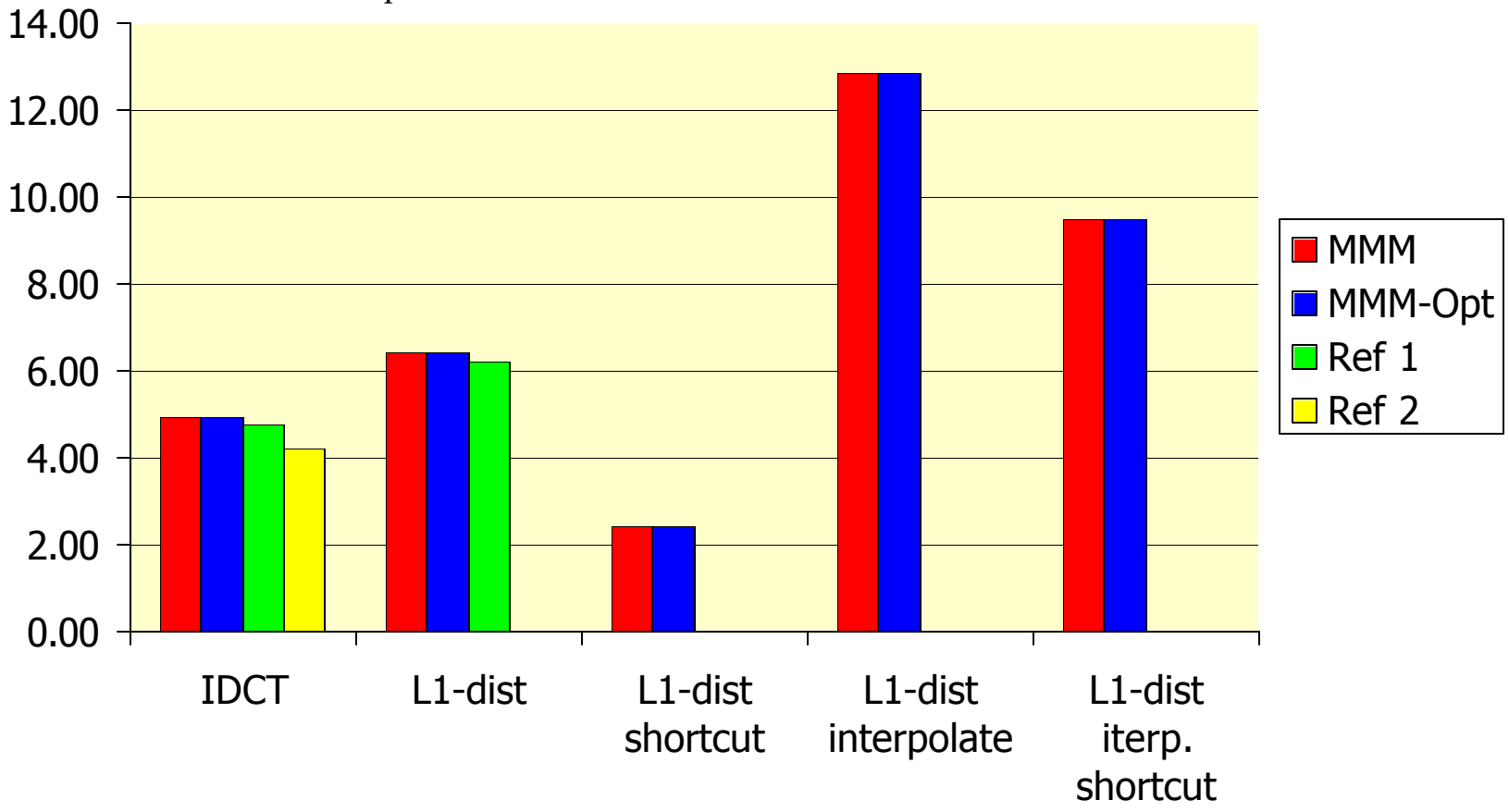
```
DECLARE_U8x16(R1)
DECLARE_U8x16(I)
DECLARE_U32x4(Sad)
UINT32 Sum;
CLEAR_U32x4(Sad)
PREPARE_LOAD_ALIGNMENT(1, pRef)
SAD_ROW(Sad, pRef + 0*RowPitch, pIn + 0*RowPitch, 1)
:
SAD_ROW(Sad, pRef +15*RowPitch, pIn +15*RowPitch, 1)
SUM2_U32x4(Sum, Sad)

#define SAD_ROW(dst, pRef, pIn, index)        \
    LOAD_U_U8x16(R1, pRef, index)             \
    LOAD_A_U8x16(I, pIn)                      \
    SAD2_ADD_M_U8x16(dst, R1, I, dst)
```

Example Reference Versions

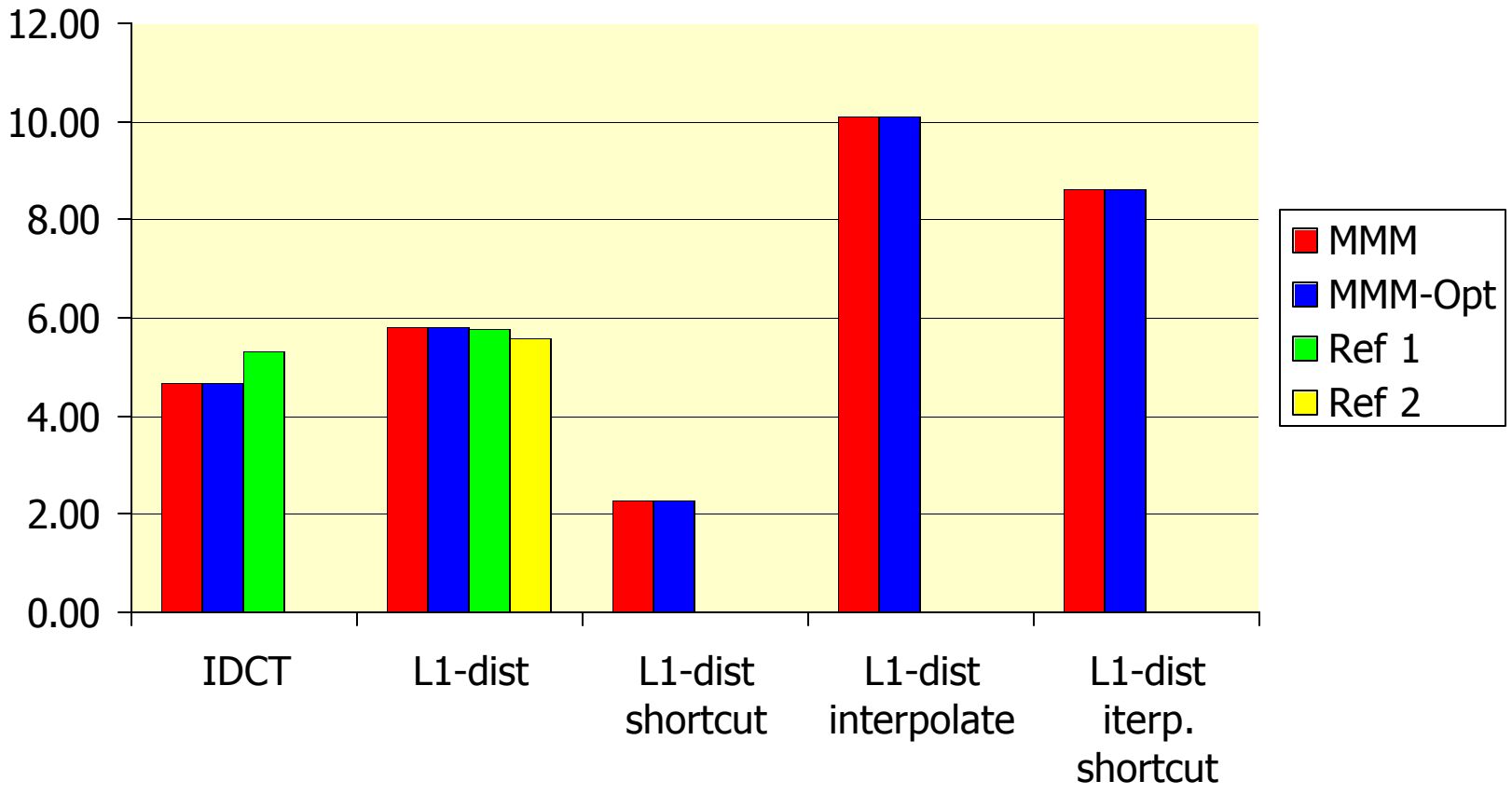
	IDCT	L_1-Distance
TriMedia®	Case study	
MMX™+SSE	Assembly	Assembly C + intrinsics
SSE2	Assembly C++ vector classes	C + intrinsics
AltiVec™	C + intrinsics	C + intrinsics

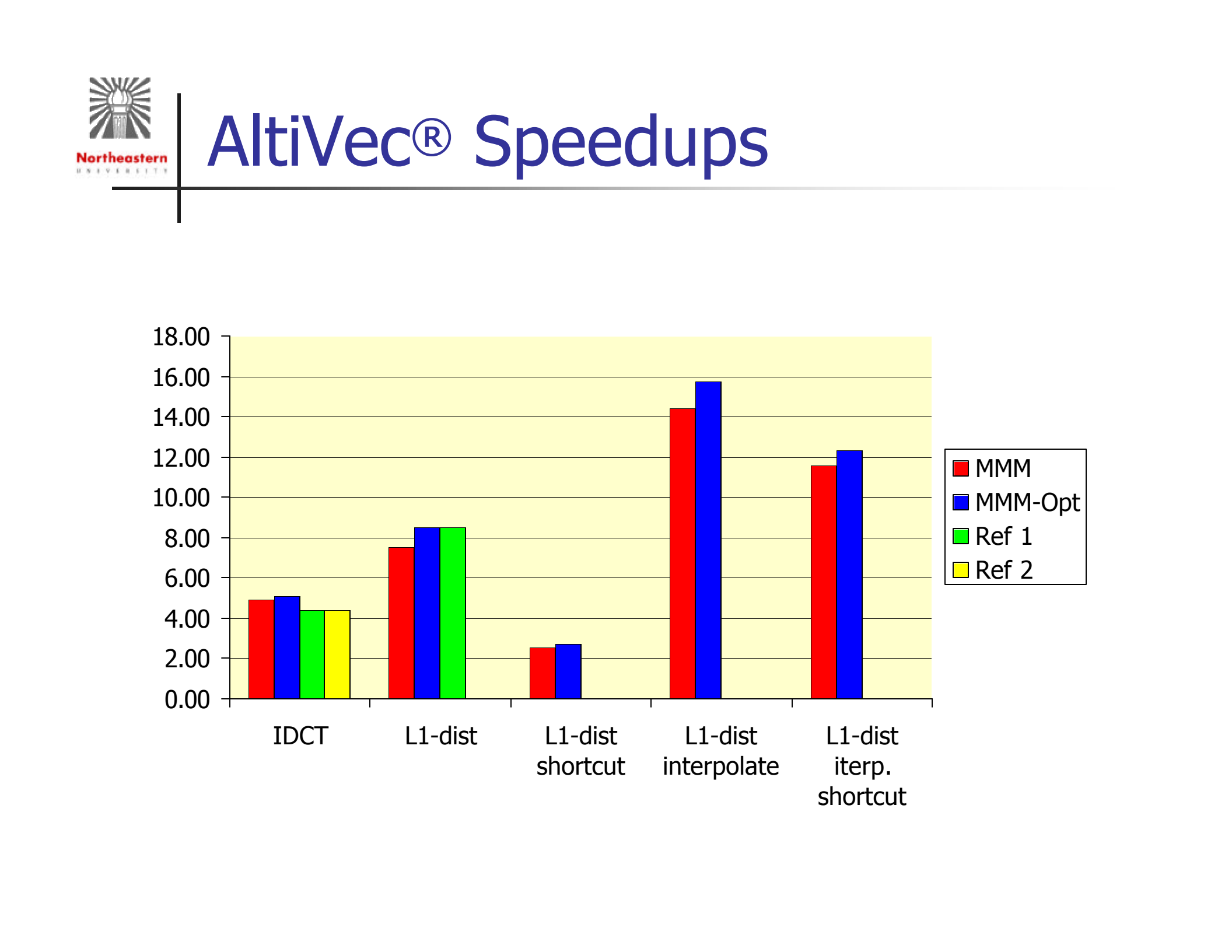
SSE2 Speedups

$$Speedup\ ?\ \frac{Time_{Scalar}}{Time_{Optimized}}$$



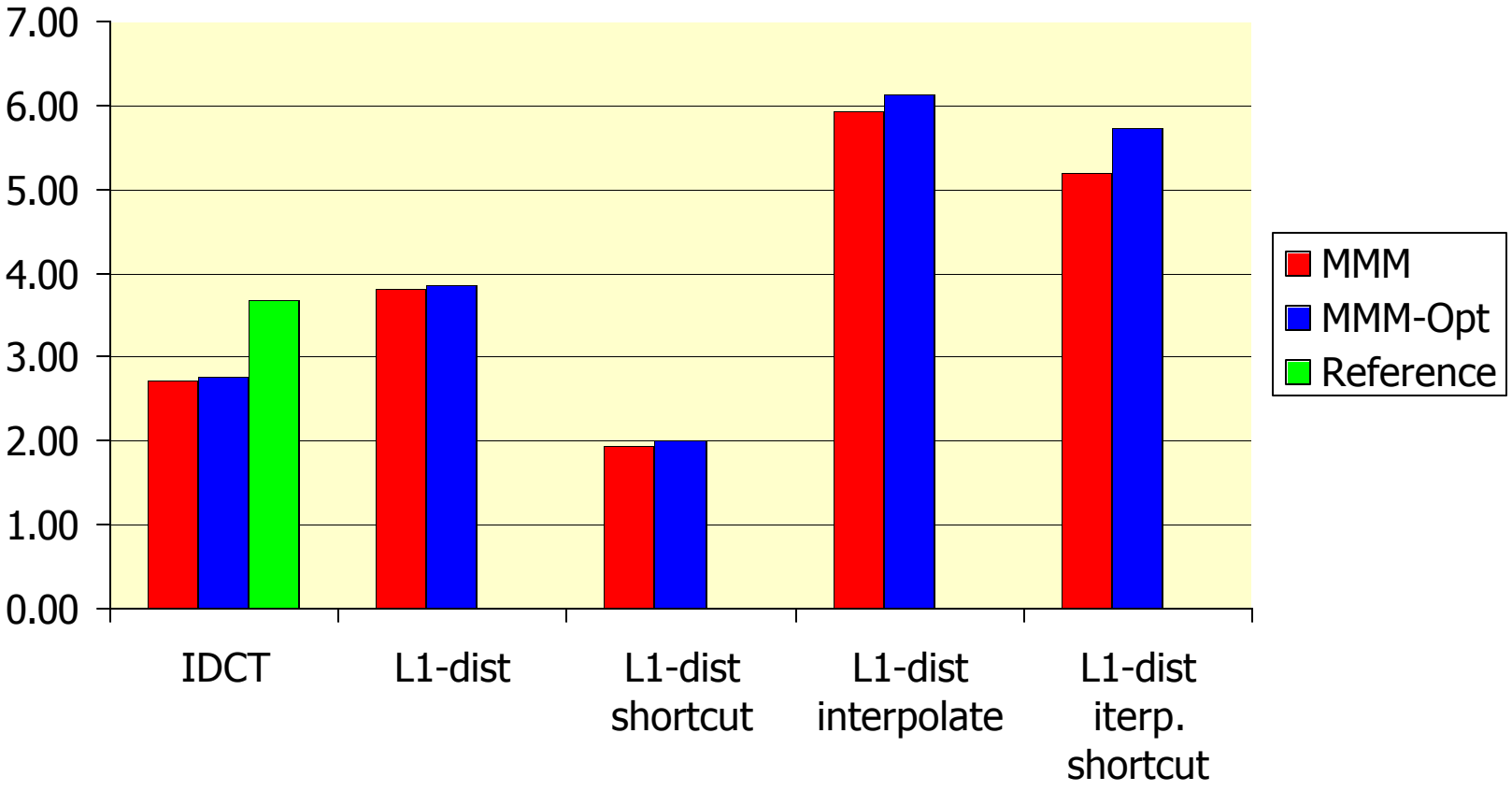
MMX™+SSE Speedups





AltiVec® Speedups

TriMedia® Speedups



Conclusions and Future Work

- MMM = Portable + Optimized
- Diverse architectures
- Complex examples, complex instructions
- Hand-coded performance
  - Within 12% of best
- Solution can be applied to other ISAs
  - SIMD & DSP
- Future Work:
  - Address ease of programming issues
    - MMC: Multimedia C

Vector SAD: MMC Version

```
uint8 *a, *b;
u8x16 A, B;
u32x4 C;
int sad;

A = *a;
B = *b;
C = SAD2(A, B);
sad = SUM2(C);
```