

Core-Level Performance Engineering

Jan Laukemann, Georg Hager

Erlangen National High Performance Computing Center (NHR@FAU)

Outline

- Analytical performance modeling
- Basic x86 processor and core architecture
- Code execution on Out-of-order processor cores
- x86 instruction set intro
- Analysis of simple kernels demo and hands-on
- Introduction to OSACA
- Arm ISA intro
- More complex case studies demo and hands-on
- Summary, caveats, and take-aways



Computational Intensity [Flop/B]



Computational Intensity [Flop/B]

- What is the best performance my code can achieve?
- What are the relevant hardware bottlenecks?
- Apply simplified model of underlying hardware, consisting of
 - In-core execution
 - Data transfer



Operational Intensity (Flops/Byte)



- What is the best performance my code can achieve?
- What are the relevant hardware bottlenecks?
- Apply simplified model of underlying hardware, consisting of
 - In-core execution
 - Data transfer



Operational Intensity (Flops/Byte)







Friedrich-Alexander-Universität Erlangen-Nürnberg

Basic x86 out-of-order core architecture

On the example of a Sapphire Rapids chip







0x00 LOAD from address 0x1f8223de to reg1	1cy on 2 3 11
0x04 LOAD from address 0x1f8244de to reg2	1cy on 2 3 11
0×08 ADD reg1 and reg2 and save in reg3	1cy on 0 1 5 6 10
0x0C STORE reg3 to address 0x2010ff08	1cy on 4 9, 1cy on 7 8



0x00 LOAD from address 0x1f8223de to reg1	1cy on 2 3 11
0x04 LOAD from address 0x1f8244de to reg2	1cy on 2 3 11
0x08 ADD reg1 and reg2 and save in reg3	1cy on 0 1 5 6 10
0x0C STORE reg3 to address 0x2010ff08	1cy on 4 9, 1cy on 7 8



0x00 LOAD from address 0x1f8223de to reg1	1cy on 2 3 11
0x04 LOAD from address 0x1f8244de to reg2	1cy on 2 3 11
0x08 ADD reg1 and reg2 and save in reg3	1cy on 0 1 5 6 10
0x0C STORE reg3 to address 0x2010ff08	1cy on 4 9, 1cy on 7 8



0x00 LOAD from address 0x1f8223de to reg1	1cy on 2 3 11
0x04 LOAD from address 0x1f8244de to reg2	1cy on 2 3 11
0x08 ADD reg1 and reg2 and save in reg3	1cy on 0 1 5 6 10
0x0C STORE reg3 to address 0x2010ff08	1cy on 4 9, 1cy on 7 8



0x00 LOAD from address 0x1f8223de to reg1	1cy on 2 3 11
0x04 LOAD from address 0x1f8244de to reg2	1cy on 2 3 11
0x08 ADD reg1 and reg2 and save in reg3	1cy on 0 1 5 6 10
0x0C STORE reg3 to address 0x2010ff08	1cy on 4 9, 1cy on 7 8







Friedrich-Alexander-Universität Erlangen-Nürnberg

Code execution on out-of-order CPUs

Terminology and explanation



Three metrics to estimate the in-core runtime:

- Reciprocal Throughput (rTP)
- Latency (LT) and Critical Path (CP)
- Loop-carried dependencies (LCD)



Simplified runtime estimation: $t_c = \max(t_{rTP}, t_{LCD})$

- Simple HW model:
 - Six types of functional units (i.e., types of instructions), each functional unit (FU) assigned to one port:



- Reciprocal throughput for each instruction: 1cy
- Latency for each instruction: 1cy

• Port model:



- Loop 1:
 - No dependencies within loop
 - No intra-loop dependencies
 - rTP prediction: 1 cy
 - CP prediction: 1 cy
 - LCD prediction: -



- Loop 1:
 O
 O
 O
 - No dependencies within loop
 - No intra-loop dependencies
 - rTP prediction: 1 cy
 - CP prediction: 1 cy
 - LCD prediction: -





- Loop 1:
 O
 O
 O
 - No dependencies within loop
 - No intra-loop dependencies
 - rTP prediction: 1 cy
 - CP prediction: 1 cy
 - LCD prediction: -





- Loop 1:
 O
 O
 O
 - No dependencies within loop
 - No intra-loop dependencies
 - rTP prediction: 1 cy
 - CP prediction: 1 cy
 - LCD prediction: -





- Loop 1:
 O
 O
 O
 - No dependencies within loop
 - No intra-loop dependencies
 - rTP prediction: 1 cy
 - CP prediction: 1 cy
 - LCD prediction: -





- Loop 1:
 O
 O
 O
 - No dependencies within loop
 - No intra-loop dependencies
 - rTP prediction: 1 cy
 - CP prediction: 1 cy
 - LCD prediction: -





- Loop 1:
 - No dependencies within loop
 - No intra-loop dependencies
 - rTP prediction: 1 cy
 - CP prediction: 1 cy
 - LCD prediction: -















- Loop 2:
 - Dependencies within loop body
 - No loop-carried dependencies
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: -



- - Dependencies within loop body
 - No loop-carried dependencies
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: -







- Loop 2:
 - Dependencies within loop body
 - No loop-carried dependencies
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: -





- Loop 2:
 - Dependencies within loop body
 - No loop-carried dependencies
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: -





- Loop 2:
 - Dependencies within loop body
 - No loop-carried dependencies
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: -



- Loop 2:
 - Dependencies within loop body
 - No loop-carried dependencies
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: -





- Loop 2:
 - Dependencies within loop body
 - No loop-carried dependencies
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: -




- Loop 2:
 - Dependencies within loop body
 - No loop-carried dependencies
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: -



P2

P3

P4

P5

P1

P0

- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy





- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy



- Loop 3:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 3 cy
 - LCD prediction: 3 cy



- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy





- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy





- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy



- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy



- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy



- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy



- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy



- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy





- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy



Loop 4:

- Dependencies within loop body
- Loop-carried dependency
- rTP prediction: 1 cy
- CP prediction: 5 cy
- LCD prediction: 3 cy





ISC25 | Core-Level Performance Engineering Tutorial | NHR@FAU

- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy





ISC25 | Core-Level Performance Engineering Tutorial | NHR@FAU

- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy





- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy





- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy





- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy

ISC25

LCD prediction: 3 cy





- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy





- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy _____



CP

LCD



- Loop 4:
 - Dependencies within loop body
 - Loop-carried dependency
 - rTP prediction: 1 cy
 - CP prediction: 5 cy
 - LCD prediction: 3 cy _____
 - Other limitations:
 - Reorder buffer
 - Loop length
 - Resources (not enough ports, …)
 - Decoder
 - Data









Friedrich-Alexander-Universität Erlangen-Nürnberg

Introduction to the x86 ISA (Instruction Set Architecture)



Basics of the x86-64 ISA

- Operands can be registers (%), memory references ((...)) or immediates (\$)
- There are two assembler syntax forms: Intel (left) and AT&T (right)
- Addressing mode:
 - Intel: [BASE + INDEX * SCALE + OFFSET]
 - AT&T: OFFSET (BASE, INDEX, SCALE)
- C: A[i] equivalent to * (A+i) (a pointer has the type: A+i*8)
- Suffixes: AT&T often uses (optional) suffixes based on the operand size
 - b (byte): 8 bits, w (word): 16 bits, 1 (long): 32 bits, q (quad): 64 bits

Intel syntax	AT&T syntax
<pre>movaps [rdi + rax*8+48], xmm3 add rax, 8 js 1b</pre>	movaps %xmm3, 48(%rdi,%rax,8) addq \$8, %rax js B1.4

Basics of the x86-64 ISA with extensions

16 general purpose registers (64bit): rax, rbx, rcx, rdx, rsi, rdi, rsp, rbp, r8-r15 alias with eight 32-bit register set:

eax, ebx, ecx, edx, esi, edi, esp, ebp



e*x, esi, edi, esp (stack pointer), ebp (base pointer)
Basics of the x86-64 ISA with extensions





VEX/EVEX prefix:	v
Operation:	mul, add, fmadd, mov
Modifier:	nontemporal (nt), unaligned (u), aligned (a), high (h), low (1)
Width:	scalar (s), packed (p)
Data type:	single (s), double (d)
and many more	

VEX/EVEX prefix:	v
Operation:	mul, add, fmadd, mov
Modifier:	nontemporal (nt), unaligned (u), aligned (a), high (h), low (1)
Width:	scalar (s), packed (p)
Data type:	single (s), double (d)
and many more	

Examples: vmulpd

VEX/EVEX prefix:	v
Operation:	mul, add, fmadd, mov
Modifier:	nontemporal (nt), unaligned (u), aligned (a), high (h), low (1)
Width:	scalar (s), packed (p)
Data type: and many more	single (s), double (d)

Examples:

vmulpd>Multiply Packed Double-Precision Floating-Point Values

VEX/EVEX prefix:	v
Operation:	mul, add, fmadd, mov
Modifier:	nontemporal (nt), unaligned (u), aligned (a), high (h), low (1)
Width:	scalar (s), packed (p)
Data type: and many more	single (s), double (d)

Examples:

vmulpd>Multiply Packed Double-Precision Floating-Point Valuesvfmadd213ps

VEX/EVEX prefix:	v
Operation:	mul, add, fmadd, mov
Modifier:	nontemporal (nt), unaligned (u), aligned (a), high (h), low (1)
Width:	scalar (s), packed (p)
Data type: and many more	single (s), double (d)

Examples:

vmulpd	\rightarrow	Multiply Packed Double-Precision Floating-Point Values
vfmadd213ps	\rightarrow	Fused Multiply-Add of Packed Single-Precision Floating-Point Values

 \rightarrow

 \rightarrow

SIMD instructions are distinguished by:

VEX/EVEX prefix:	v
Operation:	mul, add, fmadd, mov
Modifier:	nontemporal (nt), unaligned (u), aligned (a), high (h), low (1)
Width:	scalar (s), packed (p)
Data type:	single (s), double (d)
and many more	

Examples:

vmulpd	
vfmadd213ps	
addsd	

Multiply Packed Double-Precision Floating-Point Values Fused Multiply-Add of Packed Single-Precision Floating-Point Values

VEX/EVEX prefix:	v
Operation:	mul, add, fmadd, mov
Modifier:	nontemporal (nt), unaligned (u), aligned (a), high (h), low (1)
Width:	scalar (s), packed (p)
Data type:	single (s), double (d)
and many more	

Examples:

vmulpd	\rightarrow	Multiply Packed Double-Precision Floating-Point Values
vfmadd213ps	\rightarrow	Fused Multiply-Add of Packed Single-Precision Floating-Point Values
addsd	\rightarrow	Add Scalar Double-Precision Floating-Point Values

VEX/EVEX prefix:	v
Operation:	mul, add, fmadd, mov
Modifier:	nontemporal (nt), unaligned (u), aligned (a), high (h), low (1)
Width:	scalar (s), packed (p)
Data type:	single (s), double (d)
and many more	

Examples:

vmulpd	
vfmadd213ps	
addsd	
vmovntdq	

→ Multiply Packed Double-Precision Floating-Point Values
 → Fused Multiply-Add of Packed Single-Precision Floating-Point Values
 → Add Scalar Double-Precision Floating-Point Values

VEX/EVEX prefix:	v
Operation:	mul, add, fmadd, mov
Modifier:	nontemporal (nt), unaligned (u), aligned (a), high (h), low (1)
Width:	scalar (s), packed (p)
Data type:	single (s), double (d)
and many more	

Examples:

vmulpd	\rightarrow	Multiply Packed Double-Precision Floating-Point Values
vfmadd213ps	\rightarrow	Fused Multiply-Add of Packed Single-Precision Floating-Point Values
addsd	\rightarrow	Add Scalar Double-Precision Floating-Point Values
vmovntdq	\rightarrow	Store Packed Integers Using Non-Temporal Hint

```
double sum = 0.0;
```

```
for (int i=0; i<size; i++) {
    sum += data[i];
}</pre>
```

To get object code use **objdump** -d on object file or executable or compile with -S

Assembly code w/ -O1 (AT&T syntax, Intel compiler):

.label:

addsd	0(%rdi, %rax,	<mark>8)</mark> ,%xmm0
inc	%rax	
cmp	% rsi, % rax	
jl	.label	

```
double sum = 0.0;
```

```
for (int i=0; i<size; i++) {
    sum += data[i];
}</pre>
```

To get object code use **objdump** -d on object file or executable or compile with -S

Assembly code w/ -01 (AT&T syntax, Intel compiler):

.label:



Assembly code w/ -O3 -xCORE-AVX512 -qopt-zmm-usage=high:

..B3.28:

vaddpd	(%r13,%rcx,8), %zmm5, %zmm5
vaddpd	64(%r13,%rcx,8), %zmm4, %zmm4
vaddpd	128 (%r13,%rcx,8), %zmm3, %zmm3
vaddpd	192 (%r13,%rcx,8), %zmm2, %zmm2
addq	\$32, %rcx
cmpq	%rbx, %rcx
jb	B3.28
B3.29:	
vaddpd	<pre>%zmm4, %zmm5, %zmm4</pre>
vaddpd	<pre>%zmm2, %zmm3, %zmm2</pre>
vaddpd	<pre>%zmm2, %zmm4, %zmm5</pre>
# [SNIP]	
B3.34:	
vshuff32x4	1 \$238 , %zmm5, %zmm5, %zmm2
vaddpd	<pre>%zmm5, %zmm2, %zmm3</pre>
vpermpd	\$78, %zmm3, %zmm4
vaddpd	%zmm4, %zmm3, %zmm5
vpermpd	\$177, %zmm5, %zmm6
vaddpd	<pre>%zmm6, %zmm5, %zmm7</pre>
vaddsd	<pre>%xmm1, %xmm7, %xmm1</pre>

Assembly code w/ -O3 -xCORE-AVX512 -qopt-zmm-usage=high:

..B3.28:

vaddpd	(%r13,%rcx,8), %zmm5, %zmm5
vaddpd	64(%r13,%rcx,8), %zmm4, %zmm4
vaddpd	128 (%r13,%rcx,8), %zmm3, %zmm3
vaddpd	192 (%r13,%rcx,8), %zmm2, %zmm2
addq	\$32, %rcx
cmpq	<pre>%rbx, %rcx</pre>
jb	B3.28
B3.29:	
vaddpd	%zmm4, %zmm5, %zmm4
vaddpd	%zmm2, %zmm3, %zmm2
vaddpd	<pre>%zmm2, %zmm4, %zmm5</pre>
# [SNIP]	
B3.34:	
vshuff32x4	4 \$238, %zmm5, %zmm5, %zmm2
vaddpd	<pre>%zmm5, %zmm2, %zmm3</pre>
vpermpd	\$78, %zmm3, %zmm4
vaddpd	<pre>%zmm4, %zmm3, %zmm5</pre>
vpermpd	\$177, %zmm5, %zmm6
vaddpd	<pre>%zmm6, %zmm5, %zmm7</pre>
vaddsd	<pre>%xmm1, %xmm7, %xmm1</pre>

Bulk loop code (8x4-way unrolled)

Assembly code w/ -O3 -xCORE-AVX512 -qopt-zmm-usage=high:

..B3.28:

vaddpd	(%r13,%rcx,8), %zmm5, %zmm5
vaddpd	64(%r13,%rcx,8), %zmm4, %zmm4
vaddpd	128(%r13,%rcx,8), %zmm3, %zmm3
vaddpd	192(%r13,%rcx,8), %zmm2, %zmm2
addq	\$32, %rcx
cmpq	%rbx, %rcx
jb	B3.28
B3.29:	
vaddpd	%zmm4, %zmm5, %zmm4
vaddpd	%zmm2, %zmm3, %zmm2
vaddpd	%zmm2, %zmm4, %zmm5
# [SNIP]	Demoinder emitted
B3.34:	Remainder ommed
vshuff32x4	4 \$238 , %zmm5, %zmm5, %zmm2
vaddpd	%zmm5, %zmm2, %zmm3
vpermpd	\$78, %zmm3, %zmm4
vaddpd	%zmm4, %zmm3, %zmm5
vpermpd	\$177, %zmm5, %zmm6
vaddpd	%zmm6, %zmm5, %zmm7
vaddsd	<pre>%xmm1, %xmm7, %xmm1</pre>

Bulk loop code (8x4-way unrolled)

Assembly code w/ -O3 -xCORE-AVX512 -qopt-zmm-usage=high:

...B3.28: vaddpd (%r13,%rcx,8), %zmm5, %zmm5 64 (%r13,%rcx,8), %zmm4, %zmm4 vaddpd vaddpd 128(%r13,%rcx,8), %zmm3, %zmm3 vaddpd **192**(%r13,%rcx,8), %zmm2, %zmm2 addq \$32, %rcx cmpq %rbx, %rcx ..B3.28 jb ...B3.29: vaddpd %zmm4, %zmm5, %zmm4 vaddpd %zmm2, %zmm3, %zmm2 %zmm2, %zmm4, %zmm5 vaddpd # [... SNIP ...] Remainder omitted ...B3.34: vshuff32x4 \$238, %zmm5, %zmm5, %zmm2 %zmm5, %zmm2, %zmm3 vaddpd vpermpd **\$78**, **%zmm3**, **%zmm4** vaddpd %zmm4, %zmm3, %zmm5 \$177, %zmm5, %zmm6 vpermpd %zmm6, %zmm5, %zmm7 vaddpd vaddsd %xmm1, %xmm7, %xmm1

Bulk loop code (8x4-way unrolled)

Sum up 32 partial sums into xmm1

Example for masked execution

Masking is very helpful in cases such as, e.g., remainder loop handling or conditionals

Available on x86 starting with AVX-512

Example: vaddps %zmm0, %zmm1, %zmm2{%k1}







Friedrich-Alexander-Universität Erlangen-Nürnberg

STREAM Triad

A pen & paper in-core analysis



		P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	
STREAM	M TRIAD		ALU LEA	LD	LD AGU	ST	ALU LEA	ALU	ST AGU	ST	ST AGU	ALU LEA	256b LD	
a[i] =	= b[i] + s * c[i]	T Execution C BR AVX ALU	INT MUL INT DIV 256b ALU				INT MUL AVX ALU FMA	BR					AGU	
.B2.42: vmovups vfmadd213pd vmovupd addq cmpq jb	<pre>(%r14,%rdx,8), %zmm1 (%r15,%rdx,8), %zmm2, %zmm1, (%r12,%rdx,8) \$8, %rdx %rsi, %rdxB2.42</pre>	% zmm1	256b FMA											

		P0	P1	P2 P3	P4	P5 P6	P7 P8	P9	P10	P11
STREAM			ALU LEA	LD LD	ST /	ALU ALU	ST ST AGU	ST AGU	ALU LEA	256b LD
<pre>a[i] =B2.42: vmovups</pre>	b[i] + s * c[i] (%r14,%rdx,8), %zmm1	BR SHFT AVX ALU FMA DIV	INT MUL INT DIV 256b ALU 256b FMA		F	INT MUL AVX ALU FMA				AGU
vfmadd213pd vmovupd addq cmpq jb	(%r15,%rdx,8), %zmm2, %zmm %zmm1, (%r12,%rdx,8) \$8, %rdx %rsi, %rdx B2.42	n1		LD						

		P0	P1	P2 P3	P4	P5 P6	P7 P8	P9	P10	P11
STREAM			ALU LEA	LD LD	ST /	ALU ALU	ST ST AGU	ST AGU	ALU LEA	256b LD
<pre>a[i] =B2.42: vmovups</pre>	b[i] + s * c[i] (%r14,%rdx,8), %zmm1	BR SHFT AVX ALU FMA DIV	INT MUL INT DIV 256b ALU 256b FMA		F	INT MUL AVX ALU FMA				AGU
vfmadd213pd vmovupd addq cmpq jb	(%r15,%rdx,8), %zmm2, %zmm %zmm1, (%r12,%rdx,8) \$8, %rdx %rsi, %rdx B2.42	n1		LD						

	P0	P1	P2 P3	P4 P	5 P6	P7	P8	P9	P10	P11	
STREAM TRIAD a[i] = b[i] + s * c[i] B2.42: vmovups (%r14,%rdx,8), %zmm1	ALU LEA BR SHFT AVX ALU FMA DIV	ALU LEA INT MUL INT DIV 256b ALU 256b FMA	LD LD AGU AGU	ST AL LE IN ML AV AL FM	U ALU A LEA T BR IL SHFT U A	ST AGU	ST	ST AGU	ALU LEA	256b LD AGU	
vfmadd213pd (%r15,%rdx,8), %zmm2, %zmm vmovupd %zmm1, (%r12,%rdx,8) addq \$8, %rdx cmpq %rsi, %rdx jbB2.42	1 FMA										

• •

	P0 P'	1 P2 P3	P4 P5	P6 P7	P8 P9 P10	P11
STREAM TRIAD		U LD LD A AGU AGU	ST ALU	ALU ST AGU	ST ST ALU AGU LEA	256b LD
<pre>a[i] = b[i] + s * c[i]B2.42: vmovups (%r14,%rdx,8), %zmm1</pre>	T Continue of the security of	T JL T V δb U δb	INT MUL AVX ALU FMA	BR SHFT		AGU
vfmadd213pd (%r15,%rdx,8), %zmm2, %zm vmovupd %zmm1, (%r12,%rdx,8) addq \$8, %rdx cmpq %rsi, %rdx	FMA	LD				
JDB242			STR	STR		

	P0	P1	P2 P3	P4 P5	P6	P7 P8	P9 F	P10 P11	
STREAM TRIAD		ALU LEA	LD LD AGU	ST ALU		ST ST AGU	ST AGU	LU 256b LD	
<pre>a[i] = b[i] + s * c[i] B2.42: vmovups (%r14,%rdx,8), %zmm1</pre>	BR SHFT AVX ALU FMA DIV	INT MUL INT DIV 256b ALU 256b FMA		INT MUL AVX ALU FMA	BR SHFT			AGU	
vfmadd213pd (%r15,%rdx,8), %zmm2, %zm vmovupd %zmm1, (%r12,%rdx,8) addq \$8, %rdx cmpq %rsi, %rdx jbB2.42	nm1 FMA		LD	STR	S	TR			
				ADD					

	P0	P1	P2 P3	P4 P5	P6	P7 P8	P9 F	P10 P11	
STREAM TRIAD		ALU LEA	LD LD AGU	ST ALU		ST ST AGU	ST AGU	LU 256b LD	
<pre>a[i] = b[i] + s * c[i] B2.42: vmovups (%r14,%rdx,8), %zmm1</pre>	BR SHFT AVX ALU FMA DIV	INT MUL INT DIV 256b ALU 256b FMA		INT MUL AVX ALU FMA	BR SHFT			AGU	
vfmadd213pd (%r15,%rdx,8), %zmm2, %zm vmovupd %zmm1, (%r12,%rdx,8) addq \$8, %rdx cmpq %rsi, %rdx jbB2.42	nm1 FMA		LD	STR	S	TR			
				ADC					

	P0 P1	P2 P3 P4	P5 P6 P7	P8 P9 P10 P11
<pre>STREAM TRIAD a[i] = b[i] + s * c[i]B2.42:</pre>	ALU ALU LEA LEA BR INT MUL SHFT DIV AVX ALU 256b ALU 256b	LD LD ST AGU AGU	ALUALUST AGULEALEAINTBRMULSHFTAVXSHFTALUFMA	ST ST ALU 256b LD LEA AGU
vmovups (%r14,%rdx,8), %zmm1 vfmadd213pd (%r15,%rdx,8), %zmm2, % vmovupd %zmm1, (%r12,%rdx,8) addq \$8, %rdx cmpq %rsi, %rdx jb B2.42	Zmm1 FMA	LD LD STR	ADD	

STREAM TRIAD a[i] = b[i] + s * c[i] B2.42: MMOVINS (%r14.%rdx.8), %rm1	PO ALU LEA BR SHFT AVX ALU FMA	P1 P2 ALU LD LEA AGU INT MUL INT DIV 256b ALU 256b	P3 P4 LD ST AGU	P5 P6 ALU ALU LEA LEA INT BR MUL BR SHFT AVX ALU FMA	P7 P8	P9 P10 ST ALU AGU LEA	P11 256b LD AGU
vfmadd213pd (%r15,%rdx,8), %zmm2, vmovupd %zmm1, (%r12,%rdx,8) addq \$8,%rdx cmpq %rsi,%rdx jbB2.42	%zmm1 It x+1 FMA	LD	LD STR	ADD CMP&JMF	STR		

STREAM TRIAD a[i] = b[i] + s * c[i] B2.42: MMOVINS (%r14.%rdx.8), %rm1	PO ALU LEA BR SHFT AVX ALU FMA	P1 P2 ALU LD LEA AGU INT MUL INT DIV 256b ALU 256b	P3 P4 LD ST AGU	P5 P6 ALU ALU LEA LEA INT BR MUL BR SHFT AVX ALU FMA	P7 P8	P9 P10 ST ALU AGU LEA	P11 256b LD AGU
vfmadd213pd (%r15,%rdx,8), %zmm2, vmovupd %zmm1, (%r12,%rdx,8) addq \$8,%rdx cmpq %rsi,%rdx jbB2.42	%zmm1 It x+1 FMA	LD	LD STR	ADD CMP&JMF	STR		

STREAM TRIAD a[i] = b[i] + s * c[i] B2.42: MOVINS (%r14 %rdx 8) %rm1	PO ALU LEA BR SHFT AVX ALU FMA	P1 P2 ALU LD LEA AGU INT MUL INT DIV 256b ALU 256b	P3 P4	P5P6ALUALULEALEAINT MULBRAVX ALUSHFTFMAFMA	P7 P8	P9 P10 ST ALU AGU LEA	P11 256b LD AGU
vfmadd213pd (%r15,%rdx,8), %zmm2, vmovupd %zmm1, (%r12,%rdx,8) addq \$8,%rdx cmpq %rsi,%rdx jbB2.42	%zmm1 It x+1 FMA It x+2 FMA	LD	LD STR	ADD CMP&JMF	² STR STR		

STREAM a[i] =	TRIAD b[i] + s * c[i]	Execution Units	P0 ALU LEA BR SHFT AVX ALU	P1 ALU LEA INT MUL INT DIV 256b ALU	P2 LD AGU	P3 LD AGU	P4 ST	P5 ALU LEA INT MUL AVX ALU FMA	P6 ALU LEA BR SHFT	P7 ST AGU	P8 ST	P9 ST AGU	P10 ALU LEA	P11 256b LD AGU	
vmovups vfmadd213pd vmovupd addq cmpq jb	<pre>(%r14,%rdx,8), %zmm1 (%r15,%rdx,8), %zmm2, %zmm1, (%r12,%rdx,8) \$8, %rdx %rsi, %rdxB2.42</pre>	%zmm1 It x+1 It x+2 It x+3	DIV FMA FMA FMA	FMA	LD LD LD	LD LD LD	STR STR STR	ADD ADD ADD	CMP&JMP CMP&JMP CMP&JMP	STR STR STR					

STREAM a[i] =	TRIAD b[i] + s * c[i]	Execution Units	PO P1 LU ALU EA LEA BR INT MUL HFT INT VX LU 256b ALU	P2 LD AGU	P3 LD AGU	P4 ST	P5 ALU LEA INT MUL AVX ALU FMA	P6 ALU LEA BR SHFT	P7 ST AGU	P8 ST	P9 ST AGU	P10 ALU LEA	P11 256b LD AGU
vmovups vfmadd213pd vmovupd addq cmpq jb	<pre>(%r14,%rdx,8), %zmm1 (%r15,%rdx,8), %zmm2, %zmm1, (%r12,%rdx,8) \$8, %rdx %rsi, %rdxB2.42</pre>	% zmm1 It x+1 FN It x+2 FN It x+3 FN	1A 1A 1A	LD LD LD	LD LD LD	STR STR STR	ADD ADD ADD	CMP&JMP CMP&JMP CMP&JMP	STR STR STR				
	1 cy / 8 it	>											





Friedrich-Alexander-Universität Erlangen-Nürnberg

Hands-On #1: Dot product

→ <u>https://go-nhr.de/CLPE-ex1</u>



Hands-On: Benchmarking Dot Product



\rightarrow Moodle, hands-on #1

Dot Product on SPR







Friedrich-Alexander-Universität Erlangen-Nürnberg

Hands-On #2: Dot product (with Compiler Explorer)

→ <u>https://go-nhr.de/CLPE-ex2</u>



Dot Product on SPR – CE view

\rightarrow Moodle, hands-on #2

	Ē	CPLOKER Add • More • Templates							Share ▼ Other ▼	Set ASM compiler
	C++ source	#1 0			x86-64 icpc	2021.6.0 (Editor #1)	1 X			compiler and flags
Add new compiler	A - 🖬	Save/Loa + Add new 🔻 ۷ Vim 🔑 CppInsights 📌 Quick-bench	C++	-	x86-64 i	срс 2021.6.0	🔹 🖸 🥝 -Ofast -qope	enmp-simd -xHost -qopt-zmm-usage	e=high -fargument-noalias -funroll-l	
Add new complier	116	// ment and after anoth		The second secon	A• 🌣	Output 👻 🔻 Filte	er 🝷 📕 Libraries 🛛 🔑 Overrides	+ Add new 🔨 🖍 Add tool 🕇		
	117	<pre>for(int k=0: k<niter: ++k)="" pre="" {<=""></niter:></pre>		BARRATE AND	283	cmpl	%esi. %eax	#122.7		Add new analysis
	119	if (s == 1.20000) printf("%lu\n", &s);		A CONTRACTOR OF A CONTRACTOR O	284	jl	B1.48 # Prob 50%	#122.7		
	120	//		IR Silves	285	B1.45:	# PredsB1	1.44		tool
	121	// benchmark loop		Particular and the second second	286	movslq	%r10d, %r10 %oox %rsi	#122.7		
Right click and	122	s = s + a[i] * b[i];		EXAMPLATION OF A PARTY AND A PARTY AND A PARTY AND A PARTY A P	287		# Preds	#122.7 1.46B1.45	1 Same	 Add new executor
"Reveal linked	124	}	Compile Ctrl+Enter	MIN.	289	vmovups	(%rcx,%r10,8), %zmm6	#123.24	Kanada and Andreas	
code" can beln vou	125	// end of benchmark loop	Reveal linked code Ctrl+F10	C INCOMENTATION	290	vmovups	64(%rcx,%r10,8), %zmm7	#123.24		
code can help you	126	//	Change All Occurrences Ctrl+F2	i i decentrative. Successo per continue : per continue :	291	vmovups	128(%rcx,%r10,8), %zmm8	#123.24 #123.24		
find your region of	128	}	Format Document Shift+Alt+F	The second se	293	vfmadd23	1pd (%r8,%r10,8), %zmm6, %zn	nm2 #123.24		
interest	129	<pre>ioctl(perf_fd, PERF_EVENT_IOC_DISABLE, 0);</pre>	Cut	English and an and	294	vfmadd23	1pd 64(%r8,%r10,8), %zmm7, %	%zmm3 #123.24		
	130	<pre>wct_end = getTimeStamp();</pre>	Copy	BED 2.5 WEW 22 Marsh	295	vfmadd23	1pd 128(%r8,%r10,8), %zmm8,	%zmm5 #123.24		
	131	err = read(pert_td, &ncycles, sizeot(long long)); NTTER = NTTER*2:	Paste		296	vtmadd23 addo	1pd 192(%r8,%r10,8), %zmm9, \$32 %r10	%zmm4 #123.24 #122.7		
	133	<pre>} while (wct_end-wct_start<1.0); // at least 1000 ms</pre>	Search on Cppreference Ctrl+F8		298	cmpq	%rsi, %r10	#122.7		
	134		Command Palette F1		299	jb	B1.46 # Prob 82%	#122.7	Part I	
	135	NITER = NITER/2;			300	B1.47:	# PredsB1	1.46	R Robert B	Click to see your
	130	err = read(perf fd, &ncycles, sizeof(long long));			301	vaddpd vaddpd	%ZMM3, %ZMM2, %ZMM2 %ZMM4 %ZMM5 %ZMM3	#123.13 #123.13		compiler log
	138	if (err < 0) {			303	vaddpd	%zmm3, %zmm2, %zmm2	#123.13		(warnings and errors)
	139	return 1;			304	B1.48:	# PredsB1	1.44B1.47 <u>B1.69</u>	and the second sec	(Warninge and errore)
	140	}			305	lea	1(%rax), %esi	#122.7	See a	
	141	<pre>print("Size: %.21 kB, %d elements(h', Size, %); printf("Cycles per high-level iteration: %f\n", (double)(</pre>	<pre>ncycles-ncycles tmp)/NITER/N);</pre>	_	C 🗒 Ou	itput (0/0)	pc 2021.6.0 i - 6234ms (164950B) ~6076	lines filtered 🔳		
	143	printf("Total walltime: %f, NITER: %d\n",wct_end-wct_star	t,NITER);		OSACA x86	6-64 icpc 2021.6.0 (Edi	tor #1. Compiler #1) 🖉 🗙			
Satevecutor	144	Fore (a) -				Wrap lines > Arg	uments			
Seiexeculei	145	free(a);								
compiler and flags	147	return 0;			lines 2	88-299arch spi				Set OSACA runtime
(separately from	European and	0 0 4 inter 2024 C 0 (0) + Enter #41 A M								parameters
ASM compiler)	Executor x				P - Thro	ughput of LOAD c	peration can be hidden behir	nd a past or future STORE instruct	tion	•
ASIN complier)	A* U	Wrap lines 🗧 Libraries 🎾 Overrides 🦓 Compilation 🗲 Argumer	ts 🚽 Stdin 🔍 Runtime tools 🚺 Compiler outpu	ut	* - Inst	ruction micro-op	s not bound to a port			
•	x86-64 i	cpc 2021.6.0 🔻 🖸 🥝 -Ofast -qopenmp-simd -xHo	st -qopt-zmm-usage=high -fargument-noalias	-funroll-loop:	X - No t	hroughput/latenc	y information for this instr	ruction in data file		
	40									
	40				Combined	Analysis Report				
	Program r	eturned: 0								
	Program s	tdout					Port	t pressure in cycles		
	Cycles per	high-level iteration: 0.161946			0	- 0DV 1	- 1DV 2 3 4	5 6 7 8 9	9 10 11 CP LCD	
	Total wall	time: 1.983860, NITER: 8388608								
Set runtime					288					
					289		0.50 0.50			
parameters					291		0.32 0.33		0.850	
					292		0.00 0.00		1.500	
					293 0.	50	0.50 0.50	0.50	9.0	
					294 0.	50	0.50 0.50	0.50		
					295 0.	50	0.50 0.50	0.50		
					296 0.	50	0.50 0.50	0.50	4.0	
					297 0.	00 0.50				
	O C :	(86-64 icpc 2021.6.0 į́ - 5935ms ⋿			4	aa 18.58				




Friedrich-Alexander-Universität Erlangen-Nürnberg

The Open-Source Architecture Code Analyzer (OSACA)

An introduction



OSACA

- Open Source Architecture Code Analyzer
- Static in-core code analysis

Assumptions

- Steady-state execution (no warm-up/cool-down)
- All data in L1
- Perfect out-of-order scheduling
- (currently) no front-end, i.e., no limit in instruction fetching, decoding, etc...
- Architecture specific model for each µArch

Python module

\$ pip install osaca

OSACA – Usage

osaca [-h] [-V] [--arch ARCH] [--fixed] [--lines LINES] [--ignore-unknown] [--lcd-timeout SECONDS] [--db-check] [--import MICROBENCH] [--insert-marker] [--export-graph GRAPHNAME] [--consider-flag-deps] [--out OUT] [--verbose] FILEPATH

Important flags:

--arch ARCH

Currently supported:

Intel SNB – SPR, AMD ZEN1, ZEN2, ZEN3, ZEN4 Arm TX2, A72, N1, A64FX, TSV110, Grace

--lines L1,L2,L3-L4,L5:L6

Specify lines to analyze (if no markers are used)

--ignore-unknown

Assume **0cy** TP/LAT for unknown instructions

Marking the region of interest

Comment marker

OSACA-BEGIN
.L22:
<pre>vmovapd @(%r13,%rax),%ymm0</pre>
vfmadd213pd (%r14,%rax),%ymm1,%ymm0
<pre>vmovapd %ymm0,(%r12,%rax)</pre>
addq \$32, <mark>%rax</mark>
cmpq %rax,%r15
jne .L22
OSACA-END

// OSACA-BEGIN
.L18:
ldr q2, [x20, x0]
ldr q1, [x21, x0]
fmla v1.2d, v2.2d, v0.2d
str q1, [x19, x0]
add <mark>x0</mark> , <mark>x0</mark> , #16
cmp x22, x0
bne .L18
// OSACA-END

arm

Marking the region of interest

Comment marker

OSACA-BEGIN
<pre>.L22: vmovapd 0(%r13,%rax),%ymm0 vfmadd213pd (%r14,%rax),%ymm1,%ymm0 vmovapd %ymm0,(%r12,%rax)</pre>
addq \$32, <mark>%rax</mark>
cmpq %rax,%r15
jne .L22
OSACA-END

x86

arm

// OSACA-BEGIN
.L18:
ldr q2, [x20, x0]
ldr q1, [x21, x0]
fmla v1.2d, v2.2d, v0.2d
str q1, [x19, x0]
add <mark>x0, x0, #1</mark> 6
cmp x22, x0
bne .L18
// OSACA-END

Insertion tool



ISC25 | Core-Level Performance Engineering Tutorial | NHR@FAU

Marking the region of interest

Comment marker

OSACA-BEGIN
<pre>.L22: vmovapd @(%r13,%rax),%ymm0 vfmadd213pd (%r14,%rax),%ymm1,%ymm vmovapd %ymm0,(%r12,%rax) addq \$32,%rax cmpq %rax,%r15 jne .L22</pre>
OSACA-END
// OSACA-BEGIN
.L18:
ldr q2, [x20, x0]
ldr q1, [x21, x0]
fmla v1.2d, v2.2d, v0.2d
str q1, [x19, x0]
add x0, x0, #16
cmp x22, x0
DNE .L18

x86

arm

Insertion tool



Triad on SPR with OSACA

Recap: Manual analysis resulted in 1 cy/8 it



Triad on SPR with OSACA

Recap: Manual analysis resulted in 1 cy/8 it

* - Instruction micro-ops not bound to a port

X - No throughput/latency information for this instruction in data file

										Por	t i	pressi	ıre	in c	ycl	.es													
 	0 – 0D) V (1 -	1DV	2		3	4	 	5		6	I	7	l 	8	9) 	10		11	<	CP	LCI	D				
2		 ا		 									1			 ا		 ا				11		· I		в2.42:			
3		- 1			0.50		0.50	1			I		1		1						0.50		5		I	vmovups (%r:	4,%rdx,8), %zmm1		
4	0.50	- 1		I	0.50		0.50	1	I	0.50			1		1					I	0.50		4	I	I	vfmadd213pd	(%r15,%rdx,8), %	zmm2, %z	zmm1
5		- 1		1		I		1.	00				1	L.00	1	.00	1.	00		I			0	I	I	vmovupd	%zmm1, (%r12,%rd	x,8)	
6	0.10	I	0.26			I		1	I	0.10)	0.27	I		I			I	0.27	7				1	I	addq	\$8, %rdx		
7	0.00	I	0.34			I		1		0.00)	0.33	I		I	l		I	0.33	3				I	I	cmpq	%rsi, %rdx		
8		I		I		I		I	I		I		I		I	I		I		I				I	*	jb	B2.42		
	0.60		0.60		1.00		1.00	1.	00	0.60)	0.60	1	L.00	1	.00	1.	00	0.60)	1.00		9	1					

Loop-Carried Dependencies	Analysis Report	
6 1.0 addq	\$8, %rdx	[6]

P3

LD AGU

LD

LD LD

P2

LD

AGU

LD

LD

P0

ALU LEA BR

SHFT

AVX ALU

FMA

DIV

FMA

FMA

FMA

Execution Units

P1

ALU LEA

INT MUL

INT

DIV 256b

ALU

256b

FMA

P4

ST

P5

ALU

LEA

INT MUL AVX ALU

FMA

ADD

ADD

STR ADD

P6

ALU

LEA

BR

SHFT

CMP&JMP STR

P7

ST AGU P9

ST AGU

P8

ST

P10

ALU LEA P11

256b LD AGU

Triad on SPR with OSACA

Recap: Manual analysis resulted in 1 cy/8 it

\$ osaca --arch SPR triad.s Open Source Architecture Code Analyzer (OSACA) - 0.6.0 Architecture: SPR

* - Instruction micro-ops not bound to a port

X - No throughput/latency information for this instruction in data file

												Port	pres	sur	e in	cyc	les												
	0	- 01	DV	1 -	1DV	2	I	3	I	4	I	5	6	I	7	I	8	I	9	I	10	I	11	CP	? I	'CD			
2	l		I		I				I		I							I				I					B2.42:		
3	1		I		l	0.50		0.50	1		1			- I				1				1	0.50	5	5		vmovups (%r	14,%rdx,8), %zmm1	
4	0.	.50			I	0.50	(0.50	1		(0.50		- 1				1				1	0.50	4		I	vfmadd213pd	l (%r15,%rdx,8), %zmm2,	%zmm1
5			I		I		I		1:	1.00	I		I	I	1.00		1.00	1	L.00	I		I		0		1	vmovupd	<pre>%zmm1, (%r12,%rdx,8)</pre>	
6	0.	.10	1	0.26	I		I		I		(0.10	0.2	7		I		I		().27	I			I	1	addq	\$8, %rdx	
7	0.	.00	1	0.34	Î				1			0.00	0.3	3		1		1		().33	I		11			cmpq	%rsi, %rdx	
8	Ì		I		ĺ		Ì		Ì		Ì		Ì	Ì		Ì		Ì		Ì		Ì		11	Ì	*	ib		
							· ·		·		·		·			· ·				·		·					2		
	0.	. 60		0.60		1.00		L.00		L.00		0.60	0.6	0	1.00		1.00	1	L.00).60		1.00	9		1			
Lo	0p-qo	Carri	ed D	epend	encie	s Ana	lvs	is Re	oqe	rt																			
	6	1.	0 1	addq		\$8,	۶re	dx				1	[6]																
				-																									

ALU LEA BR Execution Units INT INT MUL AVX ALU BR MUL SHFT SHFT INT AVX ALU DIV 256b FMA ALU FMA 256b DIV FMA FMA ADD LD LD STR FMA ADD ĽĎ LD STR MP&JMP FMA LD LD STR ADD

P4

ST

P5

ALU

LEA

P6

ALU

LEA

P7

ST AGU

P8

ST

P9

ST AGU

P3

LD

AGU

P2

LD

AGU

P0

P1

ALU

LEA

P10

ALU LEA

P11

256b LD AGU





Friedrich-Alexander-Universität Erlangen-Nürnberg

Hands-On #3: Dot Product with OSACA

→ <u>https://go-nhr.de/CLPE-ex3</u>



Hands-On: Benchmarking Dot Product (DP)



 \rightarrow Moodle, hands-on #3





Friedrich-Alexander-Universität Erlangen-Nürnberg

Hands-On #4: PI by integration

→ <u>https://go-nhr.de/CLPE-ex4</u>



Hands-On: PI by integration

ΡΙ $\pi = \int_0^1 \frac{4}{1+x^2} dx$ $= 45^{\circ} = \frac{\pi}{4}$ double delta x = 1./n; $\tan(\alpha) = \frac{1}{1} = 1 \implies \arctan(1) = \frac{\pi}{\Lambda}$ double sum = 0.0;for (int i=0; i<n; i++)</pre> $\Rightarrow \pi = 4 \cdot \arctan(1)$ { x = (i + 0.5) * delta x; $\frac{d}{dx}\arctan(x) = \frac{1}{1+x^2} \Rightarrow \pi = \int_0^1 \frac{4}{1+x^2} dx$ sum += (4.0 / (1.0 + x * x));

\rightarrow Moodle, hands-on #4





Friedrich-Alexander-Universität Erlangen-Nürnberg

Break

1





Friedrich-Alexander-Universität Erlangen-Nürnberg

AArch64 Arm ISA



AArch64 ISA – differences to x86

- Addressing mode: [BASE, INDEX, SCALE]
- Similar to Intel (left) syntax with STORE (STR/STP) as exception
- stp x0, x1, [x2] # mem at x2 < x0, x1</pre>
- 31 general purpose registers (64 bits):
 - x0-x30 (aliases with 32-bit GPRs w0-w30)
 - 32nd register is stack pointer and zero register



AArch64 ISA – differences to x86



- p0-p15, multiples of 16 bits
- optional with predication operation /z, /m, /x (zeroing, merging, don't care)





Friedrich-Alexander-Universität Erlangen-Nürnberg

Case Study: SpMV on A64FX

Sparse Matrix-Vector Multiplication



Based on:

C. Alappat, N. Meyer, J. Laukemann, T. Gruber, G. Hager, G. Wellein, and T. Wettig: *ECM modeling and performance tuning of SpMV and Lattice QCD on A64FX.* Concurrency and Computation: Practice and Experience, e6512 (2021).

DOI: 10.1002/cpe.6512











Sparse Matrix-Vector Multiplication (SpMV): b=Ax

```
for i = 0:nrows-1 //Long outer loop
  for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```



Sparse Matrix-Vector Multiplication (SpMV): b=Ax

```
for i = 0:nrows-1 //Long outer loop
  for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```



Sparse Matrix-Vector Multiplication (SpMV): b=Ax

```
for i = 0:nrows-1 //Long outer loop
  for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```







```
for i = 0:nrows-1 //Long outer loop
for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.16:		
	ld1sw	<pre>z0.d, p0/z, [x17, x20, lsl 2]</pre>
	ld1d	z2.d, p0/z, [x18, x20, lsl 3]
	ld1d	<pre>z3.d, p0/z, [x30, z0.d, lsl 3]</pre>
	add	x20, x20, 8
	fmla	z1.d, p0/m, z3.d, z2.d
	whilelo	p0.d, x20, x14
	b.any	.16
	faddv	d4, p1, z1.d

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.L6:		
†	ld1sw	<pre>z0.d, p0/z, [x17, x20, lsl 2]</pre>
	ld1d	z2.d, p0/z, [x18, x20, lsl 3]
	ld1d	<pre>z3.d, p0/z, [x30, z0.d, lsl 3]</pre>
	add	x20, x20, 8
	fmla	z1.d, p0/m, z3.d, z2.d
	whilelo	p0.d, x20, x14
	b.any	.16
	faddv	d4, p1, z1.d

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.L6:			
Ť	ld1sw	<pre>z0.d, p0/z, [x17, x20, lsl 2]</pre>	load col_idx[j]
	ld1d	z2.d, p0/z, [x18, x20, lsl 3]	
	ld1d	<pre>z3.d, p0/z, [x30, z0.d, lsl 3]</pre>	
	add	x20, x20, 8	
	fmla	z1.d, p0/m, z3.d, z2.d	
	whilelo	p0.d, x20, x14	
	b.any	.16	
	faddv	d4, p1, z1.d	

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.16:			
†	ld1sw	<pre>z0.d, p0/z, [x17, x20, lsl 2]</pre>	load col_idx[j]
	ld1d	z2.d, p0/z, [x18, x20, lsl 3]	load A[j]
	ld1d	<pre>z3.d, p0/z, [x30, z0.d, lsl 3]</pre>	
	add	x20, x20, 8	
	fmla	z1.d, p0/m, z3.d, z2.d	
	whilelo	p0.d, x20, x14	
	b .any	.16	
	faddv	d4, p1, z1.d	

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.16:		
f ld1sw	<pre>z0.d, p0/z, [x17, x20, lsl 2]</pre>	load col_idx[j]
ld1d	z2.d, p0/z, [x18, x20, lsl 3]	load A[j]
ld1d	<pre>z3.d, p0/z, [x30, z0.d, lsl 3]</pre>	gather from x[]
add	x20, x20, 8	
fmla	z1.d, p0/m, z3.d, z2.d	
whilelo	p0.d, x20, x14	
b.any	.16	
faddv	d4, p1, z1.d	

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.L6:			
t	ld1sw	<pre>z0.d, p0/z, [x17, x20, lsl 2]</pre>	load col_idx[j]
	ld1d	z2.d, p0/z, [x18, x20, lsl 3]	load A[j]
	ld1d	<pre>z3.d, p0/z, [x30, z0.d, lsl 3]</pre>	gather from x[]
	add	x20, x20, 8	increase loop counter j
	fmla	z1.d, p0/m, z3.d, z2.d	
	whilelo	p0.d, x20, x14	
	b.any	.16	
	faddv	d4, p1, z1.d	

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.16:		
↑ ld1sw	<pre>z0.d, p0/z, [x17, x20, lsl 2]</pre>	load col_idx[j]
ld1d	z2.d, p0/z, [x18, x20, lsl 3]	load A[j]
ld1d	<pre>z3.d, p0/z, [x30, z0.d, ls1 3]</pre>	gather from x[]
add	x20, x20, 8	increase loop counter j
fmla	z1.d, p0/m, z3.d, z2.d	"the work"
whilelo	p0.d, x20, x14	
b.any	.16	
fadd v	d4, p1, z1.d	

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.16:		
f ld1s	<pre>sw z0.d, p0/z, [x17, x20, lsl 2]</pre>	load col_idx[j]
ldlo	$z_{2.d}, p_{0/z}, [x_{18}, x_{20}, lsl_3]$	load A[j]
ldlo	d z3.d, p0/z, [x30, z0.d, lsl 3] gather from x[]
add	x20, x20, 8	increase loop counter j
fmla	$z_{1.d}, p_{0/m}, z_{3.d}, z_{2.d}$	"the work"
whil	lelo p0.d, x20, x14	loop mechanics
b.ar	ny .L6	
fado	dv d4, p1, z1.d	

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.16:		
f ld1sw	<pre>z0.d, p0/z, [x17, x20, lsl 2]</pre>	load col_idx[j]
ld1d	z2.d, p0/z, [x18, x20, lsl 3]	load A[j]
ld1d	<pre>z3.d, p0/z, [x30, z0.d, lsl 3]</pre>	gather from x[]
add	x20, x20, 8	increase loop counter j
fmla	z1.d, p0/m, z3.d, z2.d	"the work"
whilelo	p0.d, x20, x14	loop mechanics
b.any	.16	
faddv	d4, p1, z1.d	reduction across SIMD register

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```

Assembly of the short inner-loop

.16:		
	ld1sw	<pre>z0.d, p0/z, [x17, x20, lsl 2]</pre>
	ld1d	z2.d, p0/z, [x18, x20, lsl 3]
	ld1d	<pre>z3.d, p0/z, [x30, z0.d, lsl 3]</pre>
	add	x20, x20, 8
	fmla	z1.d, p0/m, z3.d, z2.d
	whilelo	p0.d, x20, x14
	b .any	.16
	faddv	d4, p1, z1.d

```
for i = 0:nrows-1 //Long outer loop
   for j = row_ptr[i]:row_ptr[i+1]-1 // Short inner loop
        b[i] = b[i] + A[j] * x[col_idx[j]]
```
Assembly of the short inner-loop

.16:		
	ld1sw	z0.d, p0/z, [x17, x20, 1s1 2] FMA: Update z1.d
	ld1d	z2.d, p0/z, [x18, x20, 1s1 3]
	ld1d	z3.d, p0/z, [x30, z0.d, 1s1 3] Latency: 9 cycles
	add	x20, x20, 8
	fmla	z1.d, p0/m, z3.d, z2.d
	whilelo	p0.d, x20, x14
	b.any	.16
	faddv	d4, p1, z1.d

In Compressed Row Storage (CRS) format

Assembly of the short inner-loop



Assembly of the short inner-loop



Assembly of the short inner-loop



\$ osaca --arch a64fx spmv-inner-loop.s
[...]

Combined Analysis Report

_ _ _ .

	0 -	• 0DV	1	2	3	4	5	- 5D	6 -	6D	7	CP	LCD	
 11				 I	 I	 I	 I		 I	 I			 I	16:
2			Ì	i i	Ì	i i	0.50	0.50	0.50	0.50		8.0	i i	ld1sw z0.d, p0/z, [x17,x20,lsl 2]
3			I	I	I]	0.50	0.50	0.50	0.50		11	1	ld1d z2.d, p0/z, [x18,x20,lsl 3]
4	1.00		1	1	1.00)	2.00	2.00	2.00	2.00		11.0	I	ld1d z3.d, p0/z, [x30,z0.d,ls1 3]
5	0.00		I	0.00	0.00) 1.00)		ĺ	Ĵ			I	add x20, x20, <mark>8</mark>
6	0.00		I	1.00	l]	I		ĺ	Ĵ		9.0	9.0	fmla z1.d, p0/m, z3.d, z2.d
7			1.00		1.00)						11	1	whilelo p0.d, x20, x14
8			1	1	1	1	1		1	l	1.00	11	I I	b.any .L6
	1.00		1.00	1.00	2.00	$\overline{)}$ 1.00	3.00	3.00	3.00	3.00	1.00	28.0	9.0	

Loop-Carried Dependencies Analysis Report

6	9.0 fmla	z1.d, p0/m, z3.d, z2.d	[6]
5	1.0 add	x20, x20, 8	[5]

\$ osaca --arch a64fx spmv-inner-loop.s
[...]

Combined Analysis Report

	Port pressure in cycles																	
I	0 –	0DV	1	2	2	3	I	4	5 -	· 5D	6	-	6D	7		СР	LCD	1
1					l						J						l	.L6:
2					I		- 1		0.50	0.50	0.5	50	0.50			8.0		<pre> ld1sw z0.d, p0/z, [x17,x20,lsl 2]</pre>
3			1				1		0.50	0.50	0.5	50	0.50	1			1	<pre> ld1d z2.d, p0/z, [x18,x20,lsl 3]</pre>
4	1.00		1		ĺ	1.00	I		2.00	2.00	2.0	00	2.00	1		11.0	I	<pre> ld1d z3.d, p0/z, [x30,z0.d,ls1 3]</pre>
5	0.00		I	0.	00	0.00		1.00]]						l	add x20, x20, 8
61	0.00		1	1.	00		1		l		1			1		9.0	9.0	fmla z1.d, p0/m, z3.d, z2.d
7			1.00			1.00	1]]						l	whilelo p0.d, x20, x14
8			J]					1.00				b.any . <mark>L6</mark>
	1.00		1.00	1.	.00	2.00		1.00	3.00	3.00	3.0	00	3.00	1.00		28.0	9.0	

Loop-Carried Dependencies Analysis Report

6	9.0 fmla	z1.d, p0/m, z3.d, z2.d	[6]
5	1.0 add	x20, x20, <mark>8</mark>	[5]

SELL-C- σ

Idea

- Sort rows according to length within sorting scope σ
- Store nonzeros column-major in zero-padded blocks of height C





- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)



- Inner loop goes down one block column
- Column-major storage within block
 → consecutive access of matrix
- Enables SIMD FMA instructions for column traversal in block and LHS update
- No reductions across SIMD register slots
- Longer inner loop (in assembly) than CRS
- RHS access still indirect (gather)

How to choose the parameters?

- *C*
 - $n \times SIMD$ width to allow good utilization of SIMD units
 - n > 1 useful for hiding ADD pipeline latency

σ

- As small as possible, as large as necessary
- Large σ reduces zero padding
- Sorting alters RHS access pattern

M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. R. Bishop: A *unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units*. SIAM Journal on Scientific Computing **36**(5), C401–C423 (2014). DOI: 10.1137/130930352,



6

_ 6D | 7

|| CP

| LCD

5

- 5D

92		I	I	I	I	I	I.	I	11	I	I	. L4 :
93		I	I	I	I	0.50	0.50 0.50	0.50	11	I	I	ld1sw z16.d, p0/z, [x11]
94		I	I.	I	I	0.50	0.50 0.50	0.50	11	I	I	ld1sw z17.d, p0/z, [x11, #1, mul vl]
95		I	I	I	I	0.50	0.50 0.50	0.50	11	Ι	I	ld1sw z20.d, p0/z, [x11, #2, mul vl]
96		I	I	I	I	0.50	0.50 0.50	0.50	8.	0	Ι	ld1sw z21.d, p0/z, [x11, #3, mul vl]
97	0.00	I	0.00	0.00	1.00	I	I.	I.	11	Ι	Ι	add x10, x10, 32
98	0.00	I	0.00	0.00	1.00	I	L	L	11	I	Ι	add x11, x11, 128
99	0.00	I	0.00	0.00	1.00	I	I	I	11	I.	I	add x12, x12, 256
100		I	I	I	I	0.50	0.50 0.50	0.50	11	I	Ι	ldld z19.d, p0/z, [x12, #-4, mul vl]
101		I	I	I	I	0.50	0.50 0.50	0.50	11	I	Ι	ldld z18.d, p0/z, [x12, #-3, mul vl]
102		I	I	I	I	0.50	0.50 0.50	0.50	11	I	I	ld1d z25.d, p0/z, [x12, #-2, mul vl]
103		I	- I	I	I	0.50	0.50 0.50	0.50	11	I	Ι	ld1d z27.d, p0/z, [x12, #-1, mul vl]
104	1.00	I	- I	1.00	I	2.00	2.00 2.00	2.00	11	I	Ι	ldld z22.d, p0/z, [x3, z16.d, lsl 3]
105	1.00	I	- I	1.00	I	2.00	2.00 2.00	2.00	11	I.	Ι	ldld z23.d, p0/z, [x3, z17.d, lsl 3]
106	1.00	I	- I	1.00	I	2.00	2.00 2.00	2.00	11	I.	Ι	ld1d z24.d, p0/z, [x3, z20.d, 1s1 3]
107	1.00	I	- I	1.00	I	2.00	2.00 2.00	2.00	11.	0	Ι	ldld z26.d, p0/z, [x3, z21.d, lsl 3]
108		1.00	I	1.00	I	I	I	I	11	I	Ι	whilelo p1.d, x10, x9
109	0.00	I	1.00	I	I	I	I	I	11	I	I	fmla z4.d, p0/m, z19.d, z22.d
110	0.00	I	1.00	I	I	I	I	I	11	I	I	fmla z5.d, p0/m, z18.d, z23.d
111	0.00	I	1.00	I	I	I	I	I	11	I	Ι	fmla z6.d, p0/m, z25.d, z24.d
112	0.00	I	1.00	I	I	I	I	I	9.	0 9	0.0	fmla z7.d, p0/m, z27.d, z26.d
113	0.00	I	0.00	0.00	1.00	I	I	I	11	I	Ι	mov p0.b, p1.b
114		I	I	I	I	I	I	1.0	0	I.	Ι	b.any .L4
	4.00	1.00	4.00	5.00	4.00	12.0	12.0 12.0	12.0 1.0	0 28.	0 9.	0	

ISC25 | Core-Level Performance Engineering Tutorial | NHR@FAU

0

- 0DV

2

1

3

Δ

6

_ 6D | 7

|| CP

| LCD

5

- 5D

92		I	I	I	I	I	I.	I	11	I	I	. L4 :
93		I	I	I	I	0.50	0.50 0.50	0.50	11	I	I	ld1sw z16.d, p0/z, [x11]
94		I	I.	I	I	0.50	0.50 0.50	0.50	11	I	I	ld1sw z17.d, p0/z, [x11, #1, mul vl]
95		I	I	I	I	0.50	0.50 0.50	0.50	11	Ι	I	ld1sw z20.d, p0/z, [x11, #2, mul vl]
96		I	I	I	I	0.50	0.50 0.50	0.50	8.	0	Ι	ld1sw z21.d, p0/z, [x11, #3, mul vl]
97	0.00	I	0.00	0.00	1.00	I	I.	I.	11	Ι	Ι	add x10, x10, 32
98	0.00	I	0.00	0.00	1.00	I	L	L	11	I	I	add x11, x11, 128
99	0.00	I	0.00	0.00	1.00	I	I	I	11	I.	I	add x12, x12, 256
100		I	I	I	I	0.50	0.50 0.50	0.50	11	I	Ι	ldld z19.d, p0/z, [x12, #-4, mul vl]
101		I	I	I	I	0.50	0.50 0.50	0.50	11	I	Ι	ldld z18.d, p0/z, [x12, #-3, mul vl]
102		I	I	I	I	0.50	0.50 0.50	0.50	11	I	Ι	ld1d z25.d, p0/z, [x12, #-2, mul vl]
103		I	- I	I	I	0.50	0.50 0.50	0.50	11	I	Ι	ld1d z27.d, p0/z, [x12, #-1, mul vl]
104	1.00	I	- I	1.00	I	2.00	2.00 2.00	2.00	11	I	Ι	ldld z22.d, p0/z, [x3, z16.d, lsl 3]
105	1.00	I	- I	1.00	I	2.00	2.00 2.00	2.00	11	I.	Ι	ldld z23.d, p0/z, [x3, z17.d, lsl 3]
106	1.00	I	- I	1.00	I	2.00	2.00 2.00	2.00	11	I.	Ι	ld1d z24.d, p0/z, [x3, z20.d, 1s1 3]
107	1.00	I	- I	1.00	I	2.00	2.00 2.00	2.00	11.	0	Ι	ldld z26.d, p0/z, [x3, z21.d, lsl 3]
108		1.00	I	1.00	I	I	I	I	11	I	Ι	whilelo p1.d, x10, x9
109	0.00	I	1.00	I	I	I	I	I	11	I	Ι	fmla z4.d, p0/m, z19.d, z22.d
110	0.00	I	1.00	I	I	I	I	I	11	I	I	fmla z5.d, p0/m, z18.d, z23.d
111	0.00	I	1.00	I	I	I	I	I	11	I	Ι	fmla z6.d, p0/m, z25.d, z24.d
112	0.00	I	1.00	I	I	I	I	I	9.	0 9	0.0	fmla z7.d, p0/m, z27.d, z26.d
113	0.00	I	0.00	0.00	1.00	I	I	I	11	I	Ι	mov p0.b, p1.b
114		I	I	I	I	I	I	1.0	0	I.	Ι	b.any .L4
	4.00	1.00	4.00	5.00	4.00	12.0	12.0 12.0	12.0 1.0	0 28.	0 9.	0	

ISC25 | Core-Level Performance Engineering Tutorial | NHR@FAU

0

- 0DV

2

1

3

Δ

6

- 6D | 7 || CP | LCD

5D

						·	·				
92							 I	 I	 	 	.14:
93		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1sw z16.d, p0/z, [x11]
94		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1sw z17.d, p0/z, [x11, #1, mul vl]
95		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1sw z20.d, p0/z, [x11, #2, mul v1]
96		I	I	I	I	0.50	0.50 0.50	0.50	8.0	I I	ld1sw z21.d, p0/z, [x11, #3, mul v1]
97	0.00	I	0.00	0.00	1.00	I	I	I	11	I I	add x10, x10, 32
98	0.00	I	0.00	0.00	1.00	I	I.	I	11	I I	add x11, x11, 128
99	0.00	I	0.00	0.00	1.00	I	I.	I	11	I I	add x12, x12, 256
100		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1d z19.d, p0/z, [x12, #-4, mul vl]
101		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1d z18.d, p0/z, [x12, #-3, mul vl]
102		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1d z25.d, p0/z, [x12, #-2, mul v1]
103		I	I	I	I	0.50	0.50 0.50	0.50	11	l I	ld1d z27.d, p0/z, [x12, #-1, mul vl]
104	1.00	I	I	1.00	I	2.00	2.00 2.00	2.00	11	l I	ld1d z22.d, p0/z, [x3, z16.d, ls1 3]
105	1.00	I	I	1.00	I	2.00	2.00 2.00	2.00	11	l I	ld1d z23.d, p0/z, [x3, z17.d, ls1 3]
106	1.00	I	I	1.00	I	2.00	2.00 2.00	2.00	11	l I	ld1d z24.d, p0/z, [x3, z20.d, ls1 3]
107	1.00	I	I	1.00	I	2.00	2.00 2.00	2.00	11.0	l I	ld1d z26.d, p0/z, [x3, z21.d, ls1 3]
108		1.00	I	1.00	I	I	I	I.	11		whilelo p1.d, x10, x9
109	0.00	I	1.00	I	I	I	I	I.	11	I	fmla z4.d, p0/m, z19.d, z22.d
110	0.00	I	1.00	I	I	I	I	I.	11	I I	fmla z5.d, p0/m, z18.d, z23.d
111	0.00	I	1.00	I	I	I	I	l I	11	I I	fmla z6.d, p0/m, z25.d, z24.d
112	0.00	I	1.00	I	I	I	I	l I	9.0	9.0	fmla z7.d, p0/m, z27.d, z26.d
113	0.00	I	0.00	0.00	1.00	I	I	I	11		mov p0.b, p1.b
114		I	I	I	I	I	I	1.0	00	l I	b.any .L4
	4.00	1.00	4.00	5.00	4.00	12.0	12.0 12.0	12.0 1.0	00 28.0	9.0	

0

- 0DV |

1

2

3

6

- 6D | 7 || CP | LCD

5D

						·	·				
92							 I	 I	 	 	.14:
93		I	I	I	I	0.50	0.50 0.50	0.50	11		ld1sw z16.d, p0/z, [x11]
94		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1sw z17.d, p0/z, [x11, #1, mul vl]
95		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1sw z20.d, p0/z, [x11, #2, mul v1]
96		I	I	I	I	0.50	0.50 0.50	0.50	8.0	I I	ld1sw z21.d, p0/z, [x11, #3, mul v1]
97	0.00	I	0.00	0.00	1.00	I	I	I	11	I I	add x10, x10, 32
98	0.00	I	0.00	0.00	1.00	I	I.	I	11	I I	add x11, x11, 128
99	0.00	I	0.00	0.00	1.00	I	I.	I	11	I I	add x12, x12, 256
100		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1d z19.d, p0/z, [x12, #-4, mul vl]
101		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1d z18.d, p0/z, [x12, #-3, mul vl]
102		I	I	I	I	0.50	0.50 0.50	0.50	11	I I	ld1d z25.d, p0/z, [x12, #-2, mul v1]
103		I	I	I	I	0.50	0.50 0.50	0.50	11	l I	ld1d z27.d, p0/z, [x12, #-1, mul vl]
104	1.00	I	I	1.00	I	2.00	2.00 2.00	2.00	11	l I	ld1d z22.d, p0/z, [x3, z16.d, ls1 3]
105	1.00	I	I	1.00	I	2.00	2.00 2.00	2.00	11	l I	ld1d z23.d, p0/z, [x3, z17.d, ls1 3]
106	1.00	I	I	1.00	I	2.00	2.00 2.00	2.00	11	l I	ld1d z24.d, p0/z, [x3, z20.d, ls1 3]
107	1.00	I	I	1.00	I	2.00	2.00 2.00	2.00	11.0	l I	ld1d z26.d, p0/z, [x3, z21.d, ls1 3]
108		1.00	I	1.00	I	I	I	I.	11		whilelo p1.d, x10, x9
109	0.00	I	1.00	I	I	I	I	I.	11	I	fmla z4.d, p0/m, z19.d, z22.d
110	0.00	I	1.00	I	I	I	I	I.	11	I I	fmla z5.d, p0/m, z18.d, z23.d
111	0.00	I	1.00	I	I	I	I	l I	11	I I	fmla z6.d, p0/m, z25.d, z24.d
112	0.00	I	1.00	I	I	I	I	l I	9.0	9.0	fmla z7.d, p0/m, z27.d, z26.d
113	0.00	I	0.00	0.00	1.00	I	I	I	11		mov p0.b, p1.b
114		I	I	I	I	I	I	1.0	00	l I	b.any .L4
	4.00	1.00	4.00	5.00	4.00	12.0	12.0 12.0	12.0 1.0	00 28.0	9.0	

0

- 0DV |

1

2

3

I	0 - 0DV	1	2	Ι	3	4	Ι	5 -	- 5D)	6 -	- 6D	Ι	7		СР	LCD	I	
92			 I			 I	 I				 I						 I	-	. L4:
93		I	I	Т		I	Ι	0.50	0.5	0	0.50	0.50	Т		П		I	Т	ld1sw z16.d, p0/z, [x11]
94		I	I	Ι		I	Ι	0.50	0.5	0	0.50	0.50	Ι				I.	Ι	ld1sw z17.d, p0/z, [x11, #1, mul v1]
95		I	I	Ι		I	Ι	0.50	0.5	0	0.50	0.50	Ι				T	Ι	ld1sw z20.d, p0/z, [x11, #2, mul v1]
96		I	I	Ι		I	Ι	0.50	0.5	0	0.50	0.50	Ι			8.0	T	Ι	ld1sw z21.d, p0/z, [x11, #3, mul v1]
97	0.00	I	0.00		0.00	1.00	Ι				I		Ι				1	Ι	add x10, x10, 32
98	0.00	I	0.00	I I	0.00	1.00	Ι				I		Ι				T	Ι	add x11, x11, 128
99	0.00	I	0.00		0.00	1.00	Ι				L		Т		П		T	Т	add x12, x12, 256
100		I	I	Т		I	Ι	0.50	0.5	0	0.50	0.50	Т		П		1 I	Ι	ld1d z19.d, p0/z, [x12, #-4, mul v1]
101		I	I	Ι		I	Ι	0.50	0.5	0	0.50	0.50	Ι				T	Ι	ld1d z18.d, p0/z, [x12, #-3, mul v1]
102		I	I	Ι		I	Ι	0.50	0.5	0	0.50	0.50	Ι				T	Ι	ld1d z25.d, p0/z, [x12, #-2, mul v1]
103		I	I	Ι		I	Ι	0.50	0.5	0	0.50	0.50	Ι				T	Ι	ld1d z27.d, p0/z, [x12, #-1, mul v1]
104	1.00	I	I	Ι	1.00	I	Ι	2.00	2.0	0	2.00	2.00	Ι				T	Ι	ldld z22.d, p0/z, [x3, z16.d, lsl 3]
105	1.00	I	I	Ι	1.00	I	Ι	2.00	2.0	0	2.00	2.00	Ι				T	Ι	ld1d z23.d, p0/z, [x3, z17.d, lsl 3]
106	1.00	I	I	Ι	1.00	I	Ι	2.00	2.0	0	2.00	2.00	Ι				T	Ι	ld1d z24.d, p0/z, [x3, z20.d, ls1 3]
107	1.00	I	I	Ι	1.00	I	Ι	2.00	2.0	0	2.00	2.00	Ι			11.0	T	Ι	ld1d z26.d, p0/z, [x3, z21.d, lsl 3]
108		1.00	I	Ι	1.00	I	Ι				I		Ι				1	Т	whilelo p1.d, x10, x9
109	0.00	I	1.00	I		I	Ι				I		Ι				1	Ι	fmla z4.d, p0/m, z19.d, z22.d
110	0.00	I	1.00	I		I	Ι				I		Ι				1	Ι	fmla z5.d, p0/m, z18.d, z23.d
111	0.00	I	1.00	I		I	Ι				I		I		11		1	Ι	fmla z6.d, p0/m, z25.d, z24.d
112	0.00	I	1.00	I		I	Ι				Shift o	of b	ott	len	ecl	.0	J 9.0	Т	fmla z7.d, p0/m, z27.d, z26.d
113	0.00	I	0.00	I	0.00	1.00	Ι				I		I		П		1	I	mov p0.b, p1.b
114		I	I	Ι		I	Ι				I		T	1.00	1		I	I	b.any .L4
	4.00	1.00	4.00		5.00	4.00	Г	12.0	12.	0	12.0	12.0	T	1.00		28.0	9.0		

ISC25 | Core-Level Performance Engineering Tutorial | NHR@FAU

SpMV performance with SELL-C- σ (1 CMG)

- SELL-C-σ separates
 SIMD from sum
 reduction
- C>8 allows for reduction of fmla latency impact







Friedrich-Alexander-Universität Erlangen-Nürnberg

Case Study: Domain Wall (DW) Kernel

from Quantum Chromodynamics (QCD)



Based on:

© Brookhaven National Lab

C. Alappat, N. Meyer, J. Laukemann, T. Gruber, G. Hager, G. Wellein, and T. Wettig: *ECM modeling and performance tuning of SpMV and Lattice QCD on A64FX.* Concurrency and Computation: Practice and Experience, e6512 (2021).

DOI: <u>10.1002/cpe.6512</u>

DW stencil kernel (simplified)

```
#define x p 1 // x-plus direction
                                                              • "Grid" lattice QCD framework
#define x m 2 // x-minus direction

    Uses SVE intrinsics

#define y_p 3 // y-plus direction
                                                              • Data type: double complex
. . .
#pragma omp parallel for schedule(static)
for{t,z,y,x} = 1:{Lt-2,Lz-2,Ly-2,Lx-2} //collapsed loop over 4d space-time
  for(int s=0; s<Ls; ++s) //loop over fifth dimension</pre>
  {
    O[t][z][y][x][s] = R(x_p) \cdot U[x_p][t][z][y][x] \cdot P(x_p) \cdot I[t][z][y][x+1][s] +
                          R(x_m) \cdot U[x_m][t][z][y][x] \cdot P(x_m) \cdot I[t][z][y][x-1][s] +
                          R(y_p) \cdot U[y_p][t][z][y][x] \cdot P(y_p) \cdot I[t][z][y+1][x][s] +
                          R(y_m) \cdot U[y_m][t][z][y][x] \cdot P(y_m) \cdot I[t][z][y-1][x][s] +
                          R(z_p) \cdot U[z_p][t][z][y][x] \cdot P(z_p) \cdot I[t][z+1][y][x][s] +
                          R(z m) \cdot U[z m][t][z][y][x] \cdot P(z m) \cdot I[t][z-1][y][x][s] +
                          R(t_p) \cdot U[t_p][t][z][y][x] \cdot P(t_p) \cdot I[t+1][z][y][x][s] +
                          R(t m) \cdot U[t m][t][z][y][x] \cdot P(t m) \cdot I[t-1][z][y][x][s];
```
Complex numbers data layout choice





Observed performance

- Starting point: AoS layout, ACLE intrinsics, GCC/FCC
- 1320 flops/LUP (theoretical)
- Measured code balance: 1500 byte/LUP

 $B_c \approx 1.14 \frac{\text{byte}}{\text{flop}}$

• A64FX (FX1000): $B_m = 0.25 \frac{\text{byte}}{\text{flop}} \rightarrow \text{expect memory bound}$

Observed performance

- Starting point: AoS layout, ACLE intrinsics, GCC/FCC
- 1320 flops/LUP (theoretical)
- Measured code balance: 1500 byte/LUP

 $B_c \approx 1.14 \frac{\text{byte}}{\text{flop}}$

• A64FX (FX1000): $B_m = 0.25 \frac{\text{byte}}{\text{flop}} \rightarrow \text{expect memory bound}$



\$ osaca --arch a64fx riri-base-gcc.s
[...]

Combined Analysis Report

			Port p	essure in cyc	les				
I 0 - 0	0DV 1	2	3 4	5 – 5D	6 – 6D	7	CP	LCD	
									- 11
560	l.	I I		l	ļ	l			. 141:
561 0.00)	0.00	0.50 0.5	0	Ì	1			lsl w2, w13, 3
562	1	1 1	1	0.50 0.50	0.50 0.50	1		I I	ld1d z16.d, p0/z, [x11]
563 0.00	l I	0.00	0.50 0.5	0	1	1			add x18, sp, 160
564	l l	I I	1	0.50 0.50	0.50 0.50]	11		ld1d z18.d, p0/z, [x11, #-4, mul v1]
565	l I	1 1	0.50 0.5	0	1	1			sxtw x2, w2
566	l l		1	0.50 0.50	0.50 0.50	[ld1d z19.d, p0/z, [x11, #-3, mul v1]
[]									
1367 1.00	l l	I I	1	1.00	1.00]	11		st1d z2.d, p0, [x0, #4, mul v1]
1368 1.00	l l	I I	1	1.00	1.00]	11		st1d z13.d, p0, [x0, #5, mul v1]
1369	l I	1 1	0.00 1.0	0	1	1			cmp w14, w13
1370	I				1	1.00	11		bne .L41
680		500	30 30	118.5 98.5	118.5 98.5	1.0	158	1.0	

Loop-Carried Dependencies Analysis Report

1360 | 1.0 | add w13, w13, 1

| [1360]

\$ osaca --arch a64fx riri-base-gcc.s
[...]

Combined Analysis Report

			Port p	essure in cyc	les				
I 0 - 0	0DV 1	2	3 4	5 – 5D	6 – 6D	7	CP	LCD	
									- 11
560	l.	I I		l	ļ	l			. 141:
561 0.00)	0.00	0.50 0.5	0	Ì	1			lsl w2, w13, 3
562	1	1 1	1	0.50 0.50	0.50 0.50	1		I I	ld1d z16.d, p0/z, [x11]
563 0.00	l I	0.00	0.50 0.5	0	1	1			add x18, sp, 160
564	l l	I I	1	0.50 0.50	0.50 0.50]	11		ld1d z18.d, p0/z, [x11, #-4, mul v1]
565	l I	1 1	0.50 0.5	0	1	1			sxtw x2, w2
566	l I		1	0.50 0.50	0.50 0.50	[ld1d z19.d, p0/z, [x11, #-3, mul v1]
[]									
1367 1.00	l l	I I	1	1.00	1.00]	11		st1d z2.d, p0, [x0, #4, mul v1]
1368 1.00	l l	I I	1	1.00	1.00]	11		st1d z13.d, p0, [x0, #5, mul v1]
1369	l I	1 1	0.00 1.0	0	1	1			cmp w14, w13
1370	l				1	1.00	11		bne .L41
680		500	30 30	118.5 98.5	118.5 98.5	1.0	158	1.0	

Loop-Carried Dependencies Analysis Report

1360 | 1.0 | add w13, w13, 1

| [1360]







\$ osaca --arch a64fx rrii-o1-gcc.s
[...]

Combined Analysis Report

							Po	rt pres	sure	e in	cyc	les									
l I	0 –	0dv	1	I	2	3	}	4	5	-	5D	(5 -	- 61	DI	7	11	С₽	LC	D	
433			1			1		l													.166:
434			1	I		2.	50	2.50	l			1			I]		madd x0, x1, x0, x19
435 I	1.00		1						0.	50		0	0.50		I						str x0, [sp, 1896]
436	0.00		1	0	.00	0.	50	0.50				I			1					1	add x1, x1, x0
437	1.00		1	I		I		1	0.	50		C	0.50		I				I	1	str x1, [sp, 1936]
438			1			0.	50	0.50													cmp x0, x1
[.]																				
2803			1					1	0.	.00 (0.00	1	00	1.00	0					I	ldr x0, [sp, 1784]
2804			1					1	0.	00		1	00		I					I	prfd pldl2strm, p0, [x0]
2805			I]							1	1.00					b.164
2806			1			1		1				1			1				1	1	.L38:
2807	0.00		i.	0	.00	0.	50	0.50				Ì			1				1		add x1, x1, 1
28081	0.00		1	0	.00	0.	00	1.00				1			I				1	1	mov x19, 0
28091			i	Ì		Ì		I	1			i			Ì	1.00				i	b.166
	567		1.0	5	67	24	7	247	488	3.5 2	275.	5 48	88.5	275	.5	14	••	92	1.0	•	
Loop-	Carri	ed De	pend	enci	es A	naly	sis	Report	:												

| [507]

507 | 1.0 | add sp, sp, 2048









DW kernel optimizations



Summary of optimizations for DW

- AoSoA (RRII) data layout
 - Prevents use of complex arithmetic instructions fcmla/fcadd
 - Removes imbalance between FLA and FLB ports in the core
 - Some register spills occur, but still better than AoS (RIRI)
 - More instructions but better performance

- Software prefetching decreases L2 data volume
- -O1 makes compiler obey the ordering hints in the computational kernel (more efficient OoO execution)





Friedrich-Alexander-Universität Erlangen-Nürnberg

Hands-On #5: 2D Gauss-Seidel analysis

→ <u>https://go-nhr.de/CLPE-ex5</u>



- Limited by loop-carried dependency
- Create code with -Ofast, -funroll-loops



Analyze for SPR

```
for(int it=0; it<NITER; ++it) {
  for (int i=1; i<NI-1; ++i) {
    for (int k=1; k<NK-1; ++k) {
      phi[i][k] = 0.25 * (
         phi[i][k-1] + phi[i+1][k] +
         phi[i][k+1] + phi[i-1][k]
      );
      }
}</pre>
```

- Limited by loop-carried dependency
- Create code with -Ofast, -funroll-loops



Analyze for SPR

```
for(int it=0; it<NITER; ++it) {
  for (int i=1; i<NI-1; ++i) {
    for (int k=1; k<NK-1; ++k) {
      phi[i][k] = 0.25 * (
         phi[i][k-1] + phi[i+1][k] +
         phi[i][k+1] + phi[i-1][k]
      );
      }
}</pre>
```

- Limited by loop-carried dependency
- Create code with -Ofast, -funroll-loops



Analyze for SPR

```
for(int it=0; it<NITER; ++it) {
  for (int i=1; i<NI-1; ++i) {
    for (int k=1; k<NK-1; ++k) {
      phi[i][k] = 0.25 * (
         phi[i][k-1] + phi[i+1][k] +
         phi[i][k+1] + phi[i-1][k]
      );
      }
}</pre>
```

- Limited by loop-carried dependency
- Create code with -Ofast, -funroll-loops



Analyze for SPR

```
for(int it=0; it<NITER; ++it) {
  for (int i=1; i<NI-1; ++i) {
    for (int k=1; k<NK-1; ++k) {
      phi[i][k] = 0.25 * (
         phi[i][k-1] + phi[i+1][k] +
         phi[i][k+1] + phi[i-1][k]
      );
    }
}</pre>
```

Hands-On: Gauss-Seidel Method – standard version

\$ osaca --arch spr --export-graph dependencies.dot gs.s
\$ dot -Tpdf dependencies.dot -o dep graph.pdf

0 4: vaddsd [4] 5.0 5: vaddsd 6: vaddsd 22: vmulsd 23: vmovsd 21: LOAD 5.0 21: vaddsd 5: LOAD 7: vmulsd 8: vmovsd 20: LOAD 16: LOAD 5.0 15: LOAD 5.0 20: vaddsd 11: LOAD 18: vmovsd 9: vaddsd 9: LOAD 5.0 16: vaddsd 15: vaddsd 17: vmuls 13: vmovsd 11: vaddsd 12: vmuls 10: vaddsd 19: vaddso 18: LOAD 5.0 19: LOAD 14: vaddsd 5.0 14: LOAD LCD

2		vmovsd	(%rsi,%r9), %xmm2
3		incq	<mark>%rdx</mark>
4		vaddsd	8(%rsi,%r10), %xmm2, %xmm3
5		vaddsd	16(%rsi,%r9), %xmm3, %xmm4
6	2.0	vaddsd	<pre>%xmm1, %xmm4, %xmm1</pre>
7	4.0	vmulsd	%xmm1, %xmm0, %xmm5
8		vmovsd	%xmm5, 8(%rsi,%r9)
9	2.0	vaddsd	(%rsi,%r10), %xmm5, %xmm6
10	2.0	vaddsd	8(%rsi,%r11), %xmm6, %xmm7
11	2.0	vaddsd	16(%rsi,%r10), %xmm7, %xmm8
12	4.0	vmulsd	<pre>%xmm8, %xmm0, %xmm9</pre>
13		vmovsd	%xmm9, 8(%rsi,%r10)
14	2.0	vaddsd	(%rsi,%r11), %xmm9, %xmm10
15	2.0	vaddsd	8(%rsi,%r8), %xmm10, %xmm11
15 16	2.0 2.0	vaddsd vaddsd	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12
15 16 17	2.0 2.0 4.0	vaddsd vaddsd vmulsd	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12 %xmm12, %xmm0, %xmm13
15 16 17 18	2.0 2.0 4.0	vaddsdvaddsdvmulsdvmovsd	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12 %xmm12, %xmm0, %xmm13 %xmm13, 8(%rsi,%r11)
15 16 17 18 19	2.0 2.0 4.0 2.0	vaddsdvaddsdvmulsdvmovsdvaddsd	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12 %xmm12, %xmm0, %xmm13 %xmm13, 8(%rsi,%r11) (%rsi,%r8), %xmm13, %xmm14
15 16 17 18 19 20	2.0 2.0 4.0 2.0 2.0	vaddsdvaddsdvmulsdvmovsdvaddsdvaddsd	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12 %xmm12, %xmm0, %xmm13 %xmm13, 8(%rsi,%r11) (%rsi,%r8), %xmm13, %xmm14 8(%rsi,%r14), %xmm14, %xmm15
15 16 17 18 19 20 21	2.0 2.0 4.0 2.0 2.0 2.0	vaddsdvaddsdvmulsdvmovsdvaddsdvaddsdvaddsdvaddsd	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12 %xmm12, %xmm0, %xmm13 %xmm13, 8(%rsi,%r11) (%rsi,%r8), %xmm13, %xmm14 8(%rsi,%r14), %xmm14, %xmm15 16(%rsi,%r8), %xmm15, %xmm16
15 16 17 18 20 21 22	2.0 2.0 4.0 2.0 2.0 2.0 2.0 4.0	vaddsdvaddsdvmulsdvmovsdvaddsdvaddsdvaddsdvaddsdvaddsdvaddsd	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12 %xmm12, %xmm0, %xmm13 %xmm13, 8(%rsi,%r11) (%rsi,%r8), %xmm13, %xmm14 8(%rsi,%r14), %xmm14, %xmm15 16(%rsi,%r8), %xmm15, %xmm16 %xmm16, %xmm0, %xmm1
15 16 17 18 19 20 21 22 23	2.0 2.0 4.0 2.0 2.0 2.0 4.0	vaddsd vaddsd vmulsd vmovsd vaddsd vaddsd vaddsd vaddsd vmulsd vmulsd vmovsd	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12 %xmm12, %xmm0, %xmm13 %xmm13, 8(%rsi,%r11) (%rsi,%r8), %xmm13, %xmm14 8(%rsi,%r14), %xmm14, %xmm15 16(%rsi,%r8), %xmm15, %xmm16 %xmm16, %xmm0, %xmm1 %xmm1, 8(%rsi,%r8)
15 16 17 18 19 20 21 22 23 24	2.0 2.0 4.0 2.0 2.0 2.0 4.0	vaddsd vaddsd vmulsd vmovsd vaddsd vaddsd vaddsd vaddsd vmulsd vmovsd addq	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12 %xmm12, %xmm0, %xmm13 %xmm13, 8(%rsi,%r11) (%rsi,%r8), %xmm13, %xmm14 8(%rsi,%r14), %xmm14, %xmm15 16(%rsi,%r8), %xmm15, %xmm16 %xmm16, %xmm0, %xmm1 %xmm1, 8(%rsi,%r8) %r13, %rsi
15 16 17 18 19 20 21 22 23 24 25	2.0 2.0 4.0 2.0 2.0 2.0 4.0	vaddsd vaddsd vmulsd vmovsd vaddsd vaddsd vaddsd vaddsd vmulsd vmovsd cmpq	8(%rsi,%r8), %xmm10, %xmm11 16(%rsi,%r11), %xmm11, %xmm12 %xmm12, %xmm0, %xmm13 %xmm13, 8(%rsi,%r11) (%rsi,%r8), %xmm13, %xmm14 8(%rsi,%r14), %xmm14, %xmm15 16(%rsi,%r8), %xmm15, %xmm16 %xmm16, %xmm0, %xmm1 %xmm1, 8(%rsi,%r8) %r13, %rsi %r12, %rdx

26: jb

ш

24: addq [1]

3: incq 2: vmovso

4: LOAD

1: lobel

1 25: cmpq

Hands-On: Gauss-Seidel Method – standard version

\$ osaca --arch spr --export-graph dependencies.dot gs.s
\$ dot -Tpdf dependencies.dot -o dep graph.pdf

0 4: vaddsd [41 5.0 5: vaddsd 6: vaddsd 22: vmulsd 23: vmovso 21: LOAD 5.0 21: vaddsd 5: LOAD 7: vmulsd 8: vmovsd 20: LOAD 16: LOAD 5.0 15: LOAD 5.0 20: vaddsd 11: LOAD 18: vmovsd 9: vaddsd 9: LOAD 15: vaddsd 16: vaddsd 17: vmuls 13: vmovsd 11: vaddsd 12: vmuls 10: vaddsd 19: vadds 18: LOAD 5.0 19: LOAD 14: vaddsd 5.0 14: LOAD LCD

2	1	vmovsd	(%rsi,%r9), %xmm2
3		incq	%rdx
4		vaddsd	8(%rsi,%r10), %xmm2, %xmm3
5	İ	vaddsd	16(%rsi,%r9), %xmm3, %xmm4
6	2.0	vaddsd	<pre>%xmm1, %xmm4, %xmm1</pre>
7	4.0	vmulsd	%xmm1, %xmm0, %xmm5
8		vmovsd	%xmm5, 8(%rsi,%r9)
9	2.0	vaddsd	(%rsi,%r10), %xmm5, %xmm6
10	2.0	vaddsd	8(%rsi,%r11), %xmm6, %xmm7
11	2.0	vaddsd	16(%rsi,%r10), %xmm7, %xmm8
12	4.0	vmulsd	%xmm8, %xmm0, %xmm9
13		vmovsd	%xmm9, 8(%rsi,%r10)
14	2.0	vaddsd	(%rsi,%r11), %xmm9, %xmm10
15	2.0	vaddsd	8(%rsi,%r8), %xmm10, %xmm11
16	2.0	vaddsd	16(%rsi,%r11), %xmm11, %xmm12
17	4.0	vmulsd	%xmm12, %xmm0, %xmm13
18		vmovsd	%xmm13, 8(%rsi,%r11)
19	2.0	vaddsd	(%rsi,%r8), %xmm13, %xmm14
20	2.0	vaddsd	8(%rsi,%r14), %xmm14, %xmm15
21	2.0	vaddsd	16(%rsi,%r8), %xmm15, %xmm16
22	4.0	vmulsd	<pre>%xmm16, %xmm0, %xmm1</pre>
23		vmovsd	<pre>%xmm1, 8(%rsi,%r8)</pre>
24		addq	%r13, %rsi
25		cmpq	%r12, %rdx
	36.0		

26: jb

ш

24: addq [1]

3: incq 2: vmovso

4: LOAD

1: lobel

1 25: cmpq

dep chain of 36 cy out of 42 of CP → ratio 0.857

Hands-On: Gauss-Seidel Method – opt. version (SPR)

-Ofast / -03 17: jb [1] 15: addq 4: vmovsd [1] 5.0 6: vaddsd 11: vaddsd 3: inco ▶ 16: cmpq 6: LOAD 5.0 12: vaddsd 12: LOAD 2: vmovsd 5: vaddsd 4 10: vmovsd 9: vmulsd 5: LOAD 7: vaddsd 8: vaddsd 5.0 13: vmulsd 14: vmovsd [4] 7: LOAD 1: label



	LCD		
1		B1.72:	
2		vmovsd	8(%r10,%r11), %xmm2
3		incq	8rdx
4		vmovsd	16(%r10,%r11), %xmm5
5	I	vaddsd	16(%r10,%rsi), %xmm2, %xmm3
6		vaddsd	24(%r10,%rsi), %xmm5, %xmm7
7	1	vaddsd	8(%r10,%r13), %xmm3, %xmm4
8	2.0	vaddsd	<pre>%xmm1, %xmm4, %xmm1</pre>
9	4.0	vmulsd	<pre>%xmm0, %xmm1, %xmm6</pre>
10		vmovsd	%xmm6, 8(%r10,%rsi)
11	2.0	vaddsd	%xmm7, %xmm6, %xmm8
12	2.0	vaddsd	16(%r10,%r13), %xmm8, %xmm9
13	4.0	vmulsd	%xmm0, %xmm9, %xmm1
14		vmovsd	<pre>%xmm1, 16(%r10,%rsi)</pre>
15		addq	\$16, %r10
16		cmpq	%r15, %rdx
17		* jb _	B1.72
	14.0	-	

	LCD		
1		B1.34:	
2		vmovsd	(%rsi,%rdi,8), %xmm0
3		vaddsd	8(%rcx,%rdi,8), %xmm0, %xmm1
4		vaddsd	(%rax,%rdi,8), %xmm1, %xmm2
5	2.0	vaddsd	%xmm3, %xmm2, %xmm3
6	4.0	vmulsd	.L_2il0floatpacket.0(%rip), %xmm3, %xmm3
7		vmovsd	<pre>%xmm3, (%rcx,%rdi,8)</pre>
8		incq	8rdi
9		cmpq	%r13, %rdi
10		* jl	B1.34
	6.0		

Hands-On: Gauss-Seidel Method – opt. version (SPR)





	LCD		
1		B1.72:	
2		vmovsd	8(%r10,%r11), %xmm2
3		incq	Srdx .
4		vmovsd	16(%r10,%r11), %xmm5
5	I	vaddsd	16(%r10,%rsi), %xmm2, %xmm3
6	ĺ	vaddsd	24 (%r10,%rsi), %xmm5, %xmm7
7	1	vaddsd	8(%r10,%r13), %xmm3, %xmm4
8	2.0	vaddsd	8xmm1, 8xmm4, 8xmm1
9	4.0	vmulsd	%xmm0, %xmm1, %xmm6
10		vmovsd	%xmm6, 8(%r10,%rsi)
11	2.0	vaddsd	<pre>%xmm7, %xmm6, %xmm8</pre>
12	i 2.0 i	vaddsd	16(%r10,%r13), %xmm8, %xmm9
13	4.0	vmulsd	<pre>%xmm0, %xmm9, %xmm1</pre>
14		vmovsd	<pre>%xmm1, 16(%r10,%rsi)</pre>
15		addq	\$16, %r10
16		cmpq	%r15, %rdx
17	ĺ	t * jb	B1.72 don chain
	14.0		uep chain

	LCD		
1		B1.34:	
2		vmovsd	(%rsi,%rdi,8), %xmm0
3		vaddsd	8(%rcx,%rdi,8), %xmm0, %xmm1
4		vaddsd	(%rax,%rdi,8), %xmm1, %xmm2
5	2.0	vaddsd	%xmm3, %xmm2, %xmm3
6	4.0	vmulsd	.L_2il0floatpacket.0(%rip), %xmm3, %xmm3
7		vmovsd	<pre>%xmm3, (%rcx,%rdi,8)</pre>
8		incq	8rdi
9		cmpq	%r13, %rdi
10		* jl	B1.34
	6.0	-	

dep chain of 14 cy / 23 cy CP → 0.609

Hands-On: Gauss-Seidel Method – opt. version (SPR)





	LCD		
1		B1.72:	
2		vmovsd	8(%r10,%r11), %xmm2
3		incq	<mark>%rdx</mark>
4		vmovsd	16(%r10,%r11), %xmm5
5		vaddsd	16(%r10,%rsi), %xmm2, %xmm3
6		vaddsd	24(%r10,%rsi), %xmm5, %xmm7
7		vaddsd	8(%r10,%r13), %xmm3, %xmm4
8	2.0	vaddsd	<pre>%xmm1, %xmm4, %xmm1</pre>
9	4.0	vmulsd	<pre>%xmm0, %xmm1, %xmm6</pre>
10		vmovsd	%xmm6, 8(%r10,%rsi)
11	2.0	vaddsd	<pre>%xmm7, %xmm6, %xmm8</pre>
12	2.0	vaddsd	16(%r10,%r13), %xmm8, %xmm9
13	4.0	vmulsd	<pre>%xmm0, %xmm9, %xmm1</pre>
14		vmovsd	<pre>%xmm1, 16(%r10,%rsi)</pre>
15		addq	\$16, %r10
16		cmpq	%r15, %rdx
17		* jb	·· ^{B1.72} den chain of
	14.0		
			14 cv / 23 cv CP 📤 0 600



dep chain of 6 cy / 15 cy CP → 0.4

Summary & Caveats

- A code analyzer helps you to predict the **in-core** runtime of a basic block
 - Might be sufficient, but often a full analysis requires a **memory model** as well!
- An analysis of (loop-carried-)dependencies can help you find performance limitations!
- Analysis is done on compiler-generated code which always holds a factor of uncertainty
- There might be additional things slowing you, e.g.:
 - Cache trashing
 - Loads across cache lines
 - Front end limitations
 - Bank conflicts
 - •

There is not just THE one code analyzer

- OSACA: <u>https://github.com/RRZE-HPC/OSACA</u>
- uiCA: <u>https://www.uops.info/uiCA.html</u>
- LLVM-MCA: <u>https://llvm.org/docs/CommandGuide/llvm-mca.html</u>
- IACA (EoL): <u>https://www.intel.com/content/www/us/en/developer/articles/tool/architectur</u> <u>e-code-analyzer.html</u>





Friedrich-Alexander-Universität Erlangen-Nürnberg

OS ACA

Thank you! Questions?



https://github.com/RRZE-HPC/osaca

pip: \$ pip install -u osaca

Compiler Explorer: https://godbolt.org

