



POWER Block Course

Assignment 3: SIMD with AltiVec

Agenda

1. SIMD & AltiVec
2. Programming AltiVec in C
3. Example programs
4. Assignment 3
5. Performance measurement
6. Useful resources

Assignment 3

What is this assignment about?

In this assignment you will:

- Get to know AltiVec
- Use AltiVec instructions to speed up computation
- Do performance comparison on Linux using perf

Submission Deadline: 04.11.2016

by any issues: power2016@hpi.de



SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 3

1 SIMD & Altivec

SIMD & Altivec

Sven Köhler,
29.09.2016

Chart 4

1. SIMD & AltiVec

AltiVec is SIMD on POWER

SIMD ::= **S**ingle **I**nstruction **M**ultiple **D**ata

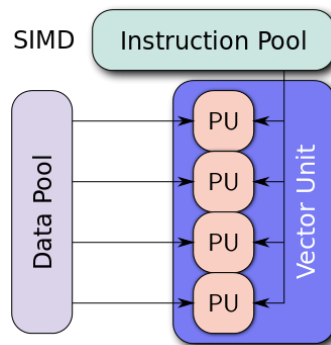
The same instruction is performed simultaneously on multiple data points (data-level parallelism).

Many architectures provide SIMD instruction set extensions.

Intel: MMX, SSE, AVX

ARM: NEON

POWER: AltiVec (VMX), VSX



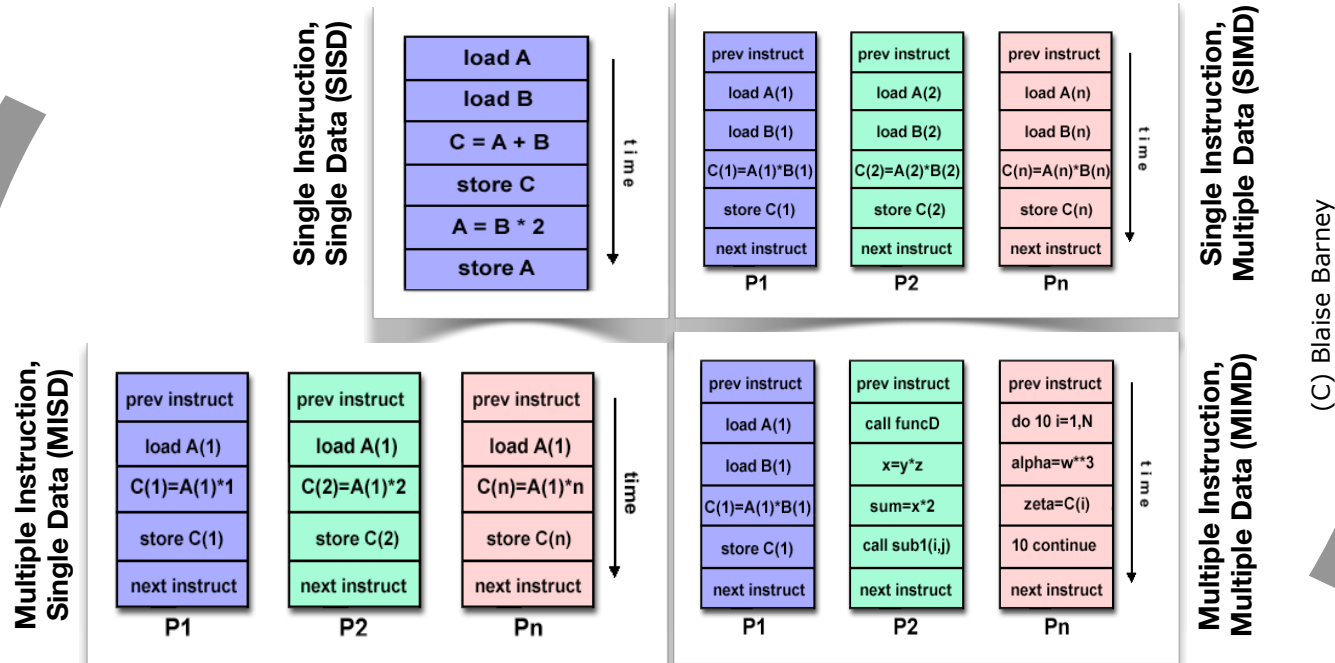
SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 5

1. SIMD & Altivec

Flynn's Taxonomy on Multiprocessors (1966)



SIMD & Altivec

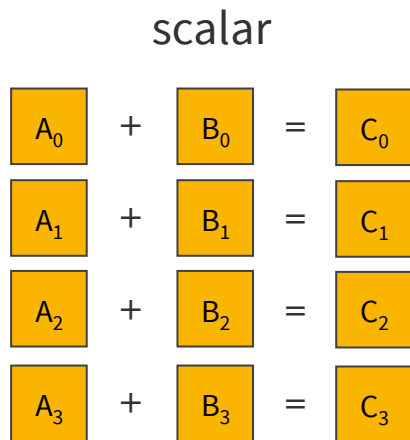
Sven Köhler,
29.09.2016

Chart 6

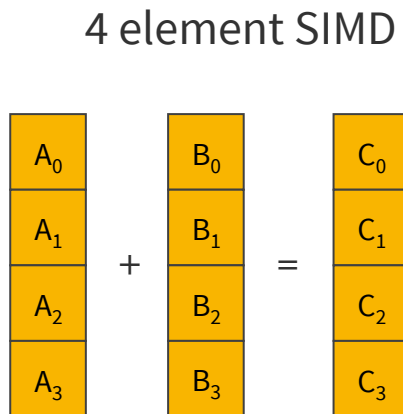
1. SIMD & AltiVec

Scalar vs. SIMD

How many instructions are needed to add four numbers from memory?



4 additions
8 loads
4 stores



1 addition
2 loads
1 store

SIMD & AltiVec

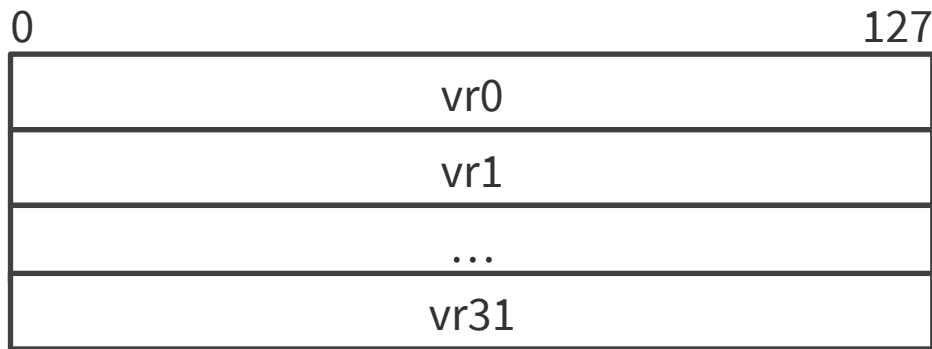
Sven Köhler,
29.09.2016

Chart 7

1. SIMD & AltiVec

Vector Registers on POWER8 (1)

32 vector registers containing 128 bits each.



These are also used by several **coprocessors**:

VSX SHA2 AES ...

Learning AltiVec allows you to make use of them.

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 8

1. SIMD & AltiVec

Vector Registers on POWER8 (2)

32 vector registers containing 128 bits each.

Depending on the instruction they are interpreted as

	16	(un)signed bytes
	8	(un)signed shorts
	4	(un)signed integers of 32bit
	4	single precision floats
	2	(un)signed long integers of 64bit
	2	double precision floats
or	2, 4, 8, 16	logic values

SIMD & AltiVec

Sven Köhler,
29.09.2016

1. SIMD & AltiVec

AltiVec Instruction Reference

For all instructions, registers and usage see

PowerISA 2.07(B), chapter 6 & 7

Version 2.07 B

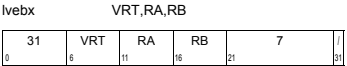
6.7.2 Vector Load Instructions

The aligned byte, halfword, word, or quadword in storage addressed by EA is loaded into register VRT.

Programming Note

The *Load Vector Element* instructions load the specified element into the same location in the target register as the location into which it would be loaded using the *Load Vector* instruction.

Load Vector Element Byte Indexed X-form



```

if RA = 0 then b ← 0
else
    b ← (RA)
EA ← b + (RB)
eb ← EA60:63

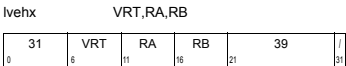
VRT ← undefined
if Big-Endian byte ordering then
    VRT8×eb:8×eb+7 ← MEM(EA,1)
else
    VRT120-(8×eb):127-(8×eb) ← MEM(EA,1)
    
```

Let the effective address (EA) be the sum (RA)0+(RB).

Let eb be bits 60:63 of EA.

If Big-Endian byte ordering is used for the storage access, the contents of the byte in storage at address EA are placed into byte eb of register VRT. The remaining bytes in register VRT are set to undefined values.

Load Vector Element Halfword Indexed X-form



```

if RA = 0 then b ← 0
else
    b ← (RA)
EA ← (b + (RB)) & 0xFFFF_FFFF_FFFE
eb ← EA60:63

VRT ← undefined
if Big-Endian byte ordering then
    VRT8×eb:8×eb+15 ← MEM(EA,2)
else
    VRT112-(8×eb):127-(8×eb) ← MEM(EA,2)
    
```

Let the effective address (EA) be the result of ANDING 0xFFFF_FFFF_FFFE with the sum (RA)0+(RB).

Let eb be bits 60:63 of EA.

If Big-Endian byte ordering is used for the storage access,

SIMD & AltiVec

Sven Köhler,
29.09.2016

2 C-Interface

```
#include <altivec.h>
```

```
gcc -maltivec -mabi=altivec
```

```
gcc -mvsx
```

```
xlc -qaltivec -qarch=auto
```

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart **11**

2. C-Interface

Vector Data Types

The C-Interface introduces new keywords and data types:

vector unsigned char
vector signed char
vector bool char

16x 1 byte

vector unsigned long
vector signed long
vector double

2 x 8 bytes

vector unsigned short
vector signed short
vector bool short
vector pixel

8x 2 bytes

vector unsigned int
vector signed int
vector bool int
vector float

4x 4 bytes

gcc -maltivec

gcc -mvsx

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 12

2. C-Interface

Vector Data Types Initialization, Loading and Storing

```
vector<int> va = {1, 2, 3, 4};
```

```
int data[] = {1, 2, 3, 4, 5, 6, 7, 8};
```

```
vector<int> vb = *((vector<int> *)data);
```

```
int output[4];
```

```
*((vector<int> *)output) = va;
```

```
printf("vb = {%d, %d, %d, %d};\n",  
      vb[0], vb[1], vb[2], vb[3]);
```

Can be very slow!

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 13

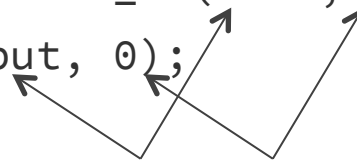
2. C-Interface

Aligned Addresses

Historically memory addresses required be **aligned at 16 byte** boundaries for efficiency reasons. (Although POWER8 has improved unaligned load/store and modern compilers will support you.)

```
int data[] __attribute__((aligned(16))) = {1, 2, 3, 4, 5, 6, 7, 8};  
                                     (compiler specific)  
int *output = aligned_alloc(16, NUM * sizeof(int));
```

```
vector<int> va = vec_ld(data, 0);  
vec_st(va, output, 0);
```



address + index (truncated to 16)

SIMD & AltiVec

Sven Köhler,
29.09.2016

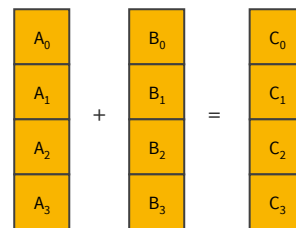
Chart 14

2. C-Interface

Vector Intrinsics

Operations are available through a rich set¹ of overloaded functions (intrinsics):

```
vector<int> va = {4, 3, 2, 1};  
vector<int> vb = {1, 2, 3, 4};  
vector<int> vc = vec_add(va, vb);
```



```
vector<float> vfa = {4, 3, 2, 1};  
vector<float> vfb = {1, 2, 3, 4};  
vector<float> vfc = vec_add(vfa, vfb);
```

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 15

¹https://gcc.gnu.org/onlinedocs/gcc-6.2.0/gcc/PowerPC-AltiVec_002fVSX-Built-in-Functions.html

2. C-Interface

Vector Intrinsics: Lots of overloads

```
vector signed char vec_add (vector bool char, vector signed char);
vector signed char vec_add (vector signed char, vector bool char);
vector signed char vec_add (vector signed char, vector signed char);
vector unsigned char vec_add (vector bool char, vector unsigned char);
vector unsigned char vec_add (vector unsigned char, vector bool char);
vector unsigned char vec_add (vector unsigned char, vector unsigned char);
vector signed short vec_add (vector bool short, vector signed short);
vector signed short vec_add (vector signed short, vector bool short);
vector signed short vec_add (vector signed short, vector signed short);
vector unsigned short vec_add (vector bool short, vector unsigned short);
vector unsigned short vec_add (vector unsigned short, vector bool short);
vector unsigned short vec_add (vector unsigned short, vector unsigned short);
vector signed int vec_add (vector bool int, vector signed int);
vector signed int vec_add (vector signed int, vector bool int);
vector signed int vec_add (vector signed int, vector signed int);
vector unsigned int vec_add (vector bool int, vector unsigned int);
vector unsigned int vec_add (vector unsigned int, vector bool int);
vector unsigned int vec_add (vector unsigned int, vector unsigned int);
vector float vec_add (vector float, vector float);
vector double vec_add (vector double, vector double);
vector long long vec_add (vector long long, vector long long);
vector unsigned long long vec_add (vector unsigned long long, vector unsigned long long);
```

Attention: **No implicit** conversion!

Also not all types for every operation.

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart **16**

2. C-Interface

Get Help: Programming Interface Manual

Highly helpful resource:

- Name of operation
- Pseudocode description
- Text description
- Graphical description
- Type table and according assembly instruction

Generic and Specific AltiVec Operations

vec_add

Vector Add

d = vec_add(**a**,**b**)

- Integer add:

```
n ← number of elements
do i=0 to n-1
  di ← ai + bi
end
```

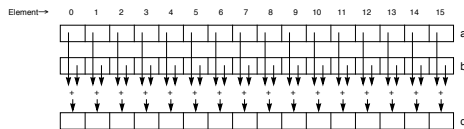
- Floating-point add:

```
do i=0 to 3
  di ← ai +fp bi
end
```

Each element of **a** is added to the corresponding element of **b**. Each sum is placed in the corresponding element of **d**.

For vector float argument types, if VSCR[NJ] = 1, every denormalized operand element is truncated to a 0 of the same sign before the operation is carried out, and each denormalized result element is truncated to a 0 of the same sign.

The valid combinations of argument types and the corresponding result types for **d** = vec_add(**a**,**b**) are shown in Figure 4-12, Figure 4-13, Figure 4-14, and Figure 4-15.



d	a	b	maps to
vector unsigned char	vector unsigned char	vector unsigned char	vaddubm d,a,b
	vector unsigned char	vector bool char	
	vector bool char	vector unsigned char	
	vector signed char	vector signed char	

vec_add

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 17

2. C-Interface

Get Help: IBM Knowledge Center

IBM has a online documentation
of an extended standard,

not fully implemented by GCC.

The screenshot shows the IBM Knowledge Center page for the `vec_add` function. The breadcrumb trail is: XL C for AIX > XL C for AIX 12.1.0 > XL C for AIX, V12.1 > Compiler Reference > Compiler built-in functions > Vector built-in functions > `vec_add`. The page title is `vec_add` (Version 12.1.0). The **Purpose** section states: "Returns a vector containing the sums of each set of corresponding elements of the given vectors. This function emulates the operation on long long vectors." The **Syntax** section shows the code: `d=vec_add(a, b)`. The **Result and argument types** section includes a table describing the types of the returned value and the function arguments.

d	a	b
vector signed char	vector signed char	vector signed char
vector unsigned char	vector unsigned char	vector unsigned char
vector signed short	vector signed short	vector signed short
vector unsigned short	vector unsigned short	vector unsigned short
vector signed int	vector signed int	vector signed int
vector unsigned int	vector unsigned int	vector unsigned int
vector signed long long	vector signed long long	vector signed long long
vector unsigned long long	vector unsigned long long	vector unsigned long long
vector float	vector float	vector float
vector double	vector double	vector double

The **Result value** section states: "The value of each element of the result is the sum of the corresponding elements of a and b. For integer vectors and unsigned vectors, the arithmetic is modular." The **Parent topic** is [Vector built-in functions](#). At the bottom, there is a [Provide feedback](#) link and a **Contact Us** button.

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 18

2. C-Interface

Some Example Instructions Working on Elements

<code>vec_add(a, b)</code>	Add a and b element-wise
<code>vec_sub(a, b)</code>	Subtract a and b element-wise
<code>vec_mul(a, b)</code>	Multiply a and b element-wise (gcc: float only)
<code>vec_madd(a, b, c)</code>	Multiply a and b element-wise and add elements of c
<code>vec_min(a, b)</code>	Select element-wise the minimum of a and b
<code>vec_re(a)</code>	Compute reciprocals of elements
<code>vec_sqrt(a)</code>	Calculate square root of elements
<code>vec_sr(a, b)</code>	Right-shift elements of vector a depending on certain bits in b

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart **19**

2. C-Interface

Conversion of Floating Point Types

<code>vec_ctf(a, n)</code>	Divides the elements of integer vector <code>a</code> by 2^n and converts them into floating-point values.
<code>vec_ctu(a, n)</code>	Multiplies the elements of floating-point vector <code>a</code> by 2^n and converts them into unsigned integers.

Idea behind this: **Fixed-point numbers** of n digits.

For just plain conversion use $n = 0$.

2. C-Interface

Type Casting

When you cast one vector type to another, the contents will be **reinterpreted** like **memory cells**.

```
vector<int> va = {0x11, 0x22, 0x33, 0x44};  
vector<short> vb = (vector<short>)va;
```

Big-endian: {0x00, 0x11, 0x00, 0x22, 0x00, 0x33, 0x00, 0x44}

Little-endian: {0x11, 0x00, 0x22, 0x00, 0x33, 0x00, 0x44, 0x00}

SIMD & AltiVec

Sven Köhler,
29.09.2016

(details later)

Chart 21

2. C-Interface

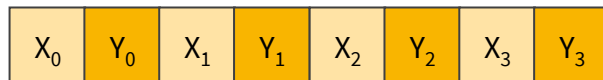
Vector Data Realignment and Permutation (1)

Sometimes memory is not correctly ordered for a certain tasks.

Example: Squared absolute of 2D points ($r = p_x^2 + p_y^2$)

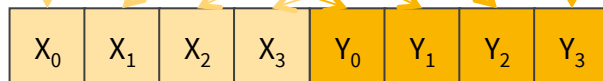
$$\begin{bmatrix} R_0 \\ R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} * \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} + \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} * \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix}$$

in memory:



struct point2d[];

in registers:



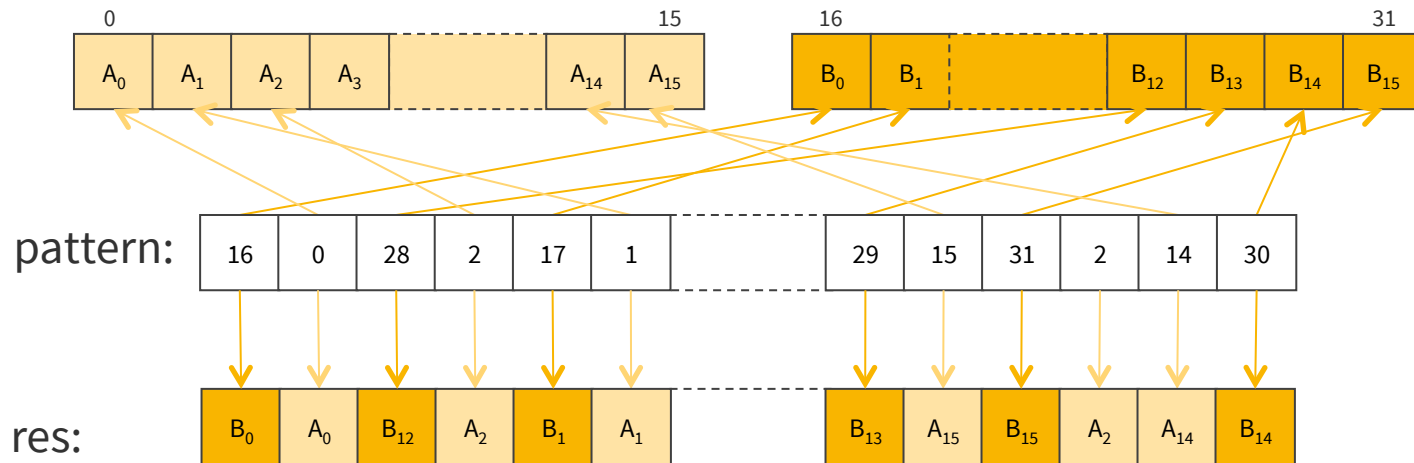
SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 22

Bytewise rearrange two vectors according to provided pattern.

pattern denotes indices in assumed 32 byte array of concatenated a and b.



SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 23

2. C-Interface

Vector Data Realignment and Permutation (3)

The permutation pattern is again endianness dependent!

Every element is interpreted according to the byte-order currently set.

`vector signed int a = {0x1234, 0xaabb, ...}`

BE: a =

00	00	12	34	00	00	AA	BB
----	----	----	----	----	----	----	----

...

LE: a =

34	12	00	00	BB	AA	00	00
----	----	----	----	----	----	----	----

...

SIMD & AltiVec

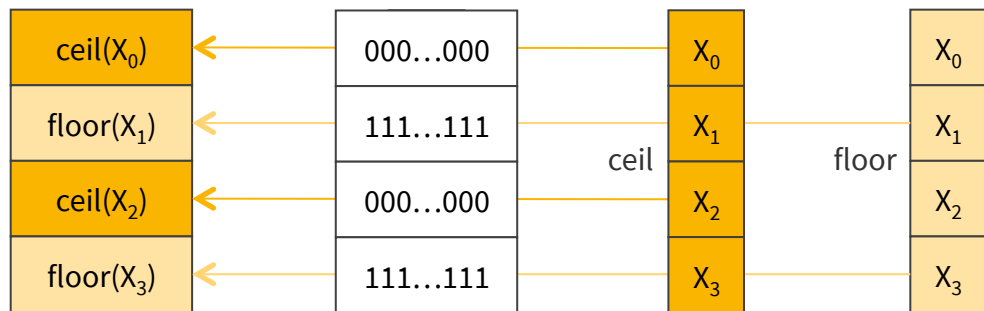
Sven Köhler,
29.09.2016

Chart 24

2. C-Interface

Vector Bit Selection (1)

Sometimes two vectors should be combined, but their bytes not moved.
Example: Every even element of a vector should be rounded up, and every odd one rounded down.



```
vector float a = vec_ceil(X);  
vector float b = vec_floor(X);  
vector unsigned int pattern = {0, 0xffffffff, 0, 0xffffffff};  
vector float res = vec_sel(a, b, pattern);
```

SIMD & AltiVec

Sven Köhler,
29.09.2016

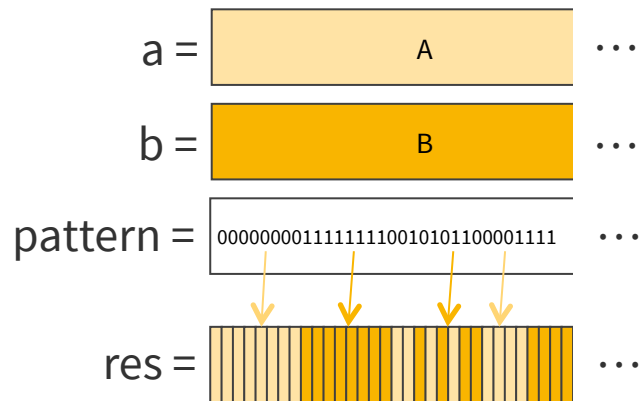
Chart 25

2. C-Interface

Vector Bit Selection (2)

```
res = vec_sel(a, b, pattern)
```

Bit-wise pick contents from a or b, depending if corresponding bit in pattern is 0 or 1.



$\text{res}[\text{bit } i] = \text{a}[\text{bit } i] \text{ if } \text{patter}[\text{bit } i] == 0 \text{ else } \text{b}[\text{bit } i]$

SIMD & AltiVec

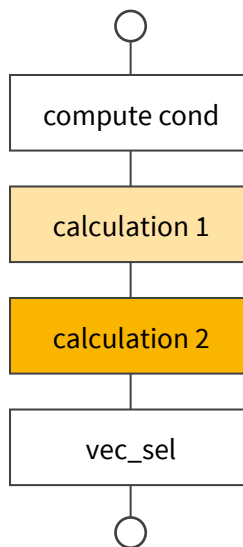
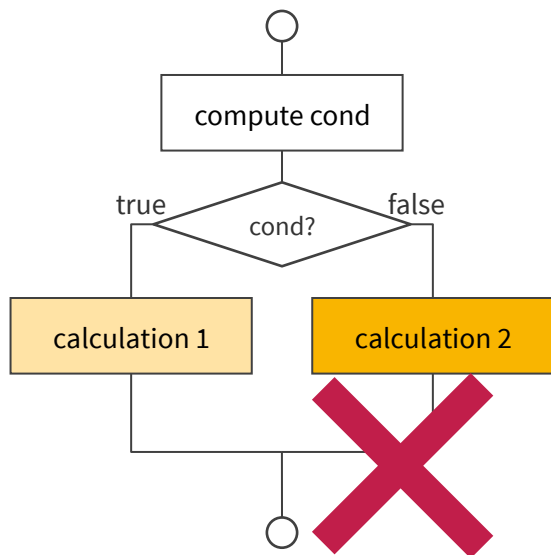
Sven Köhler,
29.09.2016

Chart 26

2. C-Interface

Conditional Programming (1)

There are **no branches** for element computation in AltiVec.
Instead compute both variants and then use **bit-wise select**.



SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 27

2. C-Interface

Conditional Programming (2)

Remember the vector types?

vector unsigned char

~~vector signed char~~

vector bool char

16x false (= 0x0) or true (0xff)

vector unsigned short

~~vector signed short~~

vector bool short

vector pixel

8x false (= 0x0) or true (0xffff)

vector unsigned int

~~vector signed int~~

vector bool int

vector float

4x false (= 0x0) or true (0xffffffff)

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 28

2. C-Interface

Conditional Programming (3)

```
vector bool int res = vec_cmpgt(a, b);
```

11111...11111		2		0
00000...00000		-3		0
11111...11111	=	4	>	0
00000...00000		-2		0

vec_cmpgt	>
vec_cmpge	>= (for gcc on floating point only)
vec_cmpeq	==
vec_cmple	<= (for gcc on floating point only)
vec_cmplt	<

vec_and	(a & b)
vec_or	(a b)
vec_nand	~(a & b)
vec_orc	(a ~b)
...	

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 29

2. C-Interface

Conditional Programming (4)

```
vector signed int calc_abs(vector signed int a)
{
    vector signed int vzero = {0, 0, 0, 0};
    vector signed int neg_a = vec_sub(vzero, a);
    vector bool int vpat = vec_cmpgt(vzero, a);

    return vec_sel(a, neg_a, vpat);
}
```



3 Learning by example

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 31

3. Learning by example

Scale an Array by Factor (Scalar)

```
void scale(float *input, int num, float scale)
{
    int i;
    for (i = 0; i < num; i++) {
        input[i] *= scale;
    }
}
```

3. Learning by example

Scale an Array by Factor (Vector)

```
void scale(float *input, int num, float scale)
{
    int i;
    vector float vscale = {scale, scale, scale, scale};
    for (i = 0; i < num; i += 4) {
        vector float *current = ((vector float *)&input[i]);
        *current = vec_mul(vscale, *current);
    }
}
```

<Do you see a problem?>

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 33

3. Learning by example

Scale an Array by Factor (Vector, Safe)

```
void scale(float *input, int num, float scale)
{
    int i;
    vector float vscale = {scale, scale, scale, scale};
    for (i = 0; i < num - 4; i += 4) {
        vector float *current = ((vector float *)&input[i]);
        *current = vec_mul(vscale, *current);
    }
    for (; i < num; i++) {
        input[i] = scale * input[i];
    }
}
```

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 34

3. Learning by example

Scale an Array by Factor (Vector, Safe, Alternative)

```
void scale(float *input, int num, float scale)
{
    int i;
    vector float vscale = {scale, scale, scale, scale};
    vector float *vinput = (vector float *)input;
    for (i = 0; i < num / 4; i++) {
        vinput[i] = vec_mul(vscale, vinput[i]);
    }
    for (i = (num / 4) * 4; i < num; i++) {
        input[i] = scale * input[i];
    }
}
```

SIMD & AltiVec

Sven Köhler,
29.09.2016

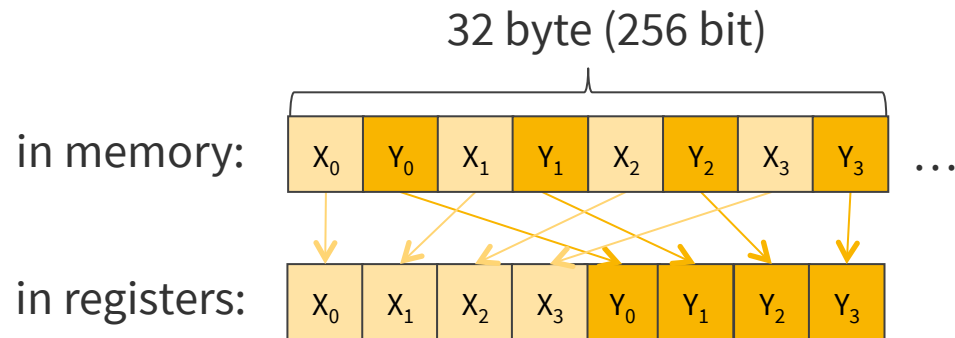
Chart 35

3. Learning by example

Squared Absolute of Points (1)

```
struct point2d {  
    float x, y;  
};
```

```
void squared_2d_abs(struct point2d *input,  
                   float *output, int num);
```



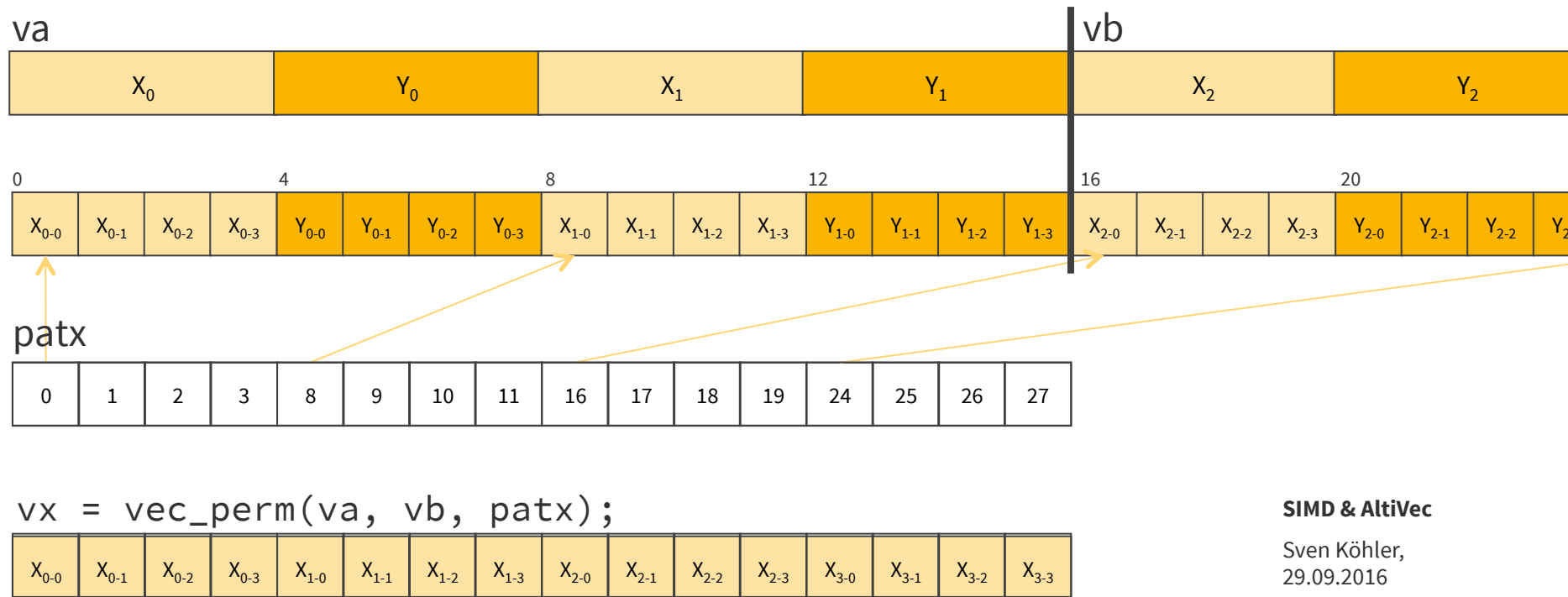
SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 36

3. Learning by example

Squared Absolute of Points (2) – Permute Bytes to Get X



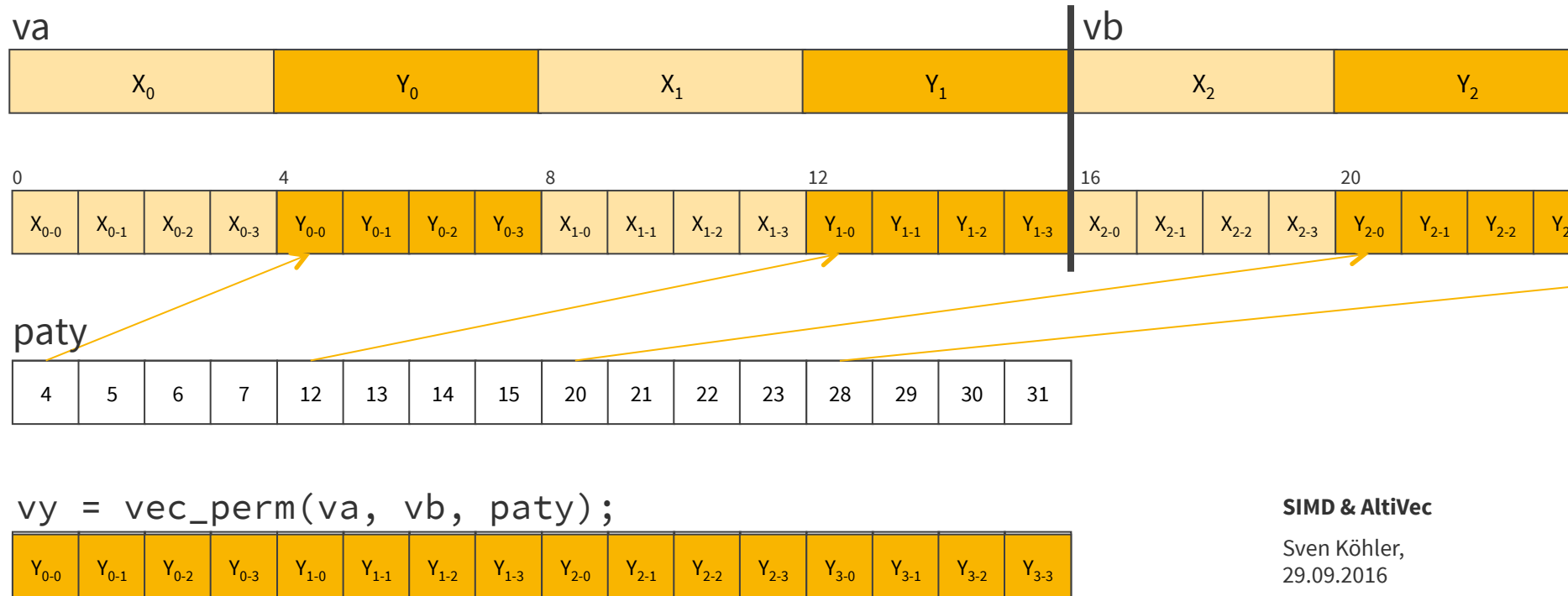
SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 37

3. Learning by example

Squared Absolute of Points (2) – Permute Bytes to Get Y



SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 38

3. Learning by example

Squared Absolute of Points (4) – Patterns in C

```
vector unsigned char patx = {0x00, 0x01, 0x02, 0x03,  
                             0x08, 0x09, 0x0a, 0x0b,  
                             0x10, 0x11, 0x12, 0x13,  
                             0x18, 0x19, 0x1a, 0x1b};
```

```
vector unsigned char paty = {0x04, 0x05, 0x06, 0x07,  
                             0x0c, 0x0d, 0x0e, 0x0f,  
                             0x14, 0x15, 0x16, 0x17,  
                             0x1c, 0x1d, 0x1e, 0x1f};
```

<Any endianness issues here?>

Rule of thumb: No element size or platform change
=> No endianness issues!

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 39

3. Learning by example

Squared Absolute of Points (5) – The Loop

```
int i;  
vector float *vinput  = (vector float *)input;  
vector float *voutput = (vector float *)output;  
  
for (i = 0; i < num / 4; i++) {  
    vector float va = vinput[2 * i];  
    vector float vb = vinput[2 * i + 1];  
  
    vector float vx = vec_perm(va, vb, patx);  
    vector float vy = vec_perm(va, vb, paty);  
    voutput[i] = vec_add(vec_mul(vx, vx), vec_mul(vy, vy));  
}
```

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 40

3. Learning by example

Squared Absolute of Points (6) – What Was Left

```
/* ... */
```

```
for (i = 4 * (num / 4); i < num; i++) {  
    output[i] =    input[i].x * input[i].x  
                  + input[i].y * input[i].y;  
}
```

4

Altivec

Colorfilter

SIMD & Altivec

Sven Köhler,
29.09.2016

Chart 42

4. Assignment

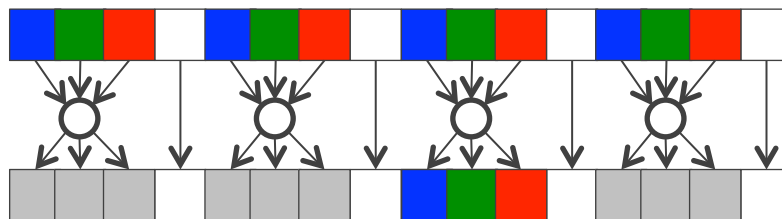
Altivec Colorfilter



You get a 32bit BMP image as one-dim array of BGRA-pixel data.

Convert those pixels to grayscale, where red is not the dominant color.

The alpha channel must not be changed.



SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 43

4. Assignment Altivec Colorfilter



To ease to assignment start by:

- ❑ flipping R, G, and B channel (but preserving the A channel)
- ❑ converting the complete image to grayscale
- ❑ selectively converting the image to grayscale

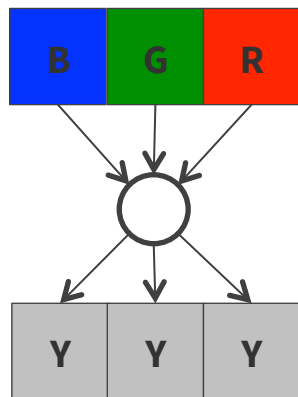
SIMD & Altivec

Sven Köhler,
29.09.2016

Chart **44**

4. Assignment

How Is Grayscale Computed?



or

$$Y = \text{avg}(R, G, B)$$

or

$$Y = (R + 2 * G + B) / 4$$

or

$$Y = 0.2126 * R + 0.7152 * G + 0.0722 * B$$

...

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 45

4. Assignment

Making Things Easier

For this assignment you can assume all input files:

- have four channels (BGRA) of 8 bits each.
- have a pixel number divisible by four

```
int is_apr_for_exercise(bmp_t *bmp)
{
    /* ... */
    return is_argb && is_simdable;
}
```

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 46

5 Performance Measurement

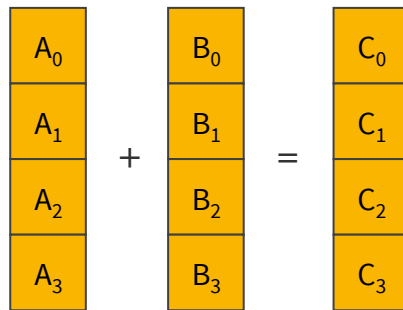
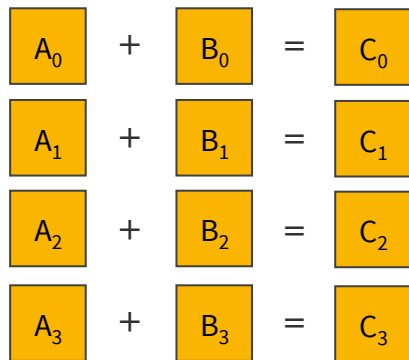
SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 47

5. Performance Measurement

Comparing a Simple Sum



```
vector float *va = (vector float *)a;  
vector float *vb = (vector float *)b;  
vector float *vc = (vector float *)c;
```

```
for (i = 0; i < n; i++) {  
    c[i] = a[i] + b[i];  
}
```

```
for (i = 0; i < n / 4; i++) {  
    vc[i] = vec_add(va[i], vb[i]);  
}
```

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 48

5. Performance Measurement

Performance Sampling on Linux (Scalar)

Using perf 10 times with $n = (1 \ll 30) / \text{sizeof(float)} = 1 \text{ GB}$

sample statistics repeat 10 times events sampled command



```
$ perf stat -r 10 -e cache-misses,instructions,branches ./sum_scalar
```

Performance counter stats for './sum_scalar' (10 runs):

127197342	cache-misses	(+- 0.12%)
1630311051	instructions	(+- 0.01%)
271443407	branches	(+- 0.01%)

 standard deviation

0.512781785 seconds time elapsed (+- 0.18%)

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 49

5. Performance Measurement

Performance Sampling on Linux (Vector)

Using perf 10 times with $n = (1 \ll 30) / \text{sizeof(float)} = 1 \text{ GB}$

sample statistics repeat 10 times events sampled command



```
$ perf stat -r 10 -e cache-misses,instructions,branches ./sum_vector
```

Performance counter stats for './sum_vector' (10 runs):

144145872	cache-misses	(+- 0.03%)
422847600	instructions	(+- 0.00%)
226690657	branches	(+- 0.02%)

 standard deviation

0.429783986 seconds time elapsed (+- 0.12%)

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 50

5. Performance Measurement

Available Events

```
$ perf list
```

List of pre-defined events (to be used in `-e`):

branch-instructions OR branches	[Hardware event]
---------------------------------	------------------

branch-misses	[Hardware event]
---------------	------------------

cache-misses	[Hardware event]
--------------	------------------

cache-references	[Hardware event]
------------------	------------------

cpu-cycles OR cycles	[Hardware event]
----------------------	------------------

instructions	[Hardware event]
--------------	------------------

...

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart **51**

5. Performance Measurement

Finding Raw PMU (Performance Measurement Unit) Events

```
$ perf stat -e ra09c:ppp,ra09e:ppp ./sum_scalar
```

```
0    ra09c:ppp
```

```
0    ra09e:ppp
```

```
$ perf stat -e ra09c:ppp,ra09e:ppp ./sum_vector
```

```
33719464    ra09c:ppp
```

33392615 ra09e:ppp

There are two symmetric Vector-and-Scalar
Units (VSU) pipelines on POWER

EventCode: 0xa09c

EventName: PM_VSU0_4FLOP

BriefDescription: SP vector versions of single flop instructions

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 52

5. Performance Measurement

More on Perf

The ultimate resource on perf: <http://brendangregg.com/linuxperf.html>

Linux Performance

This page links to various Linux performance material I've created, including the tools maps on the right. The first is a hi-res version combining observability, static performance tuning, and perf-tools/bcc (see [discussion](#)). The remainder were designed for use in slide decks and have larger fonts and arrows, and show: [Linux observability tools](#), [Linux benchmarking tools](#), [Linux tuning tools](#), and [Linux sar](#). For even more diagrams, see my slide decks below:

Tools

- [perf events](#): perf one-liners, examples, visualizations.
- [perf-tools](#): perf analysis tools using ftrace (github).
- [bcc](#): perf analysis tools using eBPF (github).
- [ktop](#): one-liners, examples, and scripts.
- [Flame Graphs](#): using [perf](#), [SystemTap](#), and [ktop](#).

Documentation

- [Linux Performance Analysis in 60,000 Milliseconds](#) shows the first ten commands to use in an investigation

hi-res: observability + static + perf-tools/bcc ([svg](#))

Linux Performance Tools

slides: observability

Linux Performance Observability Tools

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart 53

AD

end

SIMD & AltiVec

Sven Köhler,
29.09.2016

Chart **54**