A Second Generation SIMD Microprocessor Architecture

Mike Phillip

Motorola, Inc.

Austin, TX

phillip@ncct.sps.mot.com

Why SIMD ?

- "But isn't that really old technology ?"
  - Yes, architecturally speaking, but...
  - SIMD within registers provides "cheap" interelement communictaions that wasn't present in prior SIMD systems
- Provides a reasonable tradeoff between increased computational bandwidth and manageable complexity
  - Fewer register ports needed per "unit of useful work"
  - Naturally takes advantage of stream-oriented parallelism
- Can be easily scaled for various price/performance points
- Can be applied in addition to traditional techniques

Some Alternatives

- **Wider superscalar machines**

  Benefits: High degree of flexibility; easy to program

  Problems: Complex implementations; higher power consumption

- **Special purpose DSP and Media architectures**

  Benefits: Low power; efficient use of silicon

  Problems: Traditional design approaches tend to limit CPU speeds; lack of generality; lack of development tools

- **VLIW**

  Benefits: Reduced implementation complexity; impresses your friends

  Problems: Need VERY long words to provide sufficient scalability; limited to compile-time visibility of parallelism; difficult to program

The First Generation

- First generation implementations tend to overlay SIMD instructions on existing architectural space
  - Permitted quick time to market, but limits scalability
  - Intel, Cyrix, AMD (MMX), SPARC (VIS) are fairly complete
  - PA-RISC (MAX), Alpha (MVI) offer limited graphics acceleration instructions only
- Programming models are weak or non-existent
  - Most code to date written in assembly language
  - Language extensions and compilers just now appearing
- Little support for control flow & memory management

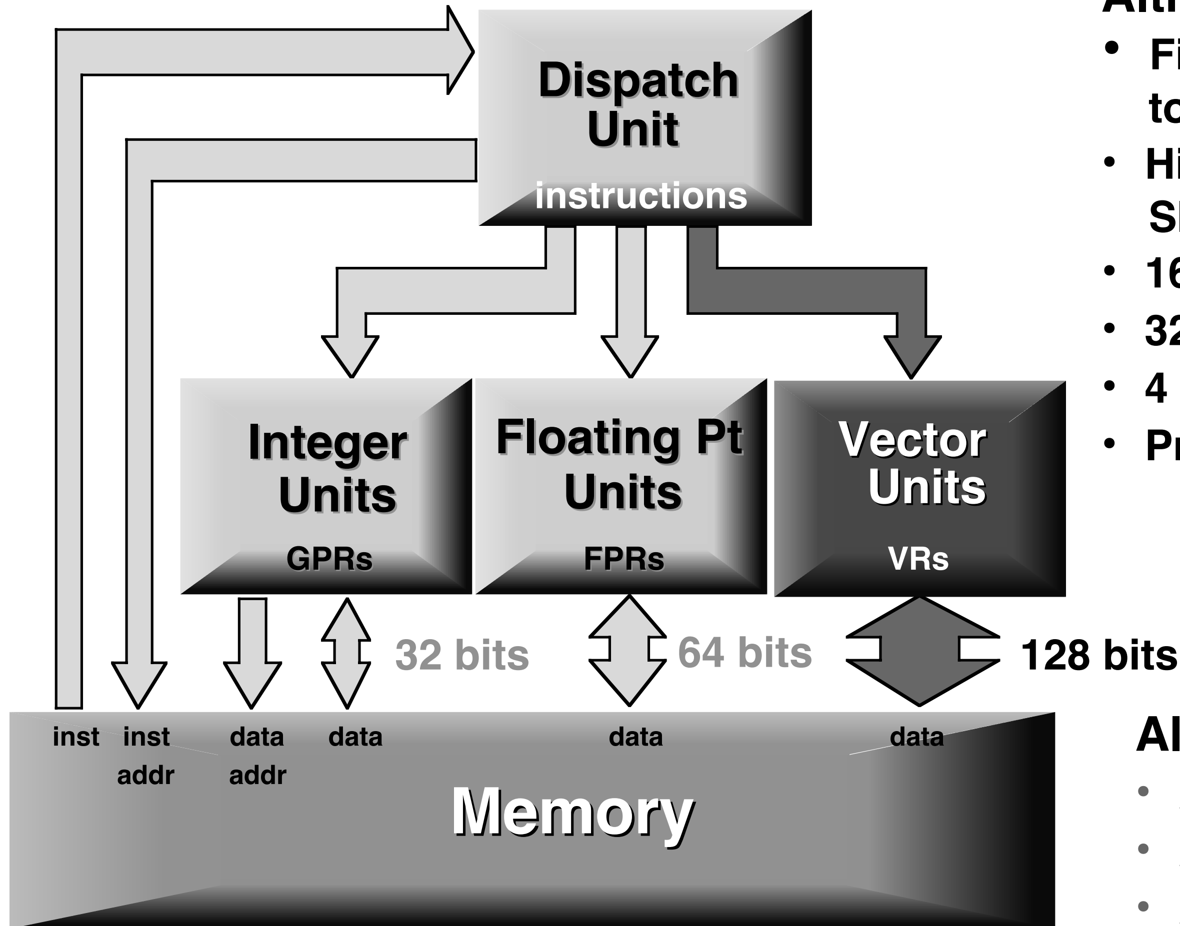
Nonetheless, significant speed-ups seen for multimedia applications

"Wouldn't It Be Nice..."

How might next generation "general purpose" SIMD be improved ?

- Independent register file and name space
- More parallelism
- More DSP-like capabilities
- More "neighbor" operations (inter vs. intra - element)
- Orthogonal element data types (including FP)
- Better control of memory hierarchy
- Better SIMD control flow capability
- Better programming model

AltiVec Overview



AltiVec is:

- First significant extension to PowerPC architecture
- High performance, scalable SIMD architecture
- 162 new instructions
- 32 new registers (128 bits each)
- 4 new integrated vector units
- Programmable from C and C++

AltiVec is not:

- An execution mode
- An on-chip coprocessor
- A hardware accelerator
- A Digital Signal Processor

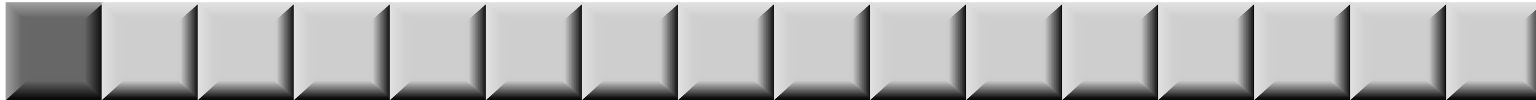
AltiVec: The Basics

- **Simplified architecture**
  - No interrupts other than data storage interrupt on loads and stores
  - No hardware unaligned access support
  - No complex functions
  - Streamline the architecture to facilitate efficient implementation
- **4-operand, non-destructive instructions**
  - Supports advanced “multiply-add/sum” and permute primitives
- **Includes key “neighbor” operations, SIMD control flow, data management and DSP-like capabilities**
  - Enables use in networking and communications markets
- **True superset of MMX functionality**
  - Delivers 2-4x performance for desktop multimedia applications
  - Twice the parallelism, four to eight times the register bandwidth

AltiVec Data Types

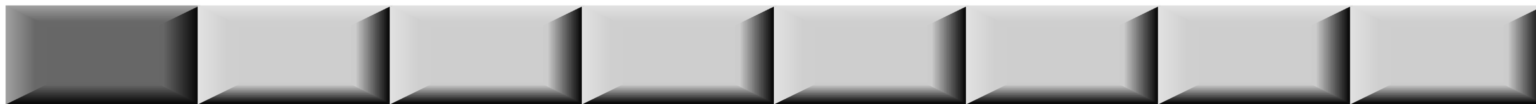
One Vector (128 bits)

8 bits



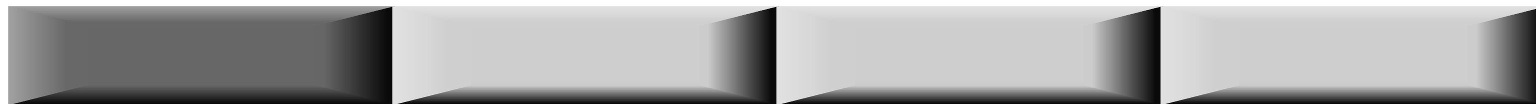
16 signed or unsigned integer bytes

16 bits



8 signed or unsigned integer halfwords

32 bits



4 signed or unsigned integer words or
4 IEEE single-precision floating-point numbers

NCSG Networking & Computing Systems Group

Digital DNA™ From Motorola

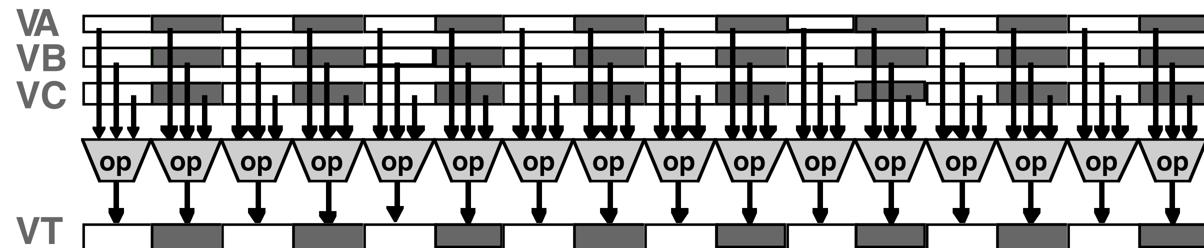
Algorithmic Features

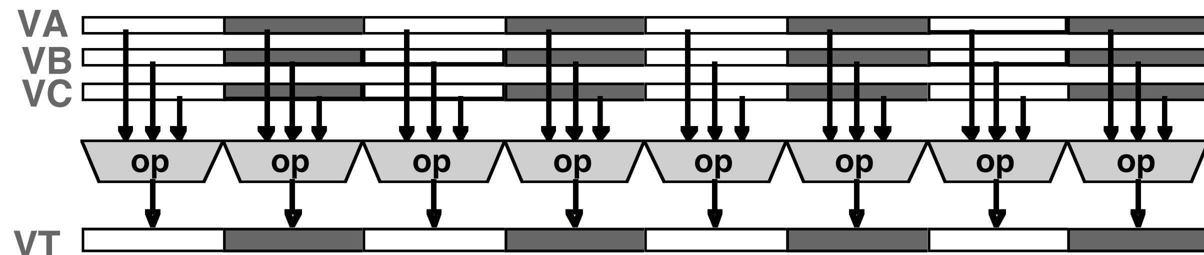
- **Intra-element Instructions**
  - Integer and Floating Point arithmetic
  - Memory access instructions
  - Rotate, Shift and Logical instructions; Min and Max instructions
- **Inter-element Instructions**
  - Permute, multi-register shifts, address alignment
  - Integer Multiply Odd/Even, Multiply-Add, Multiply-Sum
  - Integer Sum Across
- **Control flow**
  - Compare creates field mask used by select function
  - Compare Rc bit enables setting Condition Register
    - Trivial accept/reject in 3D graphics
    - Exception detection via software polling

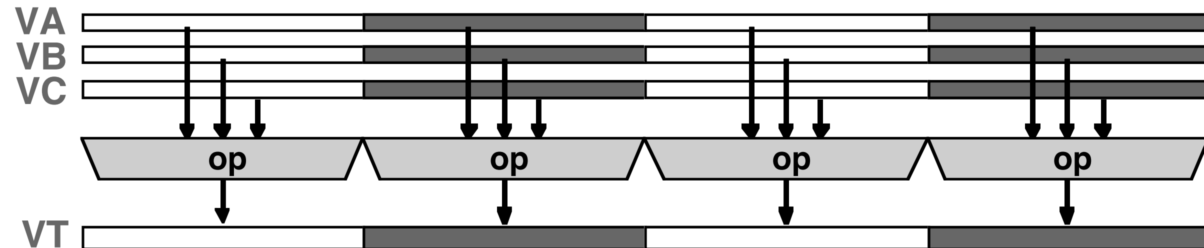
Intra-element Instructions

Operations include:

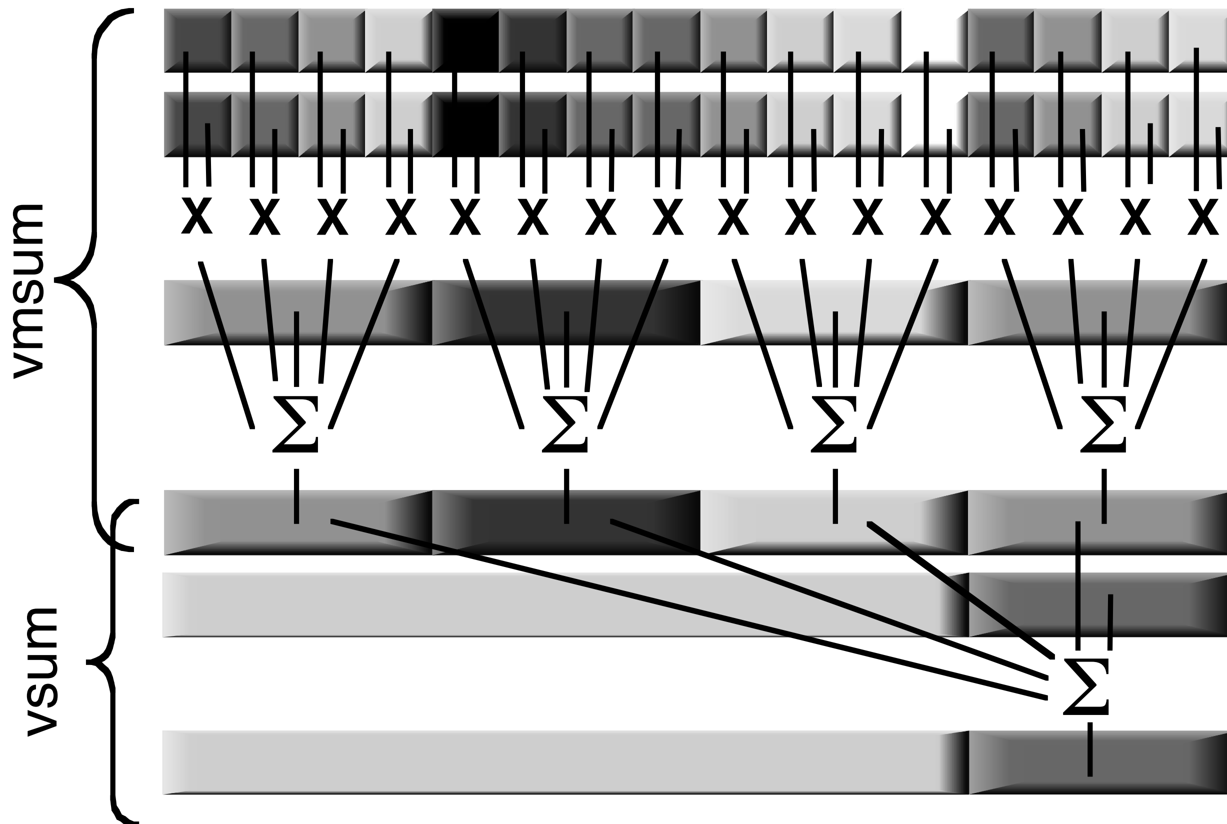
- Integer arithmetic
- FP arithmetic
- Memory access
- Conditional
- Logical
- Shift, Rotate
- Min, Max
- Saturation options







Vector Dot Product (FIR)



PowerPC:

36 instructions

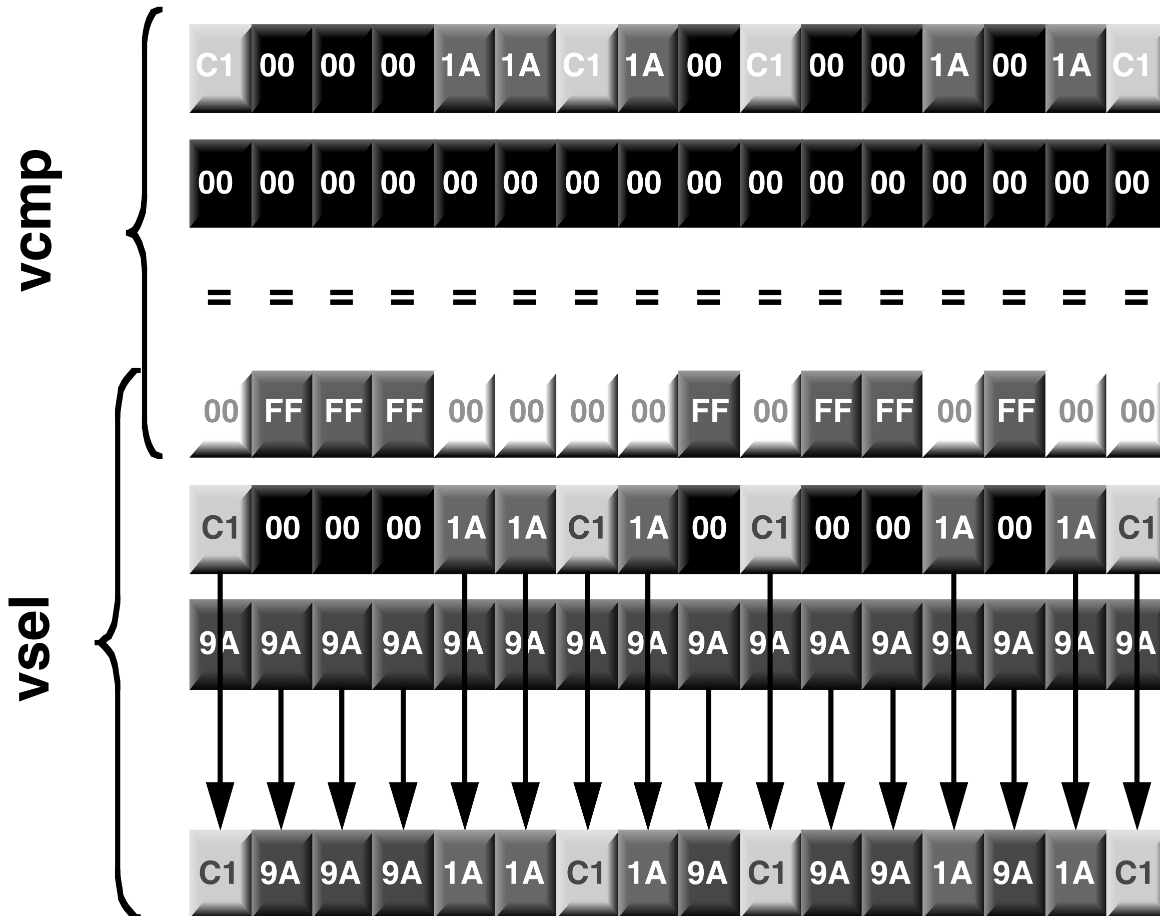
(18 cycles throughput)

AltiVec:

2 instructions

(2 cycles throughput)

Vector Compare and Select



PowerPC:

48 instructions

(32 cycles throughput)

AltiVec:

2 instructions

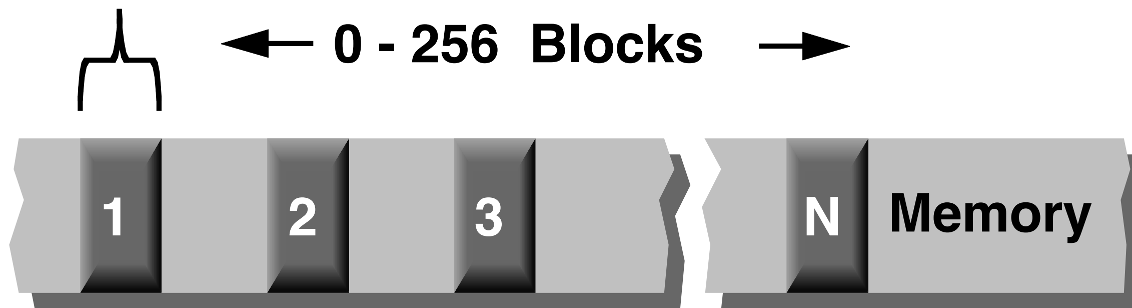
(2 cycles throughput)

Data Management Features

- **Simplified load/store architecture**
  - Simple byte, halfword, word and quadword loads & stores
  - No unaligned accesses; software-managed via permute instruction
- **Data stream prefetch and stop instructions**
  - Enables reuse of data cache as a memory access buffer
  - Alleviates memory access latency by enabling early data prefetch
- **Load & store with LRU and "transient" hints**
  - Marks loaded cache block "next to be replaced"
  - Avoids flushing cache with multimedia data exhibiting limited reuse
  - "Software-managed" memory buffer in cache
- **Permute unit**
  - Full bytewise crossbar
  - Accelerates bit interleaving, table lookups, very long shifts

Data Stream Prefetch

Block Size (0 - 32 Vectors)



Stride (± 32 KBytes)

PowerPC:
Does not exist

AltiVec:
1 instruction
(Can save hundreds of clocks)

Four independent streams

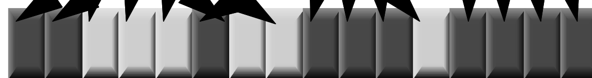
NCSG Networking & Computing Systems Group

Digital DNA™ From Motorola

Vector Permute

Control Vector

Input Vectors



Output Vector

PowerPC:

5-50 instructions

(Depending on application)

AltiVec:

1 instructions

(1 cycle throughput)

NCSG Networking & Computing Systems Group

DigitalDNA™ From Motorola

Programming SIMD

Four options for programming SIMD extensions:

1. Assembly language

- Requires minimal tools support, **but difficult to maintain/port**

2. Libraries and APIs

- Very useful, but **can't provide complete coverage;** required for Java

3. "Synthesize" SIMD code from standard compiled C/C++

- Many SIMD instructions have **no mapping to C or C++**
- **Leads to poor performance;** tends to mislead developers

4. Offer programming model to access SIMD from C / C++

- **Permits developers to more directly express intended algorithm**
- **Can match or exceed assembly language in performance**

NCSG Networking & Computing Systems Group

Digital DNA™ From Motorola

AltiVec Programming Model

1. Introduce new C and C++ data type:

```
vector unsigned short  a, b, c;
vector float  *x, *y, z;
```

2. Introduce intrinsic operators, with type overloading

```
c = vec_add(a,b);   // for i=0..7, c_i = a_i + b_i
z = vec_add(*x,*y); // for i=0..3, z_i = (*x)_i + (*y)_i
```

Compiler selects appropriate instruction, handles register allocation, instruction scheduling, inlining, loop optimizations, etc.

Programmer can focus on algorithm, while still retaining benefits of integrated development environments

Application: FIR Filters

$$y_i = \sum_{j=0}^{K-1} c_j x_{i-j}$$

"Obvious" approach:

- **Compute results horizontally**
- Advantages: **Simple to design, scalable, little register pressure**
- Drawbacks: **Results need to be interleaved, "sum across" reductions needed for large filters**

$y_0 = c_0x_0 + c_1x_{-1} + c_2x_{-2} + c_3x_{-3} + c_4x_{-4} + c_5x_{-5} + c_6x_{-6} + c_7x_{-7}$
$y_1 = c_0x_1 + c_1x_0 + c_2x_{-1} + c_3x_{-2} + c_4x_{-3} + c_5x_{-4} + c_6x_{-5} + c_7x_{-6}$
$y_2 = c_0x_2 + c_1x_1 + c_2x_0 + c_3x_{-1} + c_4x_{-2} + c_5x_{-3} + c_6x_{-4} + c_7x_{-5}$
$y_3 = c_0x_3 + c_1x_2 + c_2x_1 + c_3x_0 + c_4x_{-1} + c_5x_{-2} + c_6x_{-3} + c_7x_{-4}$

Typical performance: **1999 cycles (64 taps, 128 outputs)**

NCSG Networking & Computing Systems Group

DigitalDNA™ From Motorola

FIR Example, continued

A better approach for AltiVec:

- Compute results using vertical “tiles”
- Advantages: Results already in place, no “sum across”
- Drawbacks: More complex design, more register pressure

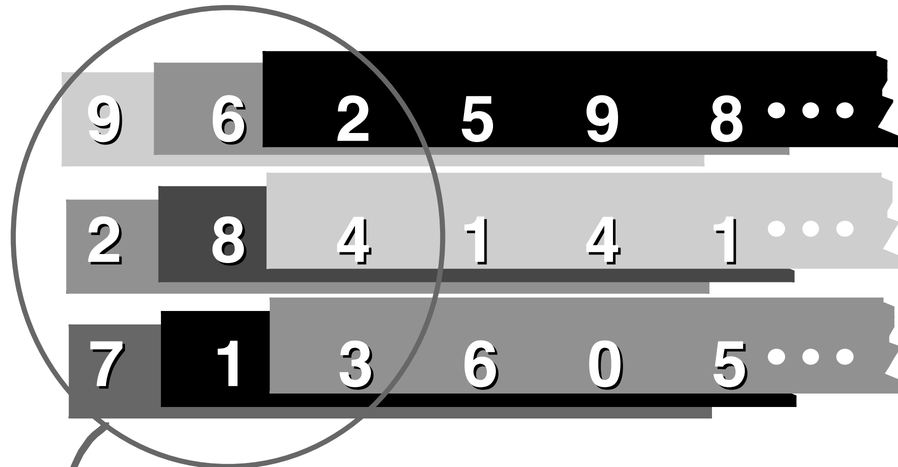
$y_0 =$	$c_0x_0 + c_1x_{-1}$	+	$c_2x_{-2} + c_3x_{-3}$	+	$c_4x_{-4} + c_5x_{-5}$	+	$c_6x_{-6} + c_7x_{-7}$
$y_1 =$	$c_0x_1 + c_1x_0$	+	$c_2x_{-1} + c_3x_{-2}$	+	$c_4x_{-3} + c_5x_{-4}$	+	$c_6x_{-5} + c_7x_{-6}$
$y_2 =$	$c_0x_2 + c_1x_1$	+	$c_2x_0 + c_3x_{-1}$	+	$c_4x_{-2} + c_5x_{-3}$	+	$c_6x_{-4} + c_7x_{-5}$
$y_3 =$	$c_0x_3 + c_1x_2$	+	$c_2x_1 + c_3x_0$	+	$c_4x_{-1} + c_5x_{-2}$	+	$c_6x_{-3} + c_7x_{-4}$

Typical performance: 1566 cycles (64 taps, 128 outputs)
(23 % improvement)

NCSG Networking & Computing Systems Group

DigitalDNA™ From Motorola

Application: 3x3 Median Filter



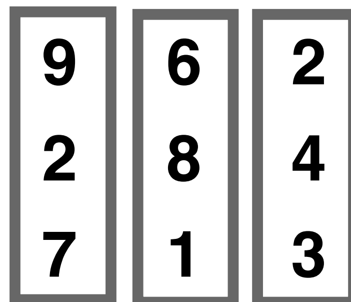
Problem: Traditional scalar algorithms contain control flow

Solution: Find a SIMD algorithm

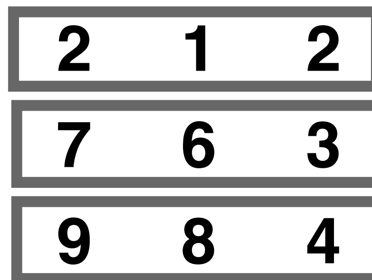
Typical AltiVec Performance:
1.06 cycles / median (pixel)

Load nine vectors (16 elements each)

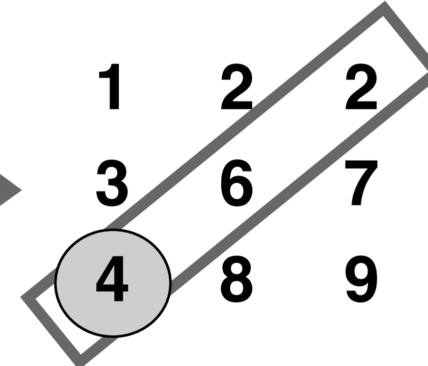
1. Sort columns



2. Sort rows



3. Median of diagonal



NCSG Networking & Computing Systems Group

Digital DNA™ From Motorola

Performance

The following potential AltiVec applications have all demonstrated "order of magnitude" speedups relative to existing scalar implementations:

- Complex FFT
- FIR filters
- Convolutional encoders / Viterbi decoders
- Videoconferencing
- Voice over IP (VoIP)
- Echo cancellation
- Encryption/key generation
- MPEG-2 Encode
- Image processing (median filters, etc.)
- Multi-channel modems

Summary

- **Second generation SIMD architectures will address many shortfalls of current generation**
- **AltiVec has been demonstrated to be applicable to a broad range of desktop and embedded applications**
  - Convergence in marketplace of traditional DSP and general purpose approaches => $/channel, MIPS/watt are the relevant metrics
  - Microprocessor complexity needs to grow at slower rate than demand for performance
  - SIMD is not a standalone answer, but is a scalable architectural component
- **SIMD requires special treatment in software**
  - Different algorithms required, but payoff is significant
  - Conventional languages do not adequately represent SIMD or DSP semantics
  - AltiVec offers standardized programming model for C and C++