AN EMPIRICAL STUDY OF SOME X86 SIMD INTEGER EXTENSIONS

Isabelle Hurbain and Georges-André Silber
École des mines de Paris
Centre de recherche en informatique
CPC'06 - January, 9th 2006

OUTLINE

- Motivations
- Overview of SSE2 instructions
- Benchmarking SSE2 instructions
- Guidelines for optimization
- Conclusion

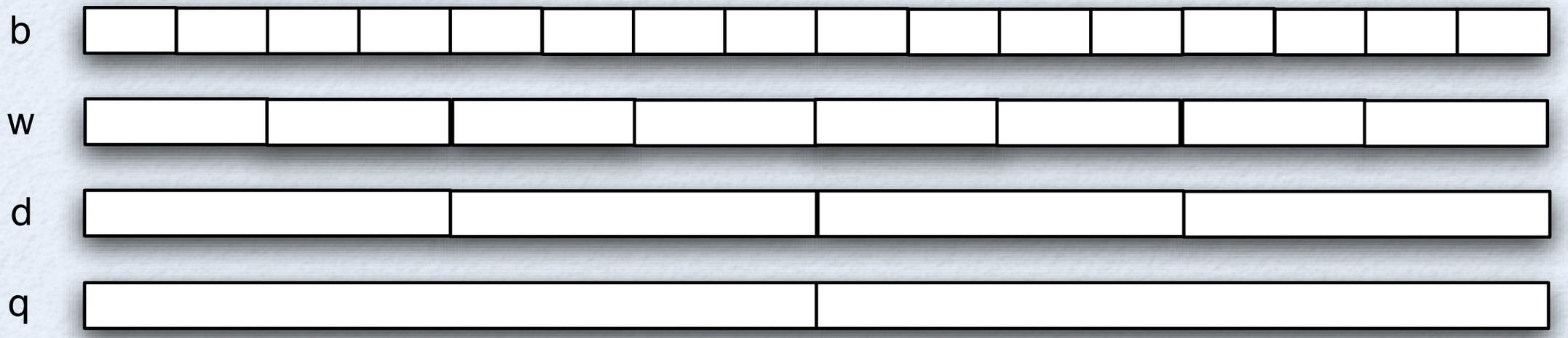
Motivations

- What speedup is attainable with integer SSE2 instructions?
- Measure the best-case impact of 32 SSE2 integer instructions in terms of performance
- Study the code generation for those instructions in state-of-the-art compilers

SSE2 EXTENSION: AN OVERVIEW

- Processor instructions introduced to improve multimedia applications and floating-point operations
- 8 new 128-bit multi-purpose registers
- Integer, floating point, memory instructions
- Similar extensions: Altivec (PowerPC), VIS (Sparc), 3DNow! (Athlon)...

INTEGER OPERATIONS

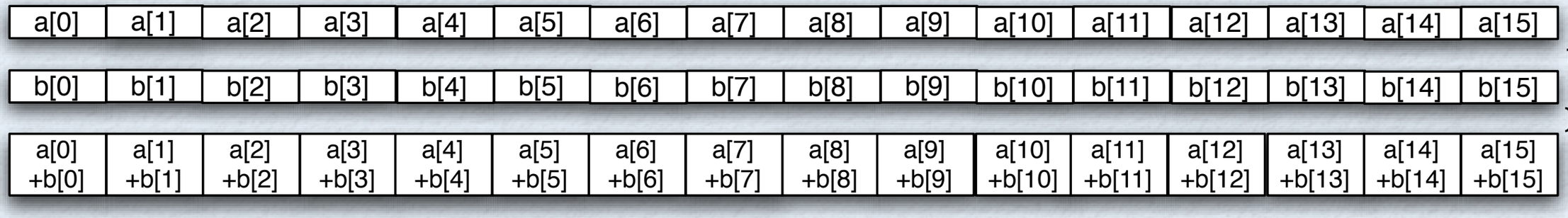


- Basic operations (+, -, *)
- Basic operations with saturation
- Comparison operations (min, max, >, =)
- Miscellaneous operations

Programming with SSE2 instructions

- Writing assembly code
- Using "intrinsics"
- Using the compiler to semi-automatically vectorize the code
- Letting the compiler automatically vectorize the code

EXAMPLE: PADDB

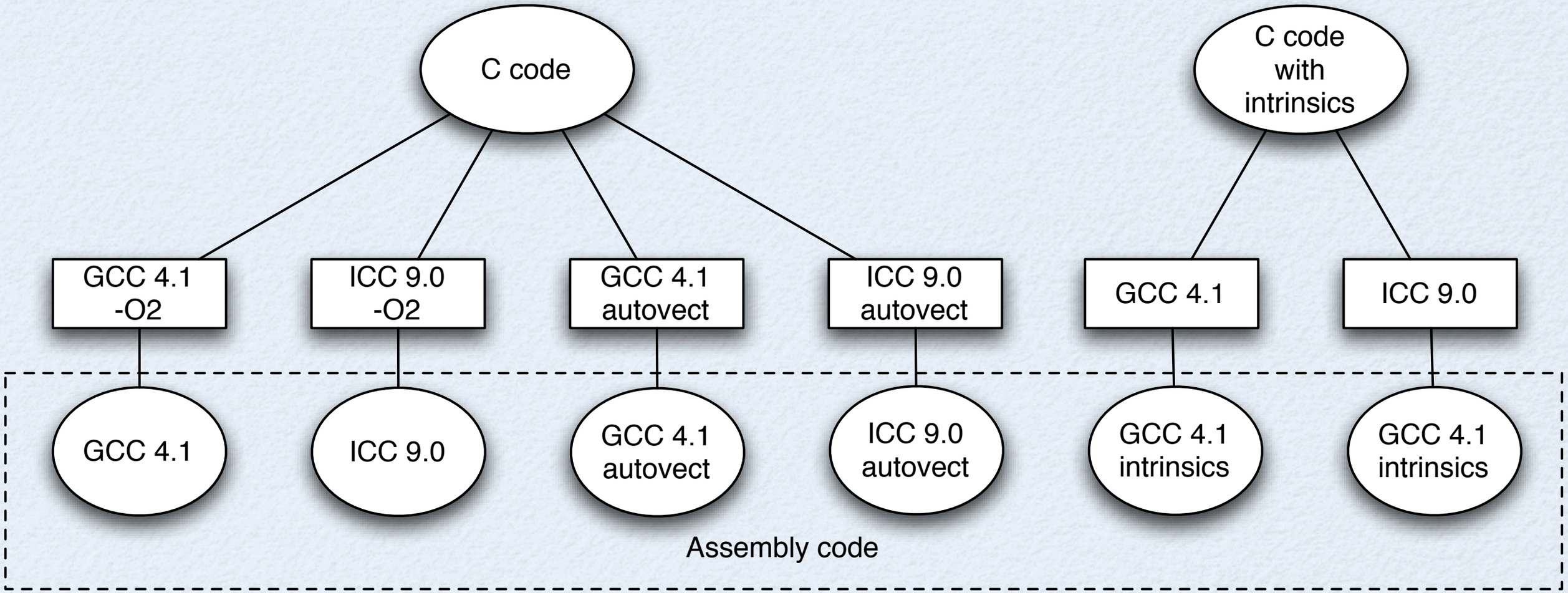



```
for (i=0; i<16;i++){
  c[i] = a[i] + b[i]
}
```

```
*c = _mm_add_epi8(*(__m128i *) a, *(__m128i *) b);
```

```
movl    12(%ebp), %eax
movdqa  (%eax), %xmm0
movl    8(%ebp), %eax
paddb   (%eax), %xmm0
movl    16(%ebp), %eax
movdqa  %xmm0, (%eax)
```

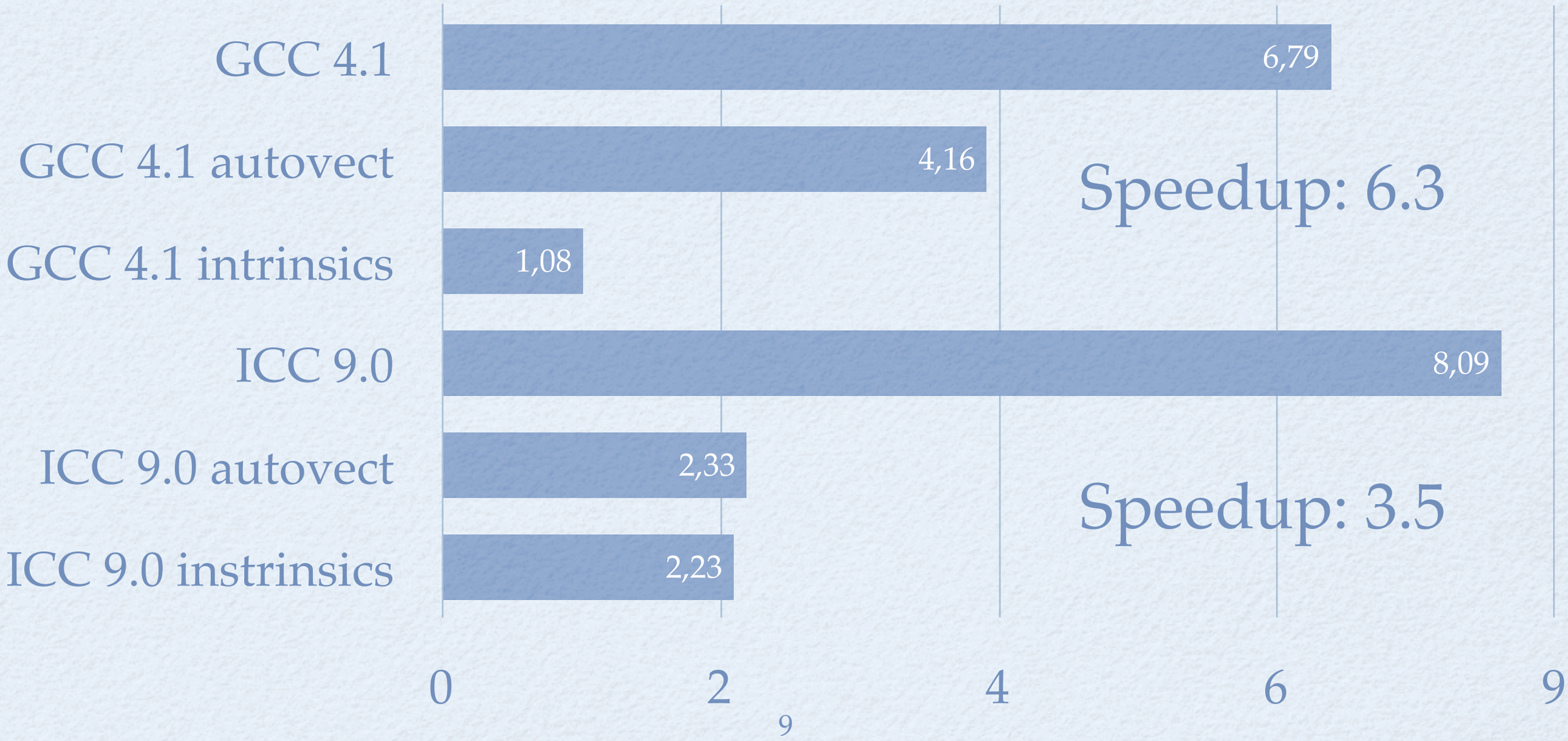
BENCHMARKING



- 10 executions of 30,000,000 calls

PADDB : A GOOD CASE

```
for (i = 0; i<16; i++) {
   c[i] = a[i] + b[i];
}
```

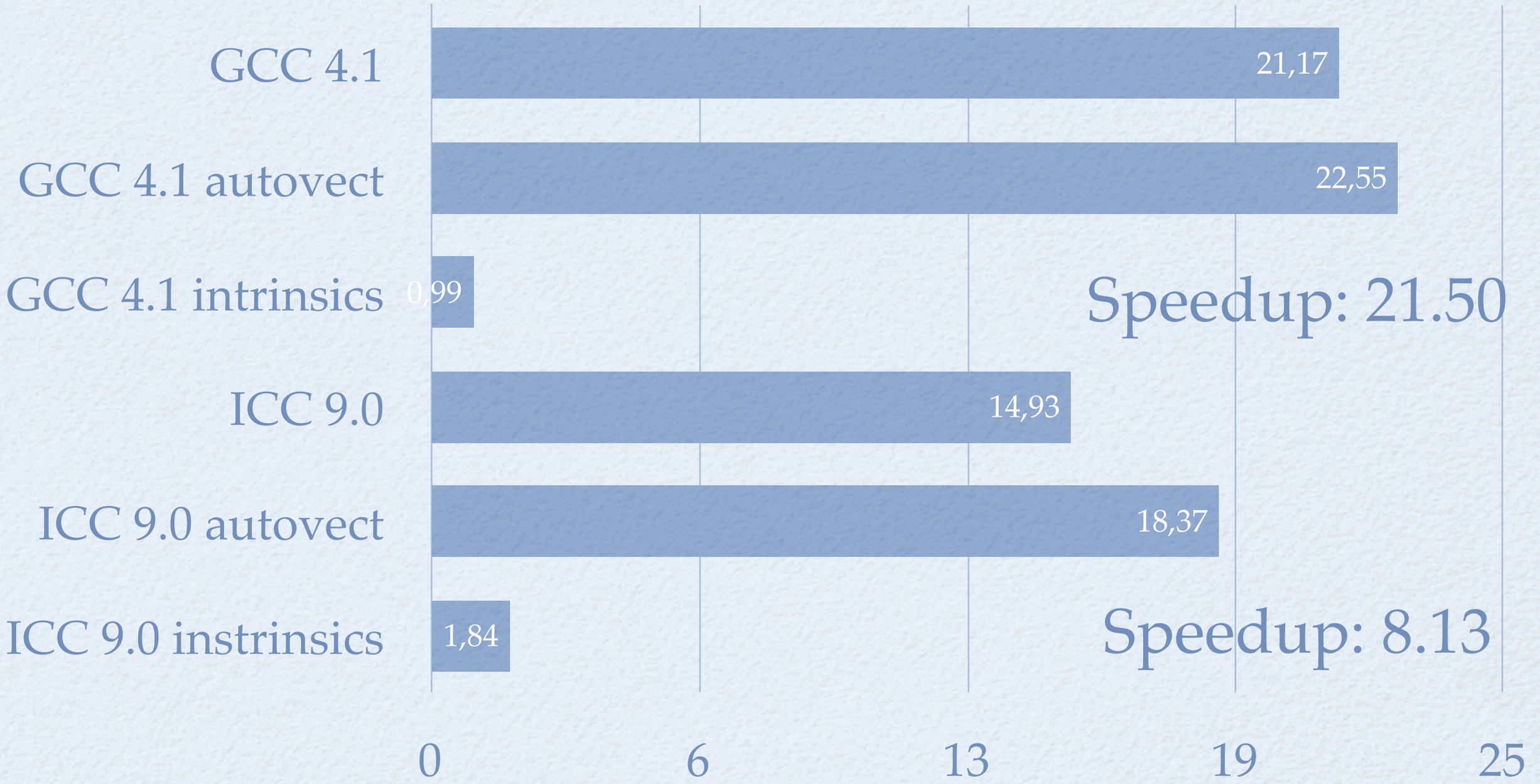


PSADBW : A CASE THAT IS NOT DETECTED

SRC	X15	X14	X13	X12	X11	X10	X9	X8	X7	X6	X5	X4	X3	X2	X1	X0
DEST	Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
TEMP	ABS (X15-Y15)	ABS (X14-Y14)	ABS (X13-Y13)	ABS (X12-Y12)	ABS (X11-Y11)	ABS (X10-Y10)	ABS (X9-Y9)	ABS (X8-Y8)	ABS (X7-Y7)	ABS (X6-Y6)	ABS (X5-Y5)	ABS (X4-Y4)	ABS (X3-Y3)	ABS (X2-Y2)	ABS (X1-Y1)	ABS (X0-Y0)
DEST	00H	00H	00H	00H	00H	00H	SUM (TEMP15, ...TEMP8)		00H	00H	00H	00H	00H	00H	SUM (TEMP7, ...TEMP0)	

```
for(i = 0; i<8; i++)
  c[i] = 0;
for(i = 0; i<16; i++)
  tmparray[i] = abs(a[i] - b[i])
for(i=0; i<8; i++){
  c[0] += tmparray[i];
  c[4] += tmparray[i+8];
}
```

PSADBW: A CASE THAT IS NOT DETECTED



RESULTS

- Number of detected vectorizable code:
  - ICC : 26 / 32
  - GCC : 8 / 32
- Quality of the semi-automatically vectorized code: ICC is a clear winner

RESULTS

Operation	GCC 4.1						ICC 9.0					
	Base (s)	Autovect	Speedup	Status	Intrinsics	Speedup	Base (s)	Autovect	Speedup	Status	Intrinsics	Speedup
paddb	6.79	4.16	1.63	✓	1.08	6.26	8.09	2.33	3.48	✓	2.29	3.53
psubb	7.25	3.83	1.89	✓	1.09	6.68	7.70	2.34	3.28	✓	2.14	3.60
paddsb	14.56	14.87	0.98	✗	1.08	13.79	15.06	2.29	6.29	✓	2.01	7.50
psubsb	14.55	13.79	1.06	✗	0.96	15.21	14.87	1.85	8.06	✓	1.78	8.35
pcmpgtb	17.61	14.26	1.23	✗	0.87	20.31	13.05	1.82	7.16	✓	1.93	6.75
pminub	10.60	12.11	0.88	✗	1.02	10.41	10.38	11.66	0.89	✗	1.82	5.71
pavgb	9.95	9.79	1.02	✗	1.33	7.50	10.98	1.78	6.18	✓	1.82	6.04
pavgw	6.18	5.88	1.05	✗	0.93	6.62	5.48	2.08	2.63	✓	2.11	2.59
psadbw	21.17	22.54	0.94	✗	0.99	21.50	14.93	18.37	0.81	✗	1.84	8.13
pmaddwd	5.00	5.39	0.93	✗	1.22	4.10	6.39	6.39	0.75	✗	2.07	2.31

GUIDELINES FOR SSE2 OPTIMIZATION

- When possible, **use intrinsics**
- Direct assembly code is a bit faster but much more error-prone
- Autovectorization: the compiler needs to be "helped"

CONCLUSION

- As of today, automatic use of SSE2 instructions by compilers does not work
  - needs interprocedural informations about alignment, overlaps...
- Source to source transformation of the code (producing intrinsics) is a good approach
- Created a set of benchmarks that could be used as testcases for compilers

QUESTIONS