ALTIVEC

(a.k.a Velocity Engine)

By:
Ian Ollmann, Ph.D.

Presented by:
Charles “Ted” Betzler

AltiVec

- What is AltiVec?
- How Do We Use It?
- Math
- Hardware Factors
- The Big Picture
- Other Resources

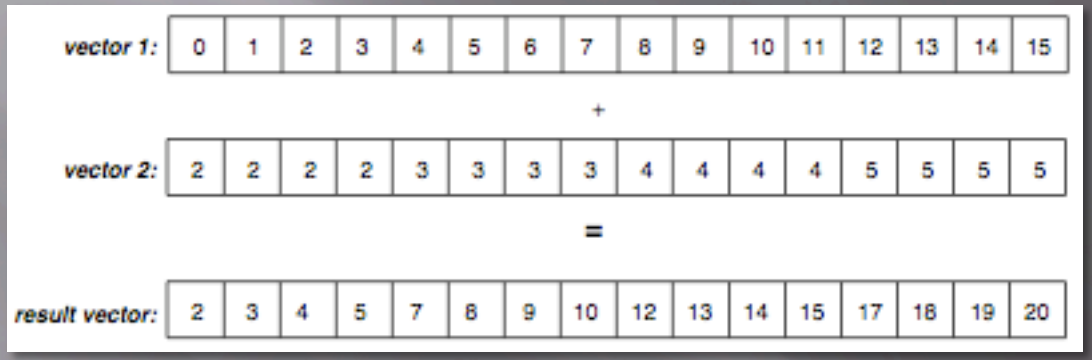
What is AltiVec?

- 128-bit
  Vector Computation Unit
- Included in G4 & G5
  Processors (32 of them)

- Pentium & other processors have similar units
- Separate from the integer unit and FPU
- SIMD – multiple pieces of data simultaneously in parallel

What is AltiVec?

- resultVector = vec_add(vector1, vector2):



- Each 128-bit Vector can hold up to 16 numbers
- Can outpace integer unit by factor of 16
- Can outpace FPU by factor of 4
- With improvements in data layout and cache usage, factor can be even higher
- They're cool

How Do We Use It?

- Compilers:
  - Project Builder
  - GNU Compilers
- Two Programming Interfaces:
  - Assembly
  - C-interface
    - C-interface maps almost 1:1 with Assembly
    - You can build pre-compiled libraries for other languages in C
- Older PowerPC
  - Must write code for older PowerPC – AltiVec Code will not run at all on older PowerPC processors

How Do We Use It?

- Data Types (per register):
  - 128 bits
  - 16 chars
  - 8 shorts
  - 8x16 bit pixels
  - 4 ints
  - 4 single-precision floats
  - Double not supported!
    - 2:1 parallelism not worth it?
    - Motorola made increased FPU handling of doubles to compensate

8 bit types	16 bit types	32 bit types
vector char	vector short	vector int
vector unsigned char	vector unsigned short	vector unsigned int
	vector pixel	vector float

How Do We Use It?

- We use the vector keyword in front of the type to declare a vector:
  - vector char
- In C, a union is used to treat the vector like an array:
  - typedef union {
    vector short vec;
    short elements[8];
    }ShortVector;

    ShortVector shortVector;
    shortVector.vec = (vector short) someVectorShort;
    theThirdElement = shortVector.elements[2];

How Do We Use It?

- Type Conversions are free if bit patterns same:
  - vector float zero = (vector float) vec_splat_u8(0); (vec_splat_u8() returns unsigned char type)
- Normal Type Conversions:

From\To:	char	short	int	16-bit pixel	32-bit pixel	float
char	-	vec_unpackh, vec_unpackl	Convert to short first	X	X	Convert to int first
short	vec_pack, vec_packs, vec_packsu	-	vec_unpackh, vec_unpackl	Static typecast	X	Convert to int first
int	Convert to short first	vec_pack, vec_packs, vec_packsu	-	X	Static Typecast	vec_ctf
16-bit pixel	X	Static typecast	X	-	vec_unpackh, vec_unpackl	Convert to 32-bit pixels first
32-bit pixel	X	X	Static typecast	vec_packpx	-	Convert channels to ints
float	Convert to ints first	Convert to ints first	vec_ctu, vec_cts	Convert to 32-bit pixel first	Convert to int first	-

How Do We Use It?

- Some operations generic, and follow types
- Specific operations override types
- Introducing Constants into Vector
  - Static integers can be expensive
    - If value not in cache, 35-250 cycles!
  - Use splat_X# to generate vectors with a set pattern
    - vec_splat_u8(1) generates a vector full of 0x01
    - vec_splat_s32(1) generates a vector full of 0x00000001
  - vec_lvsl and vec_lvsr move integers to/from integer unit while avoiding stack
    - This can save 5-7 cycles per integer

Math

- **Addition and Subtraction**
  - vec_add() and vec_sub()
  - Saturated: vec_adds() and vec_subs()
    - Clips overflow
- **Multiplication**
  - Many multiplication functions, specific to types
  - Most do A*B+C – for plain multiplication, just pass array of 0 as C

Math

- Division
  - Only possible with floats!
  - Integer division uses fixed point reciprocal multiplication
  - Very involved – please refer to manual for details!
- Square Roots
  - Also only accomplished with floats
  - Very involved – please refer to manual for details!
- Comparator and Permute functions available

Hardware Factors

- Instruction Cache
  - G4 has a 32 kB 8-way set associative instruction cache
  - First iteration of loop slow, successive loops very fast
  - Better to position often called code bocks close in memory
- Pipeline
  - Most instructions take 1-5 cycles
  - G4 Vector pipeline 3-5 stages, depending on model
  - Must keep full of independent data to take advantage

Hardware Factors

- Load/Store
  - vec_ld() and vec_st() – aligned addresses
  - Important to align data (as per earlier presentation)
  - Memory Speed is Always the Problem
    - modern PowerPC machines might be running four, five, six or even seven times as fast as their memory subsystems
- Streaming cache instructions
  - Allows you to manipulate how data is stored in cache
  - Allows you to set up “streams” and manipulate pre-fetch control
  - Set up 64-512 byte overlapping blocks
    - This prevents interruption by other processes

Hardware Factors

- Cache
  - L1 is a eight-way set-associative
    - 32 kB in size
    - Very fast
  - L2 larger, but slower
    - Some models two-way set associative, newer are 8-way
  - L3 even larger and slower
  - L2 (and L3) caches serve as a victim cache – data only comes to be in the L2 or L3 caches after being cast out of the L1 (or L2) cache
  - Data has to be moved to the L1 cache before it can be loaded into register.

Hardware Factors

- Cache penalties:
  - Loading a 32 byte cache line from L2 takes from 10-15 cycles
  - Loading a cache line from RAM to L1 takes about 35-40 cycles on a G4/400
  - If all you do is add those two vectors together (as little as 1 cycle), then during the other 39 cycles your code will do nothing
  - It is important to keep this in mind while optimizing!

The Big Picture

- AltiVec most efficient with 64 bytes or more of data
  - Unaligned cases are too slow
- Less data can be less efficient than scalar processor
- Efficient pipelining is very important
- AltiVec better at high throughput – not low latency
- Where AltiVec really shines is in that 10% of your program that eats up 90% of the CPU
  - Premature optimization is the source of all evil!

Other Resources

- C programming guide: http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPIM.pdf
- Assembly programming guide: http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPEM.pdf
- Power Developer http://www.powerdeveloper.org/
- Freescale http://www.freescale.com/