An evaluation of the automatic generation of parallel X86 SIMD integer instructions by GCC and ICC

Isabelle Hurbain

November, 15th 2005

1 Introduction

GIMPLE is defined in [2]. It is an intermediate form between C code and assembly generated by GCC.

Intel defines in [1] SIMD instructions for mainline processors such as Pentium 4. These instructions belong to the MMX and SSE2 sets of instructions. In this report, we define a mapping between those two formalisms. The objective is to be able to detect the following constructions in GIMPLE to generate optimized assembly. We consider the cases where the MMX and SSE provide a theorical speedup and those where it does not.

2 Experimental framework

We consider the simplest code possible, corresponding to a single SIMD assembly instruction (not considering the load and store operations).

The benchmarks are operated as following:

- compilation with `gcc (GCC) 3.3.5 (Debian 1:3.3.5-6)`
- compilation with `gcc (GCC) 4.0.0`
- compilation with `gcc (GCC) 4.1.0 20050424 (experimental)`
- compilation with `icc 8.1`

We use the following options :

GCC 3.3.5	`-O2 -msse2`
GCC 4.0.0	`-O2 -msse2 -ftree-vectorize`
GCC 4.1.0	`-O2 -msse2 -ftree-vectorize`
ICC 8.1	`-march=pentium4 -axN -nolib_inline -ip_no_inlining -O2 -vec_report2`

The indicated code is run 10 times, the corresponding profilings are meant out and give the results listed below.

All executions have been run on `nantes`, a computer with a 3 GHz Intel Pentium 4, HyperThreading enabled and 2 Go of RAM.

3 Basic arithmetic operations

3.1 paddb - MMX (64 bits registers) version

3.1.1 C code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      c[i] = a[i] + b[i];
    }
}
```

3.1.2 GIMPLE code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i=0;
  loop_label::
  if(i >= 8)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.1.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[8], char b[8], char c[8])
{
  *(__m64 *) c = _mm_add_pi8(*(__m64*) a,  *(__m64*) b);
}
```

3.1.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %edx`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`pushl %esi`
`movl 8(%ebp), %eax`	`.L2:`	`movl 16(%ebp), %ecx`
`paddb (%eax), %mm0`	`cmpl $7, -4(%ebp)`	`pushl %ebx`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 8(%ebp), %esi`
`movq %mm0, (%eax)`	`jmp .L1`	`movl 12(%ebp), %ebx`
`popl %ebp`	`.L5:`	`.p2align 4,,15`
`ret`	`movl -4(%ebp), %eax`	`.L2:`
	`movl 16(%ebp), %ecx`	`movzbl -1(%ebx,%edx), %eax`
	`addl %eax, %ecx`	`addb -1(%esi,%edx), %al`
	`movl -4(%ebp), %eax`	`movb %al, -1(%ecx,%edx)`
	`movl 8(%ebp), %edx`	`incl %edx`
	`addl %eax, %edx`	`cmpl $9, %edx`
	`movl -4(%ebp), %eax`	`jne .L2`
	`addl 12(%ebp), %eax`	`popl %ebx`
	`movzbl (%eax), %eax`	`popl %esi`
	`addb (%edx), %al`	`popl %ebp`
	`movb %al, (%ecx)`	`ret`
	`leal -4(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`ret`	

3.1.5 Benchmark

```
  char a[8] __attribute__((aligned));
  char b[8] __attribute__((aligned));
  char c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = i;
      b[i] = 10+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	15.396
GCC 4.0 - not optimized	15.002
GCC 4.1 - not optimized	14.586
ICC 8.1 - not optimized	14.617

GCC 4.0	6.568
GCC 4.1	6.335
ICC 8.1	4.827

GCC SIMD	1.281
ICC SIMD	1.741

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

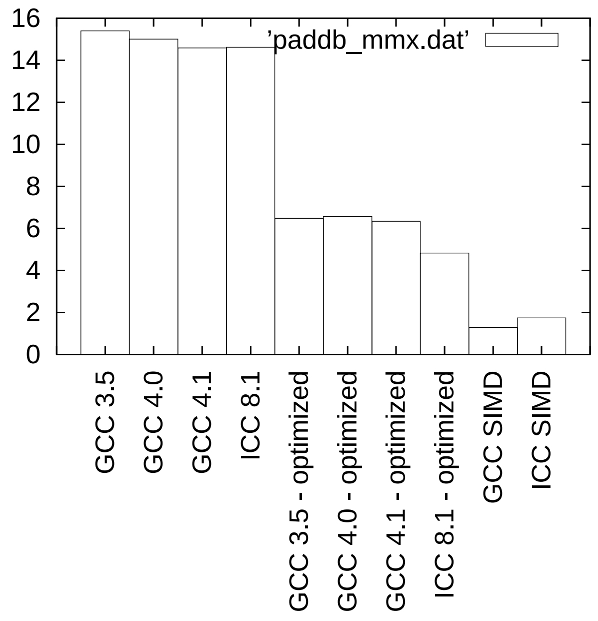

Figure 1: Benchmarks for paddb - MMX version

3.2 paddb - SSE2 (128 bits registers) version

3.2.1 C code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i;

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

3.2.2 GIMPLE code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i=0;
  loop_label::
  if(i >= 16)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.2.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[16], char b[16], char c[16])
{
  *(__m128i *) c = _mm_add_epi8(*(__m128i *) a,  *(__m128i *) b);
}
```

3.2.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`pushl %edi`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`xorl %edi, %edi`
`movl 8(%ebp), %eax`	`.L2:`	`pushl %esi`
`paddb (%eax), %xmm0`	`cmpl $15, -4(%ebp)`	`pushl %ebx`
`movl 16(%ebp), %eax`	`jle .L5`	`subl $12, %esp`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`movl 16(%ebp), %ebx`
`popl %ebp`	`.L5:`	`negl %ebx`
`ret`	`movl -4(%ebp), %eax`	`andl $15, %ebx`
	`movl 16(%ebp), %ecx`	`cmpl $0, %ebx`
	`addl %eax, %ecx`	`jbe .L20`
	`movl -4(%ebp), %eax`	`.p2align 4,,15`
	`movl 8(%ebp), %edx`	`.L11:`
	`addl %eax, %edx`	`movl 12(%ebp), %edx`
	`movl -4(%ebp), %eax`	`movzbl (%edx,%edi), %eax`
	`addl 12(%ebp), %eax`	`movl 8(%ebp), %edx`
	`movzbl (%eax), %eax`	`addb (%edx,%edi), %al`
	`addb (%edx), %al`	`movl 16(%ebp), %edx`
	`movb %al, (%ecx)`	`movb %al, (%edx,%edi)`
	`leal -4(%ebp), %eax`	`incl %edi`
	`incl (%eax)`	`cmpl %edi, %ebx`
	`jmp .L2`	`ja .L11`
	`.L1:`	`movl $16, -24(%ebp)`
	`leave`	`subl %edi, -24(%ebp)`
	`ret`	`cmpl $16, %ebx`
		`je .L13`
		`.L4:`
		`movl $16, -20(%ebp)`
		`subl %ebx, -20(%ebp)`
		`movl -20(%ebp), %esi`
		`shrl $4, %esi`
		`movl %esi, %eax`
		`sall $4, %eax`
		`cmpl $0, %eax`
		`movl %eax, -16(%ebp)`
		`jbe .L7`
		`movl 8(%ebp), %ecx`
		`movl 12(%ebp), %edx`
		`movl 16(%ebp), %eax`
		`addl %ebx, %ecx`
		`addl %ebx, %edx`

SIMD intrinsics	-nooptim	-O2 and vectorizer

```
 addl %ebx, %eax
 xorl %ebx, %ebx
 .p2align 4,,15
.L9:
 movdqu (%ecx), %xmm0
 movdqu (%edx), %xmm1
 incl %ebx
 paddb %xmm1, %xmm0
 addl $16, %ecx
 movdqa %xmm0, (%eax)
 addl $16, %edx
 addl $16, %eax
 cmpl %esi, %ebx
 jb .L9
 movl -16(%ebp), %eax
 subl %eax, -24(%ebp)
 addl %eax, %edi
 cmpl %eax, -20(%ebp)
 je .L13
.L7:
 movl 16(%ebp), %ebx
 xorl %esi, %esi
 movl 12(%ebp), %ecx
 movl 8(%ebp), %edx
 addl %edi, %ebx
 addl %edi, %ecx
 addl %edi, %edx
 .p2align 4,,15
.L12:
 movzbl (%ecx), %eax
 incl %esi
 incl %ecx
 addb (%edx), %al
 incl %edx
 movb %al, (%ebx)
 incl %ebx
 cmpl %esi, -24(%ebp)
 jne .L12
.L13:
 addl $12, %esp
 popl %ebx
 popl %esi
 popl %edi
 popl %ebp
 ret
.L20:
 movl $16, -24(%ebp)
 jmp .L4
```

3.2.5 Benchmark

```
char a[16] __attribute__((aligned));
char b[16] __attribute__((aligned));
char c[16] __attribute__((aligned));
int i;

for(i = 0; i<16; i++)
  {
      a[i] = i;
      b[i] = 10+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
  test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	31.091
GCC 4.0 - not optimized	28.385
GCC 4.1 - not optimized	28.989
ICC 8.1 - not optimized	28.009
GCC 4.0	5.672
GCC 4.1	5.831
ICC 8.1	3.368
GCC SIMD	1.217
ICC SIMD	2.45

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `paddb`

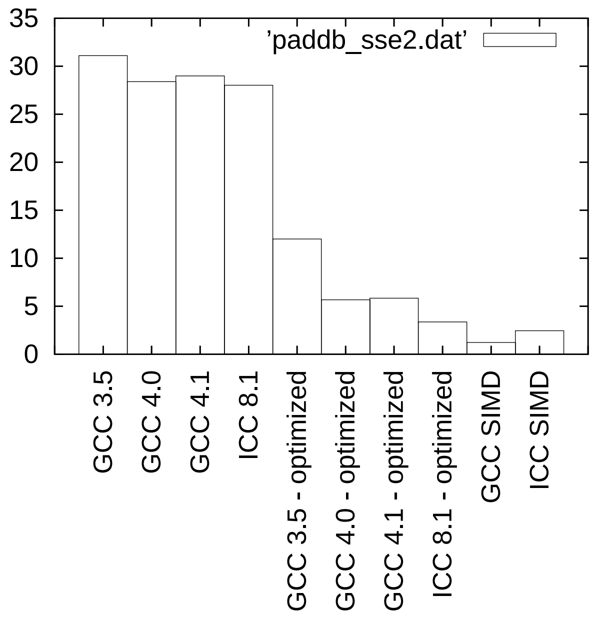

Figure 2: Benchmarks for paddb - SSE2 version

3.3 paddw - MMX (64 bits registers) version

3.3.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      c[i] = a[i] + b[i];
    }
}
```

3.3.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i=0;
  loop_label::
  if(i >= 4)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.3.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_add_pi16(*(__m64 *) a, *(__m64 *) b);
}
```

3.3.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`pushl %edi`	`pushl %edi`
`movq (%eax), %mm0`	`pushl %esi`	`movl 12(%ebp), %edi`
`movl 8(%ebp), %eax`	`pushl %ebx`	`pushl %esi`
`paddw (%eax), %mm0`	`subl $4, %esp`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`movl $0, -16(%ebp)`	`pushl %ebx`
`movq %mm0, (%eax)`	`.L2:`	`movl $1, %ebx`
`popl %ebp`	`cmpl $3, -16(%ebp)`	`.p2align 4,,15`
`ret`	`jle .L5`	`.L2:`
	`jmp .L1`	`movl 8(%ebp), %ecx`
	`.L5:`	`leal (%ebx,%ebx), %eax`
	`movl -16(%ebp), %eax`	`incl %ebx`
	`leal (%eax,%eax), %esi`	`movzwl -2(%eax,%ecx), %edx`
	`movl 16(%ebp), %edi`	`movzwl -2(%eax,%edi), %ecx`
	`movl -16(%ebp), %eax`	`addl %ecx, %edx`
	`leal (%eax,%eax), %ebx`	`cmpl $5, %ebx`
	`movl 8(%ebp), %edx`	`movw %dx, -2(%eax,%esi)`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`movl -16(%ebp), %eax`	`jne .L2`
	`leal (%eax,%eax), %ecx`	`popl %ebx`
	`movl 12(%ebp), %eax`	`popl %esi`
	`movzwl (%edx,%ebx), %edx`	`popl %edi`
	`movzwl (%eax,%ecx), %eax`	`popl %ebp`
	`leal (%eax,%edx), %eax`	`ret`
	`movw %ax, (%edi,%esi)`	
	`leal -16(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`addl $4, %esp`	
	`popl %ebx`	
	`popl %esi`	
	`popl %edi`	
	`popl %ebp`	
	`ret`	

3.3.5 Benchmark

```
  short int a[4] __attribute__((aligned));
  short int b[4] __attribute__((aligned));
  short int c[4] __attribute__((aligned));

  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = 140 + i;
      b[i] = 140 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	5.861
GCC 4.0 - not optimized	5.828
GCC 4.1 - not optimized	5.926
ICC 8.1 - not optimized	5.632

GCC 4.0	2.852
GCC 4.1	3.496
ICC 8.1	2.216

GCC SIMD	0.88
ICC SIMD	0.81

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

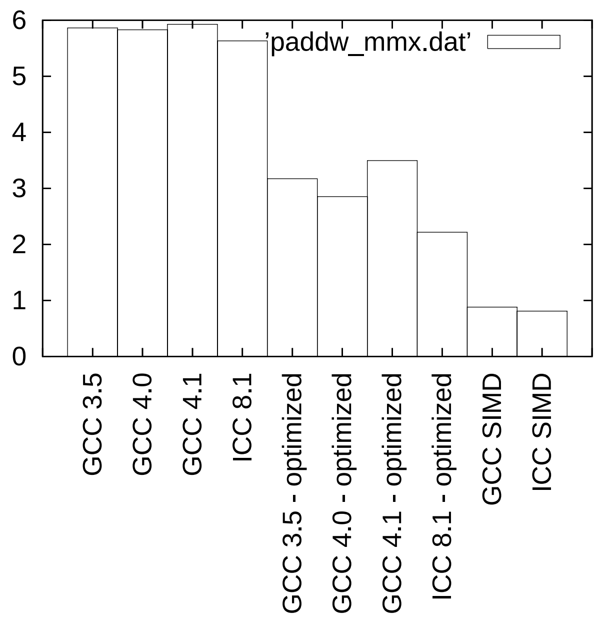

Figure 3: Benchmarks for paddw - MMX version

3.4 paddw - SSE2 (128 bits registers) version

3.4.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      c[i] = a[i] + b[i];
    }
}
```

3.4.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i=0;
  loop_label::
  if(i >= 8)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.4.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_add_epi16(*(__m128i *) a, *(__m128i *) b);
}
```

3.4.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`pushl %edi`	`pushl %edi`
`movdqa (%eax), %xmm0`	`pushl %esi`	`pushl %esi`
`movl 8(%ebp), %eax`	`pushl %ebx`	`xorl %esi, %esi`
`paddw (%eax), %xmm0`	`subl $4, %esp`	`pushl %ebx`
`movl 16(%ebp), %eax`	`movl $0, -16(%ebp)`	`subl $12, %esp`
`movdqa %xmm0, (%eax)`	`.L2:`	`movl 16(%ebp), %ebx`
`popl %ebp`	`cmpl $7, -16(%ebp)`	`andl $15, %ebx`
`ret`	`jle .L5`	`shrl %ebx`
	`jmp .L1`	`negl %ebx`
	`.L5:`	`andl $7, %ebx`
	`movl -16(%ebp), %eax`	`cmpl $0, %ebx`
	`leal (%eax,%eax), %esi`	`jbe .L20`
	`movl 16(%ebp), %edi`	`.p2align 4,,15`
	`movl -16(%ebp), %eax`	`.L11:`
	`leal (%eax,%eax), %ebx`	`movl 8(%ebp), %ecx`
	`movl 8(%ebp), %edx`	`leal (%esi,%esi), %eax`
	`movl -16(%ebp), %eax`	`incl %esi`
	`leal (%eax,%eax), %ecx`	`movl 12(%ebp), %edi`
	`movl 12(%ebp), %eax`	`movzwl (%ecx,%eax), %edx`
	`movzwl (%edx,%ebx), %edx`	`movzwl (%edi,%eax), %ecx`
	`movzwl (%eax,%ecx), %eax`	`addl %ecx, %edx`
	`leal (%eax,%edx), %eax`	`movl 16(%ebp), %ecx`
	`movw %ax, (%edi,%esi)`	`cmpl %esi, %ebx`
	`leal -16(%ebp), %eax`	`movw %dx, (%ecx,%eax)`
	`incl (%eax)`	`ja .L11`
	`jmp .L2`	`movl $8, -24(%ebp)`
	`.L1:`	`subl %esi, -24(%ebp)`
	`addl $4, %esp`	`cmpl $8, %ebx`
	`popl %ebx`	`je .L13`
	`popl %esi`	`.L4:`
	`popl %edi`	`movl $8, -20(%ebp)`
	`popl %ebp`	`subl %ebx, -20(%ebp)`
	`ret`	`movl -20(%ebp), %edi`
		`shrl $3, %edi`
		`leal 0(,%edi,8), %eax`
		`cmpl $0, %eax`
		`movl %eax, -16(%ebp)`
		`jbe .L7`
		`leal (%ebx,%ebx), %eax`
		`movl 8(%ebp), %ecx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
		movl 16(%ebp), %ebx movl 12(%ebp), %edx addl %eax, %ecx addl %eax, %edx addl %ebx, %eax xorl %ebx, %ebx .p2align 4,,15 .L9: movdqu (%ecx), %xmm0 movdqu (%edx), %xmm1 incl %ebx paddw %xmm1, %xmm0 addl $16, %ecx movdqa %xmm0, (%eax) addl $16, %edx addl $16, %eax cmpl %edi, %ebx jb .L9 movl -16(%ebp), %edi subl %edi, -24(%ebp) addl %edi, %esi cmpl %edi, -20(%ebp) je .L13 .L7: leal (%esi,%esi), %eax movl 12(%ebp), %ebx xorl %edi, %edi movl 8(%ebp), %esi movl 16(%ebp), %ecx addl %eax, %ebx addl %eax, %esi addl %eax, %ecx .p2align 4,,15 .L12: movzwl (%esi), %eax incl %edi addl $2, %esi movzwl (%ebx), %edx addl $2, %ebx addl %edx, %eax movw %ax, (%ecx) addl $2, %ecx cmpl %edi, -24(%ebp) jne .L12 .L13: addl $12, %esp popl %ebx popl %esi popl %edi popl %ebp ret .L20:

SIMD intrinsics	-nooptim	-O2 and vectorizer

		`movl $8, -24(%ebp)`
		`jmp .L4`

3.4.5 Benchmark

```
 short int a[8] __attribute__((aligned));
 short int b[8] __attribute__((aligned));
 short int c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 140 + i;
     b[i] = 140 + 2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	12.732
GCC 4.0 - not optimized	10.233
GCC 4.1 - not optimized	11.03
ICC 8.1 - not optimized	11.243
GCC 4.0	4.64
GCC 4.1	5.045
ICC 8.1	2.597
GCC SIMD	1.18
ICC SIMD	1.79
GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `paddw`

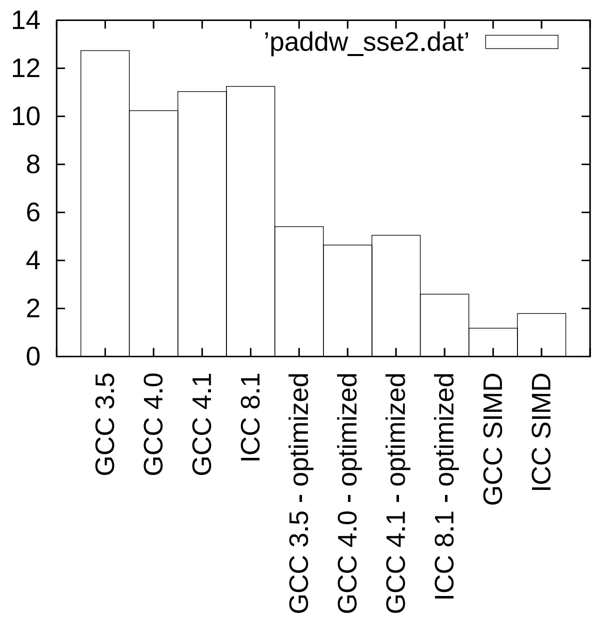

Figure 4: Benchmarks for paddw - SSE2 version

3.5 paddd - MMX (64 bits registers) version

3.5.1 C code

```
void test_loop_c(int a[2], int b[2], int c[2])
{
  int i;

  for(i=0; i<2; i++)
    {
      c[i] = a[i] + b[i];
    }
}
```

3.5.2 GIMPLE code

```
void test_loop_c(int a[2], int b[2], int c[2])
{
  int i=0;
  loop_label::
  if(i >= 2)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.5.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(int a[2], int b[2], int c[2])
{
   *(__m64 *) c = _mm_add_pi32(*(__m64 *) a, *(__m64 *) b);
}
```

3.5.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %edx`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`pushl %esi`
`movl 8(%ebp), %eax`	`.L2:`	`movl 16(%ebp), %ecx`
`paddb (%eax), %mm0`	`cmpl $7, -4(%ebp)`	`pushl %ebx`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 8(%ebp), %esi`
`movq %mm0, (%eax)`	`jmp .L1`	`movl 12(%ebp), %ebx`
`popl %ebp`	`.L5:`	`.p2align 4,,15`
`ret`	`movl -4(%ebp), %eax`	`.L2:`
	`movl 16(%ebp), %ecx`	`movzbl -1(%ebx,%edx), %eax`
	`addl %eax, %ecx`	`addb -1(%esi,%edx), %al`
	`movl -4(%ebp), %eax`	`movb %al, -1(%ecx,%edx)`
	`movl 8(%ebp), %edx`	`incl %edx`
	`addl %eax, %edx`	`cmpl $9, %edx`
	`movl -4(%ebp), %eax`	`jne .L2`
	`addl 12(%ebp), %eax`	`popl %ebx`
	`movzbl (%eax), %eax`	`popl %esi`
	`addb (%edx), %al`	`popl %ebp`
	`movb %al, (%ecx)`	`ret`
	`leal -4(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

3.5.5 Benchmark

```
  int a[2] __attribute__((aligned));
  int b[2] __attribute__((aligned));
  int c[2] __attribute__((aligned));
  int i;

  for(i = 0; i<2; i++)
    {
      a[i] = 140000 + i;
      b[i] = 140000 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	6.185
GCC 4.0 - not optimized	5.006
GCC 4.1 - not optimized	4.936
ICC 8.1 - not optimized	4.735

GCC 4.0	2.86
GCC 4.1	3.162
ICC 8.1	2.038

GCC SIMD	1.413
ICC SIMD	1.867

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

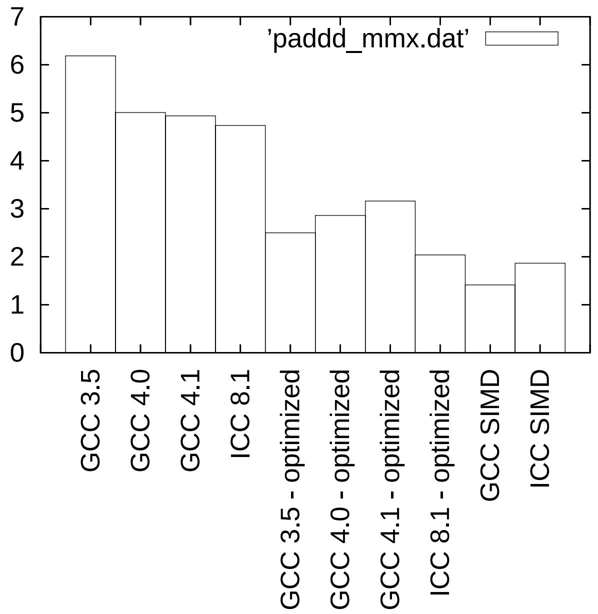

Figure 5: Benchmarks for paddd - MMX version

3.6 paddd - SSE2 (128 bits registers) version

3.6.1 C code

```
void test_loop_c(int a[4], int b[4], int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      c[i] = a[i] + b[i];
    }
}
```

3.6.2 GIMPLE code

```
void test_loop_c(int a[4], int b[4], int c[4])
```

```
{
  int i=0;
  loop_label::
  if(i >= 4)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.6.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(int a[4], int b[4], int c[4])
{
   *(__m128i *) c = _mm_add_epi32(*(__m128i *) a, *(__m128i *) b);
}
```

3.6.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`pushl %edi`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`xorl %edi, %edi`
`movl 8(%ebp), %eax`	`.L2:`	`pushl %esi`
`paddb (%eax), %xmm0`	`cmpl $15, -4(%ebp)`	`pushl %ebx`
`movl 16(%ebp), %eax`	`jle .L5`	`subl $12, %esp`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`movl 16(%ebp), %ebx`
`popl %ebp`	`.L5:`	`negl %ebx`
`ret`	`movl -4(%ebp), %eax`	`andl $15, %ebx`
	`movl 16(%ebp), %ecx`	`cmpl $0, %ebx`
	`addl %eax, %ecx`	`jbe .L20`
	`movl -4(%ebp), %eax`	`.p2align 4,,15`
	`movl 8(%ebp), %edx`	`.L11:`
	`addl %eax, %edx`	`movl 12(%ebp), %edx`
	`movl -4(%ebp), %eax`	`movzbl (%edx,%edi), %eax`
	`addl 12(%ebp), %eax`	`movl 8(%ebp), %edx`
	`movzbl (%eax), %eax`	`addb (%edx,%edi), %al`
	`addb (%edx), %al`	`movl 16(%ebp), %edx`
	`movb %al, (%ecx)`	`movb %al, (%edx,%edi)`
	`leal -4(%ebp), %eax`	`incl %edi`
	`incl (%eax)`	`cmpl %edi, %ebx`
	`jmp .L2`	`ja .L11`
	`.L1:`	`movl $16, -24(%ebp)`
	`leave`	`subl %edi, -24(%ebp)`
	`ret`	`cmpl $16, %ebx`
		`je .L13`
		`.L4:`

SIMD intrinsics	-nooptim	-O2 and vectorizer
		`movl $16, -20(%ebp)`
		`subl %ebx, -20(%ebp)`
		`movl -20(%ebp), %esi`
		`shrl $4, %esi`
		`movl %esi, %eax`
		`sall $4, %eax`
		`cmpl $0, %eax`
		`movl %eax, -16(%ebp)`
		`jbe .L7`
		`movl 8(%ebp), %ecx`
		`movl 12(%ebp), %edx`
		`movl 16(%ebp), %eax`
		`addl %ebx, %ecx`
		`addl %ebx, %edx`
		`addl %ebx, %eax`
		`xorl %ebx, %ebx`
		`.p2align 4,,15`
		`.L9:`
		`movdqu (%ecx), %xmm0`
		`movdqu (%edx), %xmm1`
		`incl %ebx`
		`paddb %xmm1, %xmm0`
		`addl $16, %ecx`
		`movdqa %xmm0, (%eax)`
		`addl $16, %edx`
		`addl $16, %eax`
		`cmpl %esi, %ebx`
		`jb .L9`
		`movl -16(%ebp), %eax`
		`subl %eax, -24(%ebp)`
		`addl %eax, %edi`
		`cmpl %eax, -20(%ebp)`
		`je .L13`
		`.L7:`
		`movl 16(%ebp), %ebx`
		`xorl %esi, %esi`
		`movl 12(%ebp), %ecx`
		`movl 8(%ebp), %edx`
		`addl %edi, %ebx`
		`addl %edi, %ecx`
		`addl %edi, %edx`
		`.p2align 4,,15`
		`.L12:`
		`movzbl (%ecx), %eax`
		`incl %esi`
		`incl %ecx`
		`addb (%edx), %al`
		`incl %edx`
		`movb %al, (%ebx)`
		`incl %ebx`
		`cmpl %esi, -24(%ebp)`
		`jne .L12`

SIMD intrinsics	-nooptim	-O2 and vectorizer
		`.L13:`
		` addl $12, %esp`
		` popl %ebx`
		` popl %esi`
		` popl %edi`
		` popl %ebp`
		` ret`
		`.L20:`
		` movl $16, -24(%ebp)`
		` jmp .L4`

3.6.5 Benchmark

```
  int a[4] __attribute__((aligned));
  int b[4] __attribute__((aligned));
  int c[4] __attribute__((aligned));
  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = 140 + i;
      b[i] = 140 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	9.24
GCC 4.0 - not optimized	9.1
GCC 4.1 - not optimized	9.078
ICC 8.1 - not optimized	7.881

GCC 4.0	6.187
GCC 4.1	6.021
ICC 8.1	3.312

GCC SIMD	1.318
ICC SIMD	2.186

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `paddd`

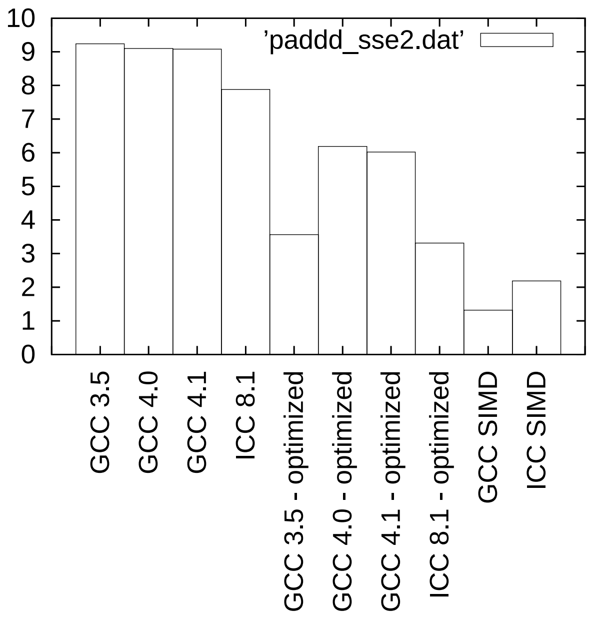

Figure 6: Benchmarks for paddd - SSE2 version

3.7 paddq - MMX (64 bits registers) version

3.7.1 C code

```
void test_loop_c(long long int a[1], long long int b[1], long long int c[1])
{
  c[0] = a[0] + b[0];
}
```

3.7.2 GIMPLE code

```
void test_loop_c(long long int a[1], long long int b[1], long long int c[1])
{
  t1 = a[0];
  t2 = b[0];
  t3 = t1 + t2;
  c[0] = t3;
}
```

3.7.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(long long int a[1], long long int b[1], long long int c[1])
{
   *(__m64 *) c = _mm_add_si64(*(__m64 *) a, *(__m64 *) b);
}
```

3.7.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %edx`

```
SIMD intrinsics              -nooptim                     -O2 and vectorizer
 movl 12(%ebp), %eax          subl $4, %esp                movl %esp, %ebp
 movq (%eax), %mm0            movl $0, -4(%ebp)            pushl %esi
 movl 8(%ebp), %eax          .L2:                          movl 16(%ebp), %ecx
 paddb (%eax), %mm0           cmpl $7, -4(%ebp)            pushl %ebx
 movl 16(%ebp), %eax          jle .L5                      movl 8(%ebp), %esi
 movq %mm0, (%eax)            jmp .L1                      movl 12(%ebp), %ebx
 popl %ebp                   .L5:                          .p2align 4,,15
 ret                          movl -4(%ebp), %eax         .L2:
                              movl 16(%ebp), %ecx          movzbl -1(%ebx,%edx), %eax
                              addl %eax, %ecx              addb -1(%esi,%edx), %al
                              movl -4(%ebp), %eax          movb %al, -1(%ecx,%edx)
                              movl 8(%ebp), %edx           incl %edx
                              addl %eax, %edx              cmpl $9, %edx
                              movl -4(%ebp), %eax          jne .L2
                              addl 12(%ebp), %eax          popl %ebx
                              movzbl (%eax), %eax          popl %esi
                              addb (%edx), %al             popl %ebp
                              movb %al, (%ecx)             ret
                              leal -4(%ebp), %eax
                              incl (%eax)
                              jmp .L2
                             .L1:
                              leave
                              ret
```

3.7.5 Benchmark

```
 long long int a[1] __attribute__((aligned));
 long long int b[1] __attribute__((aligned));
 long long int c[1] __attribute__((aligned));
 int i;

 for(i = 0; i<1; i++)
   {
     a[i] = 140000 + i;
     b[i] = 140000 + 2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	2.232
GCC 4.0 - not optimized	2.092
GCC 4.1 - not optimized	2.329
ICC 8.1 - not optimized	2.017
GCC 4.0	2.161
GCC 4.1	2.071
ICC 8.1	2.102
GCC SIMD	1.317
ICC SIMD	1.651

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

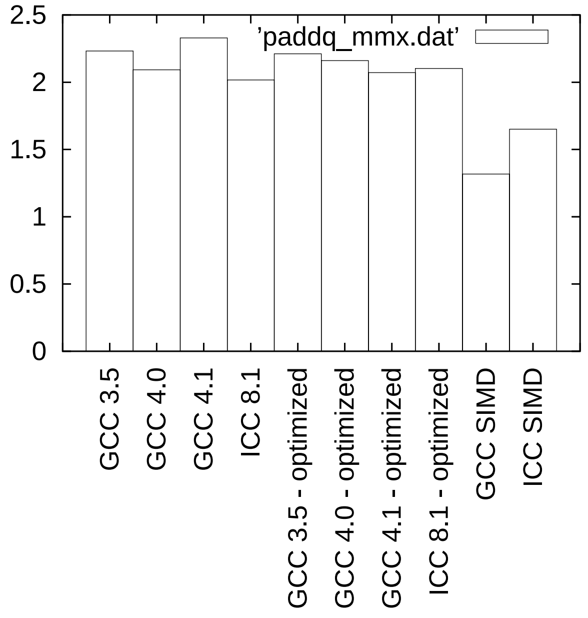

Figure 7: Benchmarks for paddq - MMX version

3.8 paddq - SSE2 (128 bits registers) version

3.8.1 C code

```
void test_loop_c(long long int a[2], long long int b[2], long long int c[2])
{
  int i;

  for(i=0; i<2; i++)
  {
    c[i] = a[i] + b[i];
  }
}
```

3.8.2 GIMPLE code

```
void test_loop_c(long long int a[2], long long int b[2], long long int c[2])
{
  int i=0;
  loop_label::
  if(i >= 2)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;
  break_label:;
}
```

3.8.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(long long int a[2], long long int b[2], long long int c[2])
{
   *(__m128i *) c = _mm_add_epi64(*(__m128i *) a, *(__m128i *) b);
}
```

3.8.4 Assembly code

The assembly codes that are generated are the following:

```
SIMD intrinsics               -nooptim                      -O2 and vectorizer
 pushl %ebp                    pushl %ebp                    pushl %ebp
 movl %esp, %ebp               movl %esp, %ebp               movl %esp, %ebp
 movl 8(%ebp), %eax            pushl %edi                    pushl %edi
 movdqa (%eax), %xmm0          pushl %esi                    pushl %esi
 movl 12(%ebp), %eax           pushl %ebx                    xorl %esi, %esi
 paddq (%eax), %xmm0           subl $4, %esp                 pushl %ebx
 movl 16(%ebp), %eax           movl $0, -16(%ebp)            subl $12, %esp
 movdqa %xmm0, (%eax)          movl $0, -16(%ebp)            movl 16(%ebp), %ebx
 popl %ebp                    .L2:                           andl $15, %ebx
 ret                           cmpl $1, -16(%ebp)            shrl $3, %ebx
                               jle .L5                       cmpl $0, %ebx
                               jmp .L1                       jbe .L20
                              .L5:                           .p2align 4,,15
                               movl -16(%ebp), %eax         .L11:
                               leal 0(,%eax,8), %esi         movl 12(%ebp), %edi
                               movl 16(%ebp), %edi           leal 0(,%esi,8), %ecx
                               movl -16(%ebp), %eax          movl (%edi,%ecx), %eax
                               leal 0(,%eax,8), %ebx         movl 4(%edi,%ecx), %edx
                               movl 8(%ebp), %ecx            movl 8(%ebp), %edi
                               movl -16(%ebp), %eax          addl (%edi,%ecx), %eax
                               leal 0(,%eax,8), %edx         adcl 4(%edi,%ecx), %edx
                               movl 12(%ebp), %eax           incl %esi
                               leal (%eax,%edx), %edx        movl 16(%ebp), %edi
                               movl (%edx), %eax             cmpl %esi, %ebx
                               movl 4(%edx), %edx            movl %eax, (%edi,%ecx)
                               addl (%ecx,%ebx), %eax        movl %edx, 4(%edi,%ecx)
                               adcl 4(%ecx,%ebx), %edx       ja .L11
                               movl %eax, (%edi,%esi)        movl $2, -24(%ebp)
                               movl %edx, 4(%edi,%esi)       subl %esi, -24(%ebp)
                               leal -16(%ebp), %eax          cmpl $2, %ebx
                               incl (%eax)                   je .L13
                               jmp .L2                      .L4:
                              .L1:                           movl $2, -20(%ebp)
                               addl $4, %esp                 subl %ebx, -20(%ebp)
                               popl %ebx                     movl -20(%ebp), %edi
                               popl %esi                     shrl %edi
                               popl %edi                     leal (%edi,%edi), %eax
                               popl %ebp                     cmpl $0, %eax
                               ret                           movl %eax, -16(%ebp)
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

```
 jbe .L7
 leal 0(,%ebx,8), %eax
 movl 8(%ebp), %ecx
 movl 16(%ebp), %ebx
 movl 12(%ebp), %edx
 addl %eax, %ecx
 addl %eax, %edx
 addl %ebx, %eax
 xorl %ebx, %ebx
 .p2align 4,,15
.L9:
 movdqu (%ecx), %xmm0
 movdqu (%edx), %xmm1
 incl %ebx
 paddq %xmm1, %xmm0
 addl $16, %ecx
 movdqa %xmm0, (%eax)
 addl $16, %edx
 addl $16, %eax
 cmpl %edi, %ebx
 jb .L9
 movl -16(%ebp), %eax
 subl %eax, -24(%ebp)
 addl %eax, %esi
 cmpl %eax, -20(%ebp)
 je .L13
.L7:
 leal 0(,%esi,8), %eax
 movl 12(%ebp), %ebx
 xorl %edi, %edi
 movl 8(%ebp), %esi
 movl 16(%ebp), %ecx
 addl %eax, %ebx
 addl %eax, %esi
 addl %eax, %ecx
 .p2align 4,,15
.L12:
 movl (%ebx), %eax
 addl (%esi), %eax
 movl 4(%ebx), %edx
 adcl 4(%esi), %edx
 incl %edi
 movl %eax, (%ecx)
 addl $8, %esi
 addl $8, %ebx
 movl %edx, 4(%ecx)
 addl $8, %ecx
 cmpl %edi, -24(%ebp)
 jne .L12
.L13:
 addl $12, %esp
 popl %ebx
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

		`popl %esi`
		`popl %edi`
		`popl %ebp`
		`ret`
		`.L20:`
		`movl $2, -24(%ebp)`
		`jmp .L4`

3.8.5 Benchmark

```
 long long int a[2] __attribute__((aligned));
 long long int b[2] __attribute__((aligned));
 long long int c[2] __attribute__((aligned));
 int i;

 for(i = 0; i<2; i++)
   {
     a[i] = 140000 + i;
     b[i] = 140000 + 2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	8.547
GCC 4.0 - not optimized	6.384
GCC 4.1 - not optimized	6.613
ICC 8.1 - not optimized	3.783

GCC 4.0	6.198
GCC 4.1	5.455
ICC 8.1	3.242

GCC SIMD	1.552
ICC SIMD	1.458

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	Unrolling

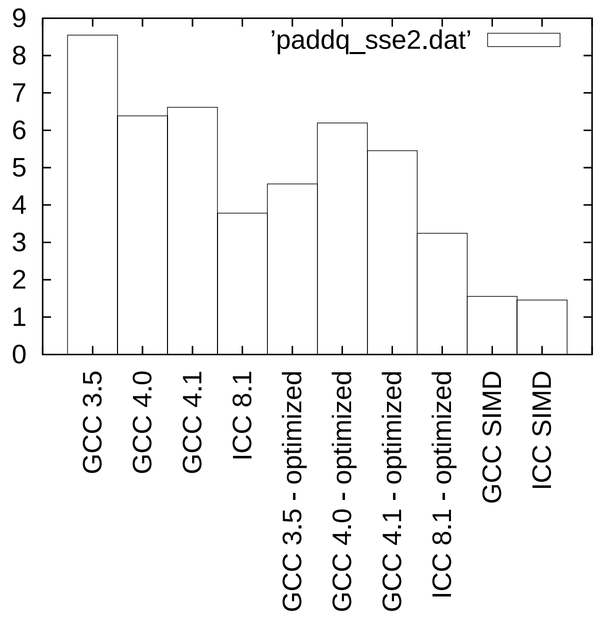

Figure 8: Benchmarks for paddq - SSE2 version

3.9 psubb - MMX (64 bits registers) version

3.9.1 C code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i;

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

3.9.2 GIMPLE code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i=0;
  loop_label::
  if(i >= 8)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 - t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.9.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[8], char b[8], char c[8])
{
  *(__m64 *) c = _mm_sub_pi8(*(__m64*) a,  *(__m64*) b);
}
```

3.9.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics               -nooptim                      -O2 and vectorizer
---------------------------------------------------------------------------------------------
  pushl %ebp                    pushl %ebp                    pushl %ebp
  movl %esp, %ebp               movl %esp, %ebp               movl $1, %edx
  movl 8(%ebp), %eax            pushl %ebx                    movl %esp, %ebp
  movq (%eax), %mm0             subl $4, %esp                 pushl %esi
  movl 12(%ebp), %eax           movl $0, -8(%ebp)             movl 16(%ebp), %ecx
  psubb (%eax), %mm0           .L2:                           pushl %ebx
  movl 16(%ebp), %eax           cmpl $7, -8(%ebp)             movl 8(%ebp), %esi
  movq %mm0, (%eax)             jle .L5                       movl 12(%ebp), %ebx
  popl %ebp                     jmp .L1                       .p2align 4,,15
  ret                          .L5:                          .L2:
                                movl -8(%ebp), %eax           movzbl -1(%esi,%edx), %eax
                                movl 16(%ebp), %ebx           subb -1(%ebx,%edx), %al
                                addl %eax, %ebx               movb %al, -1(%ecx,%edx)
                                movl -8(%ebp), %eax           incl %edx
                                movl 8(%ebp), %ecx            cmpl $9, %edx
                                addl %eax, %ecx               jne .L2
                                movl -8(%ebp), %eax           popl %ebx
                                addl 12(%ebp), %eax           popl %esi
                                movzbl (%eax), %edx           popl %ebp
                                movzbl (%ecx), %eax           ret
                                subb %dl, %al
                                movb %al, (%ebx)
                                leal -8(%ebp), %eax
                                incl (%eax)
                                jmp .L2
                               .L1:
                                addl $4, %esp
                                popl %ebx
                                popl %ebp
                                ret
```

3.9.5 Benchmark

```
  char a[8] __attribute__((aligned));
  char b[8] __attribute__((aligned));
  char c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = i;
      b[i] = 10+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	14.765
GCC 4.0 - not optimized	14.273
GCC 4.1 - not optimized	14.519
ICC 8.1 - not optimized	14.373

GCC 4.0	6.507
GCC 4.1	6.219
ICC 8.1	4.385

GCC SIMD	1.412
ICC SIMD	1.56

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

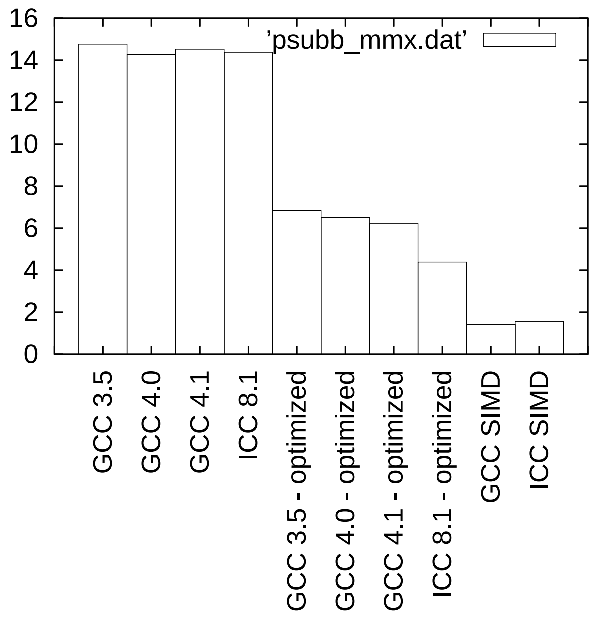

Figure 9: Benchmarks for psubb - MMX version

3.10 psubb - SSE2 (128 bits registers) version

3.10.1 C code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i;

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

3.10.2 GIMPLE code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i=0;
  loop_label::
  if(i >= 16)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 - t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.10.3 Assembly code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[16], char b[16], char c[16])
{
  *(__m128i *) c = _mm_sub_epi8(*(__m128i *) a,  *(__m128i *) b);
}
```

3.10.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl %esp, %ebp
movl 12(%ebp), %eax	subl $4, %esp	pushl %edi
movdqa (%eax), %xmm0	movl $0, -4(%ebp)	xorl %edi, %edi
movl 8(%ebp), %eax	.L2:	pushl %esi
paddb (%eax), %xmm0	cmpl $15, -4(%ebp)	pushl %ebx
movl 16(%ebp), %eax	jle .L5	subl $12, %esp
movdqa %xmm0, (%eax)	jmp .L1	movl 16(%ebp), %ebx
popl %ebp	.L5:	negl %ebx
ret	movl -4(%ebp), %eax	andl $15, %ebx
	movl 16(%ebp), %ecx	cmpl $0, %ebx
	addl %eax, %ecx	jbe .L20
	movl -4(%ebp), %eax	.p2align 4,,15
	movl 8(%ebp), %edx	.L11:
	addl %eax, %edx	movl 12(%ebp), %edx
	movl -4(%ebp), %eax	movzbl (%edx,%edi), %eax
	addl 12(%ebp), %eax	movl 8(%ebp), %edx
	movzbl (%eax), %eax	addb (%edx,%edi), %al
	addb (%edx), %al	movl 16(%ebp), %edx
	movb %al, (%ecx)	movb %al, (%edx,%edi)
	leal -4(%ebp), %eax	incl %edi
	incl (%eax)	cmpl %edi, %ebx
	jmp .L2	ja .L11
	.L1:	movl $16, -24(%ebp)
	leave	subl %edi, -24(%ebp)
	ret	cmpl $16, %ebx

SIMD intrinsics	-nooptim	-O2 and vectorizer

```
 je .L13
.L4:
 movl $16, -20(%ebp)
 subl %ebx, -20(%ebp)
 movl -20(%ebp), %esi
 shrl $4, %esi
 movl %esi, %eax
 sall $4, %eax
 cmpl $0, %eax
 movl %eax, -16(%ebp)
 jbe .L7
 movl 8(%ebp), %ecx
 movl 12(%ebp), %edx
 movl 16(%ebp), %eax
 addl %ebx, %ecx
 addl %ebx, %edx
 addl %ebx, %eax
 xorl %ebx, %ebx
 .p2align 4,,15
.L9:
 movdqu (%ecx), %xmm0
 movdqu (%edx), %xmm1
 incl %ebx
 paddb %xmm1, %xmm0
 addl $16, %ecx
 movdqa %xmm0, (%eax)
 addl $16, %edx
 addl $16, %eax
 cmpl %esi, %ebx
 jb .L9
 movl -16(%ebp), %eax
 subl %eax, -24(%ebp)
 addl %eax, %edi
 cmpl %eax, -20(%ebp)
 je .L13
.L7:
 movl 16(%ebp), %ebx
 xorl %esi, %esi
 movl 12(%ebp), %ecx
 movl 8(%ebp), %edx
 addl %edi, %ebx
 addl %edi, %ecx
 addl %edi, %edx
 .p2align 4,,15
.L12:
 movzbl (%ecx), %eax
 incl %esi
 incl %ecx
 addb (%edx), %al
 incl %edx
 movb %al, (%ebx)
 incl %ebx
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

		`cmpl %esi, -24(%ebp)`
		`jne .L12`
		`.L13:`
		`addl $12, %esp`
		`popl %ebx`
		`popl %esi`
		`popl %edi`
		`popl %ebp`
		`ret`
		`.L20:`
		`movl $16, -24(%ebp)`
		`jmp .L4`

3.10.5 Benchmark

```
 char a[16] __attribute__((aligned));
 char b[16] __attribute__((aligned));
 char c[16] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = i;
     b[i] = 10+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	27.225
GCC 4.0 - not optimized	27.064
GCC 4.1 - not optimized	28.364
ICC 8.1 - not optimized	24.105

GCC 4.0	5.73
GCC 4.1	4.593
ICC 8.1	2.908

GCC SIMD	1.416
ICC SIMD	2.172

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `psubb`

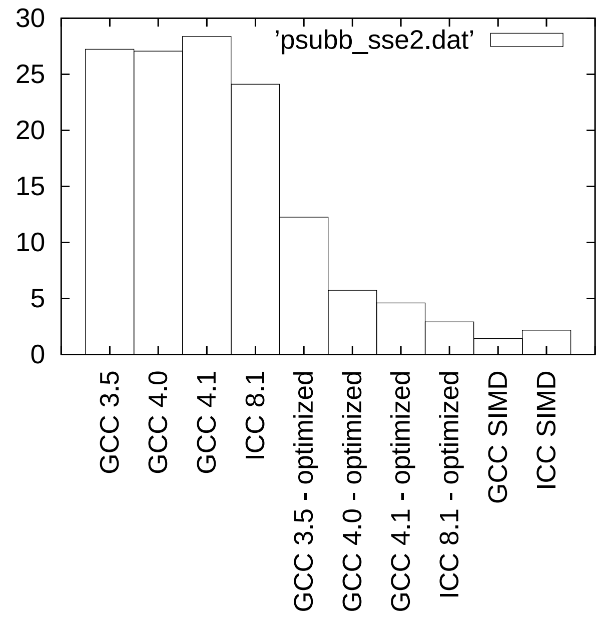

Figure 10: Benchmarks for psubb - SSE2 version

3.11 psubw - MMX (64 bits registers) version

3.11.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;

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

3.11.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i=0;
  loop_label::
  if(i >= 4)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 - t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.11.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_add_pi16(*(__m64 *) a, *(__m64 *) b);
}
```

3.11.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                      -O2 and vectorizer
--------------------------------------------------------------------------------------
 pushl %ebp                    pushl %ebp                    pushl %ebp
 movl %esp, %ebp               movl %esp, %ebp               movl %esp, %ebp
 movl 8(%ebp), %eax            pushl %edi                    pushl %edi
 movq (%eax), %mm0             pushl %esi                    movl 12(%ebp), %edi
 movl 12(%ebp), %eax           pushl %ebx                    pushl %esi
 psubw (%eax), %mm0            subl $4, %esp                 movl 16(%ebp), %esi
 movl 16(%ebp), %eax           movl $0, -16(%ebp)            pushl %ebx
 movq %mm0, (%eax)            .L2:                           movl $1, %ebx
 popl %ebp                     cmpl $3, -16(%ebp)            .p2align 4,,15
 ret                           jle .L5                      .L2:
                               jmp .L1                       movl 8(%ebp), %ecx
                              .L5:                           leal (%ebx,%ebx), %eax
                               movl -16(%ebp), %eax          incl %ebx
                               leal (%eax,%eax), %esi        movzwl -2(%eax,%ecx), %edx
                               movl 16(%ebp), %edi           movzwl -2(%eax,%edi), %ecx
                               movl -16(%ebp), %eax          subl %ecx, %edx
                               leal (%eax,%eax), %ebx        cmpl $5, %ebx
                               movl 8(%ebp), %edx            movw %dx, -2(%eax,%esi)
                               movl -16(%ebp), %eax          jne .L2
                               leal (%eax,%eax), %ecx        popl %ebx
                               movl 12(%ebp), %eax           popl %esi
                               movzwl (%edx,%ebx), %edx      popl %edi
                               movzwl (%eax,%ecx), %eax      popl %ebp
                               movl %edx, %ecx               ret
                               subl %eax, %ecx
                               movl %ecx, %eax
                               movw %ax, (%edi,%esi)
                               leal -16(%ebp), %eax
                               incl (%eax)
                               jmp .L2
                              .L1:
                               addl $4, %esp
                               popl %ebx
                               popl %esi
                               popl %edi
                               popl %ebp
                               ret
```

3.11.5 Benchmark

```
  short int a[4] __attribute__((aligned));
  short int b[4] __attribute__((aligned));
  short int c[4] __attribute__((aligned));

  int i;

  for(i = 0; i<4; i++)
```

```
    {
      a[i] = 140 + i;
      b[i] = 140 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	10.609
GCC 4.0 - not optimized	11.142
GCC 4.1 - not optimized	12.639
ICC 8.1 - not optimized	11.021

GCC 4.0	5.205
GCC 4.1	5.402
ICC 8.1	4.415

GCC SIMD	1.739
ICC SIMD	2.165

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

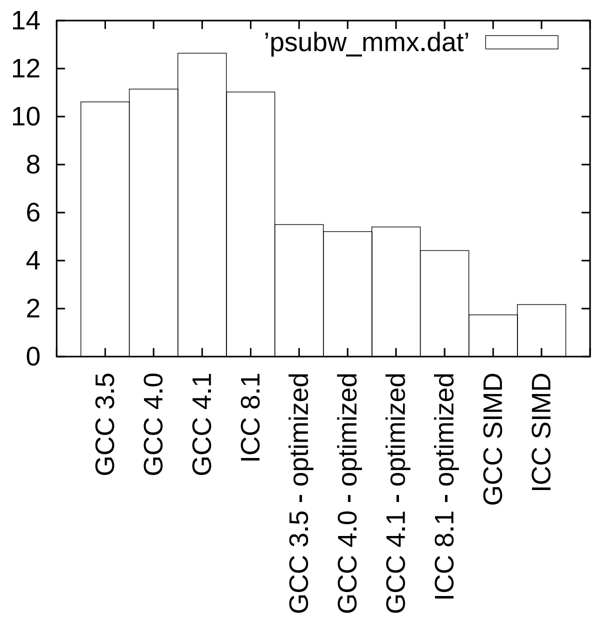

Figure 11: Benchmarks for psubw - MMX version

3.12 psubw - SSE2 (128 bits registers) version

3.12.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
```

```
{
  int i;

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

3.12.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i=0;
  loop_label::
  if(i >= 8)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 - t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.12.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_sub_epi16(*(__m128i *) a, *(__m128i *) b);
}
```

3.12.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl %esp, %ebp
movl 8(%ebp), %eax	pushl %edi	pushl %edi
movdqa (%eax), %xmm0	pushl %esi	pushl %esi
movl 12(%ebp), %eax	pushl %ebx	xorl %esi, %esi
psubw (%eax), %xmm0	subl $4, %esp	pushl %ebx
movl 16(%ebp), %eax	movl $0, -16(%ebp)	subl $12, %esp
movdqa %xmm0, (%eax)	.L2:	movl 16(%ebp), %ebx
popl %ebp	cmpl $7, -16(%ebp)	andl $15, %ebx
ret	jle .L5	shrl %ebx
	jmp .L1	negl %ebx
	.L5:	andl $7, %ebx
	movl -16(%ebp), %eax	cmpl $0, %ebx
	leal (%eax,%eax), %esi	jbe .L20
	movl 16(%ebp), %edi	.p2align 4,,15
	movl -16(%ebp), %eax	.L11:
	leal (%eax,%eax), %ebx	movl 8(%ebp), %ecx

SIMD intrinsics	-nooptim	-O2 and vectorizer
	movl 8(%ebp), %edx	leal (%esi,%esi), %eax
	movl -16(%ebp), %eax	incl %esi
	leal (%eax,%eax), %ecx	movl 12(%ebp), %edi
	movl 12(%ebp), %eax	movzwl (%ecx,%eax), %edx
	movzwl (%edx,%ebx), %edx	movzwl (%edi,%eax), %ecx
	movzwl (%eax,%ecx), %eax	subl %ecx, %edx
	movl %edx, %ecx	movl 16(%ebp), %ecx
	subl %eax, %ecx	cmpl %esi, %ebx
	movl %ecx, %eax	movw %dx, (%ecx,%eax)
	movw %ax, (%edi,%esi)	ja .L11
	leal -16(%ebp), %eax	movl $8, -24(%ebp)
	incl (%eax)	subl %esi, -24(%ebp)
	jmp .L2	cmpl $8, %ebx
	.L1:	je .L13
	addl $4, %esp	.L4:
	popl %ebx	movl $8, -20(%ebp)
	popl %esi	subl %ebx, -20(%ebp)
	popl %edi	movl -20(%ebp), %edi
	popl %ebp	shrl $3, %edi
	ret	leal 0(,%edi,8), %eax
		cmpl $0, %eax
		movl %eax, -16(%ebp)
		jbe .L7
		leal (%ebx,%ebx), %eax
		movl 8(%ebp), %ecx
		movl 16(%ebp), %ebx
		movl 12(%ebp), %edx
		addl %eax, %ecx
		addl %eax, %edx
		addl %ebx, %eax
		xorl %ebx, %ebx
		.p2align 4,,15
		.L9:
		movdqu (%ecx), %xmm0
		movdqu (%edx), %xmm1
		incl %ebx
		psubw %xmm1, %xmm0
		addl $16, %ecx
		movdqa %xmm0, (%eax)
		addl $16, %edx
		addl $16, %eax
		cmpl %edi, %ebx
		jb .L9
		movl -16(%ebp), %edi
		subl %edi, -24(%ebp)
		addl %edi, %esi
		cmpl %edi, -20(%ebp)
		je .L13
		.L7:
		leal (%esi,%esi), %eax
		movl 12(%ebp), %ebx
		xorl %edi, %edi

SIMD intrinsics	-nooptim	-O2 and vectorizer
		`movl 8(%ebp), %esi`
		`movl 16(%ebp), %ecx`
		`addl %eax, %ebx`
		`addl %eax, %esi`
		`addl %eax, %ecx`
		`.p2align 4,,15`
		`.L12:`
		`movzwl (%esi), %eax`
		`incl %edi`
		`addl $2, %esi`
		`movzwl (%ebx), %edx`
		`addl $2, %ebx`
		`subl %edx, %eax`
		`movw %ax, (%ecx)`
		`addl $2, %ecx`
		`cmpl %edi, -24(%ebp)`
		`jne .L12`
		`.L13:`
		`addl $12, %esp`
		`popl %ebx`
		`popl %esi`
		`popl %edi`
		`popl %ebp`
		`ret`
		`.L20:`
		`movl $8, -24(%ebp)`
		`jmp .L4`

3.12.5 Benchmark

```
  short int a[8] __attribute__((aligned));
  short int b[8] __attribute__((aligned));
  short int c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = 140 + i;
      b[i] = 140 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	20.737
GCC 4.0 - not optimized	23.925
GCC 4.1 - not optimized	26.394
ICC 8.1 - not optimized	19.459

GCC 4.0	8.056
GCC 4.1	8.658
ICC 8.1	4.084

GCC SIMD	1.887
ICC SIMD	3.045

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `psubw`

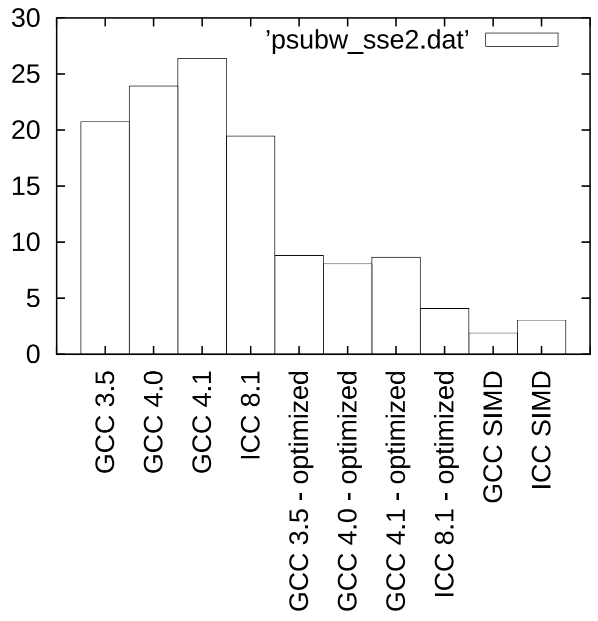

Figure 12: Benchmarks for psubw - SSE2 version

3.13 psubd - MMX (64 bits registers) version

3.13.1 C code

```
void sum(int a[2], int b[2], int c[2])
{
  int i;

  for(i=0; i<2; i++)
    {
      c[i] = a[i] - b[i];
    }
}
```

3.13.2 GIMPLE code

```
void sum(int a[2], int b[2], int c[2])
{
  int i=0;
  loop_label::
  if(i >= 2)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 - t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.13.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(int a[2], int b[2], int c[2])
{
   *(__m64 *) c = _mm_sub_pi32(*(__m64 *) a, *(__m64 *) b);
}
```

3.13.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %ecx`
`movl 8(%ebp), %eax`	`pushl %edi`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`pushl %esi`	`pushl %edi`
`movl 12(%ebp), %eax`	`pushl %ebx`	`movl 8(%ebp), %edi`
`psubd (%eax), %mm0`	`subl $4, %esp`	`pushl %esi`
`movl 16(%ebp), %eax`	`movl $0, -16(%ebp)`	`movl 12(%ebp), %esi`
`movq %mm0, (%eax)`	`.L2:`	`pushl %ebx`
`popl %ebp`	`cmpl $1, -16(%ebp)`	`movl 16(%ebp), %ebx`
`ret`	`jle .L5`	`.p2align 4,,15`
	`jmp .L1`	`.L2:`
	`.L5:`	`leal 0(,%ecx,4), %edx`
	`movl -16(%ebp), %eax`	`incl %ecx`
	`leal 0(,%eax,4), %esi`	`movl -4(%edx,%edi), %eax`
	`movl 16(%ebp), %edi`	`subl -4(%edx,%esi), %eax`
	`movl -16(%ebp), %eax`	`cmpl $3, %ecx`
	`leal 0(,%eax,4), %ebx`	`movl %eax, -4(%edx,%ebx)`
	`movl 8(%ebp), %ecx`	`jne .L2`
	`movl -16(%ebp), %eax`	`popl %ebx`
	`leal 0(,%eax,4), %edx`	`popl %esi`
	`movl 12(%ebp), %eax`	`popl %edi`
	`movl (%eax,%edx), %edx`	`popl %ebp`
	`movl (%ecx,%ebx), %eax`	`ret`
	`subl %edx, %eax`	
	`movl %eax, (%edi,%esi)`	
	`leal -16(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`addl $4, %esp`	

SIMD intrinsics	-nooptim	-O2 and vectorizer

	`popl %ebx`
	`popl %esi`
	`popl %edi`
	`popl %ebp`
	`ret`	

3.13.5 Benchmark

```
  int a[2] __attribute__((aligned));
  int b[2] __attribute__((aligned));
  int c[2] __attribute__((aligned));
  int i;

  for(i = 0; i<2; i++)
    {
      a[i] = 140000 + i;
      b[i] = 140000 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	4.848
GCC 4.0 - not optimized	4.805
GCC 4.1 - not optimized	5.182
ICC 8.1 - not optimized	4.576

GCC 4.0	3.077
GCC 4.1	3.076
ICC 8.1	2.556

GCC SIMD	0.971
ICC SIMD	2.181

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

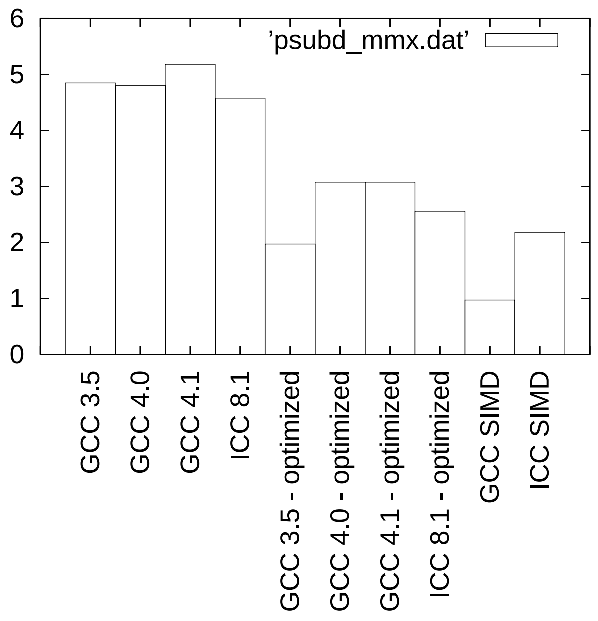

Figure 13: Benchmarks for psubd - MMX version

3.14 psubd - SSE2 (128 bits registers) version

3.14.1 C code

```
void test_loop_c(int a[4], int b[4], int c[4])
{
  int i;

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

3.14.2 GIMPLE code

```
void test_loop_c(int a[4], int b[4], int c[4])
{
  int i=0;
  loop_label::
  if(i >= 4)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 - t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.14.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(int a[4], int b[4], int c[4])
{
   *(__m128i *) c = _mm_sub_epi32(*(__m128i *) a, *(__m128i *) b);
}
```

3.14.4 Assembly code

The assembly codes that are generated are the following:

```
SIMD intrinsics             -nooptim                    -O2 and vectorizer
 pushl %ebp                  pushl %ebp                  pushl %ebp
 movl %esp, %ebp             movl %esp, %ebp             movl %esp, %ebp
 movl 8(%ebp), %eax          pushl %edi                  pushl %edi
 movdqa (%eax), %xmm0        pushl %esi                  pushl %esi
 movl 12(%ebp), %eax         pushl %ebx                  xorl %esi, %esi
 psubd (%eax), %xmm0         subl $4, %esp               pushl %ebx
 movl 16(%ebp), %eax         movl $0, -16(%ebp)          subl $12, %esp
 movdqa %xmm0, (%eax)       .L2:                         movl 16(%ebp), %ecx
 popl %ebp                   cmpl $3, -16(%ebp)          andl $15, %ecx
 ret                         jle .L5                     shrl $2, %ecx
                             jmp .L1                     negl %ecx
                            .L5:                         andl $3, %ecx
                             movl -16(%ebp), %eax        cmpl $0, %ecx
                             leal 0(,%eax,4), %esi       jbe .L20
                             movl 16(%ebp), %edi         .p2align 4,,15
                             movl -16(%ebp), %eax       .L11:
                             leal 0(,%eax,4), %ebx       movl 8(%ebp), %ebx
                             movl 8(%ebp), %ecx          leal 0(,%esi,4), %edx
                             movl -16(%ebp), %eax        incl %esi
                             leal 0(,%eax,4), %edx       movl (%ebx,%edx), %eax
                             movl 12(%ebp), %eax         movl 12(%ebp), %ebx
                             movl (%eax,%edx), %edx      movl (%ebx,%edx), %edi
                             movl (%ecx,%ebx), %eax      movl 16(%ebp), %ebx
                             subl %edx, %eax             subl %edi, %eax
                             movl %eax, (%edi,%esi)      cmpl %esi, %ecx
                             leal -16(%ebp), %eax        movl %eax, (%ebx,%edx)
                             incl (%eax)                 ja .L11
                             jmp .L2                     movl $4, -24(%ebp)
                            .L1:                         subl %esi, -24(%ebp)
                             addl $4, %esp               cmpl $4, %ecx
                             popl %ebx                   je .L13
                             popl %esi                  .L4:
                             popl %edi                   movl $4, -20(%ebp)
                             popl %ebp                   subl %ecx, -20(%ebp)
                             ret                         movl -20(%ebp), %edi
                                                         shrl $2, %edi
                                                         leal 0(,%edi,4), %eax
                                                         cmpl $0, %eax
                                                         movl %eax, -16(%ebp)
                                                         jbe .L7
                                                         leal 0(,%ecx,4), %eax
                                                         movl 16(%ebp), %ebx
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

```
 movl 8(%ebp), %ecx
 movl 12(%ebp), %edx
 addl %eax, %ecx
 addl %eax, %edx
 addl %ebx, %eax
 xorl %ebx, %ebx
 .p2align 4,,15
.L9:
 movdqu (%ecx), %xmm0
 movdqu (%edx), %xmm1
 incl %ebx
 psubd %xmm1, %xmm0
 addl $16, %ecx
 movdqa %xmm0, (%eax)
 addl $16, %edx
 addl $16, %eax
 cmpl %edi, %ebx
 jb .L9
 movl -16(%ebp), %eax
 subl %eax, -24(%ebp)
 addl %eax, %esi
 cmpl %eax, -20(%ebp)
 je .L13
.L7:
 movl 8(%ebp), %ebx
 leal 0(,%esi,4), %eax
 xorl %esi, %esi
 movl 12(%ebp), %ecx
 movl 16(%ebp), %edx
 addl %eax, %ebx
 addl %eax, %ecx
 addl %eax, %edx
 .p2align 4,,15
.L12:
 movl (%ebx), %eax
 incl %esi
 addl $4, %ebx
 movl (%ecx), %edi
 addl $4, %ecx
 subl %edi, %eax
 movl %eax, (%edx)
 addl $4, %edx
 cmpl %esi, -24(%ebp)
 jne .L12
.L13:
 addl $12, %esp
 popl %ebx
 popl %esi
 popl %edi
 popl %ebp
 ret
.L20:
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

		`movl $4, -24(%ebp)`
		`jmp .L4`

3.14.5 Benchmark

```
 int a[4] __attribute__((aligned));
 int b[4] __attribute__((aligned));
 int c[4] __attribute__((aligned));
 int i;

 for(i = 0; i<4; i++)
   {
     a[i] = 140 + i;
     b[i] = 140 + 2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	9.621
GCC 4.0 - not optimized	9.538
GCC 4.1 - not optimized	9.031
ICC 8.1 - not optimized	7.36

GCC 4.0	5.754
GCC 4.1	6.059
ICC 8.1	3.567

GCC SIMD	1.412
ICC SIMD	2.068

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	unrolling

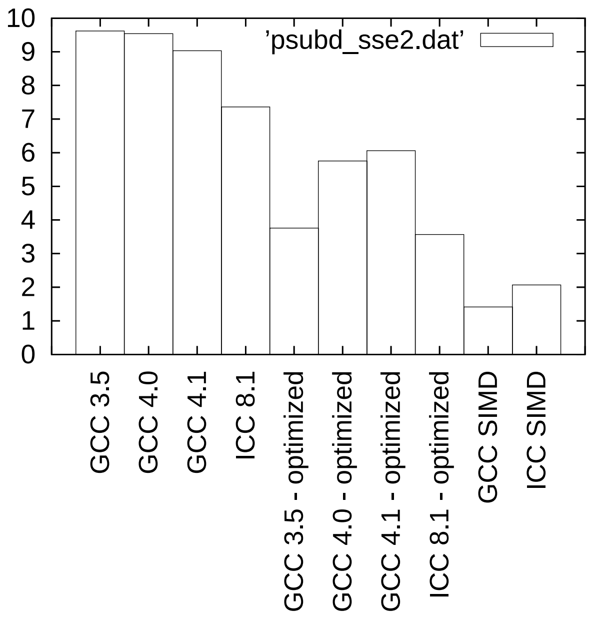

Figure 14: Benchmarks for psubd - SSE2 version

3.15 psubq - MMX (64 bits registers) version

3.15.1 C code

```
void test_loop_c(long long int a[1], long long int b[1], long long int c[1])
{
  c[0] = a[0] - b[0];
}
```

3.15.2 GIMPLE code

```
void test_loop_c(long long int a[1], long long int b[1], long long int c[1])
{
  t1 = a[0];
  t2 = b[0];
  t3 = t1 + t2;
  c[0] = t3;
}
```

3.15.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(long long int a[1], long long int b[1], long long int c[1])
{
   *(__m64 *) c = _mm_sub_si64(*(__m64 *) a, *(__m64 *) b);
}
```

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl %esp, %ebp
movl 8(%ebp), %eax	pushl %esi	movl 8(%ebp), %eax
movq (%eax), %mm0	pushl %ebx	movl 12(%ebp), %ecx
movl 12(%ebp), %eax	movl 16(%ebp), %esi	movl 4(%eax), %edx

SIMD intrinsics	-nooptim	-O2 and vectorizer
`psubq (%eax), %mm0`	`movl 8(%ebp), %edx`	`movl (%eax), %eax`
`movl 16(%ebp), %eax`	`movl 12(%ebp), %eax`	`subl (%ecx), %eax`
`movq %mm0, (%eax)`	`movl (%eax), %ecx`	`sbbl 4(%ecx), %edx`
`popl %ebp`	`movl 4(%eax), %ebx`	`movl 16(%ebp), %ecx`
`ret`	`movl (%edx), %eax`	`movl %eax, (%ecx)`
	`movl 4(%edx), %edx`	`movl %edx, 4(%ecx)`
	`subl %ecx, %eax`	`popl %ebp`
	`sbbl %ebx, %edx`	`ret`
	`movl %eax, (%esi)`	
	`movl %edx, 4(%esi)`	
	`popl %ebx`	
	`popl %esi`	
	`popl %ebp`	
	`ret`	

3.15.4 Benchmark

```
 long long int a[1] __attribute__((aligned));
 long long int b[1] __attribute__((aligned));
 long long int c[1] __attribute__((aligned));
 int i;

 for(i = 0; i<1; i++)
   {
     a[i] = 140000 + i;
     b[i] = 140000 + 2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	2.13
GCC 4.0 - not optimized	2.153
GCC 4.1 - not optimized	2.694
ICC 8.1 - not optimized	2.391
GCC 4.0	2.063
GCC 4.1	2.234
ICC 8.1	2.111
GCC SIMD	1.407
ICC SIMD	1.734

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

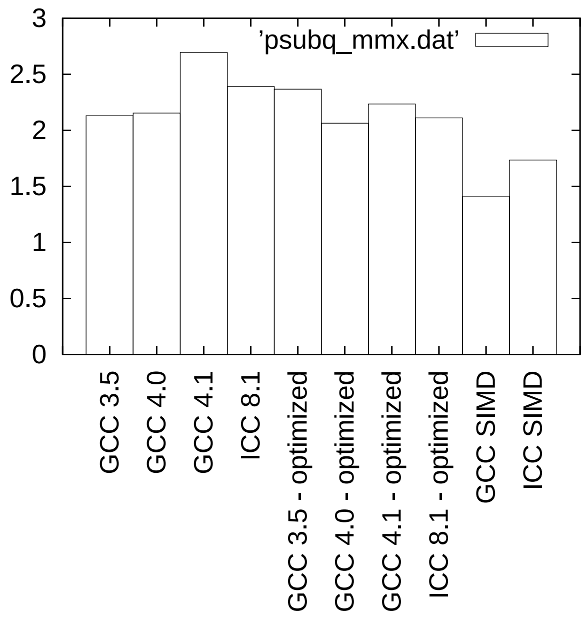

Figure 15: Benchmarks for psubq - MMX version

3.16 psubq - SSE2 (128 bits registers) version

3.16.1 C code

```
void test_loop_c(long long int a[2], long long int b[2], long long int c[2])
{
  int i;

  for(i=0; i<2; i++)
  {
    c[i] = a[i] - b[i];
  }
}
```

3.16.2 GIMPLE code

```
void test_loop_c(long long int a[2], long long int b[2], long long int c[2])
{
  int i=0;
  loop_label::
  if(i >= 2)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 - t2;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.16.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(long long int a[2], long long int b[2], long long int c[2])
{
   *(__m128i *) c = _mm_sub_epi64(*(__m128i *) a, *(__m128i *) b);
}
```

3.16.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                      -O2 and vectorizer
  pushl %ebp                   pushl %ebp                    pushl %ebp
  movl %esp, %ebp              movl %esp, %ebp               movl %esp, %ebp
  movl 12(%ebp), %eax          subl $4, %esp                 pushl %edi
  movdqa (%eax), %xmm0         movl $0, -4(%ebp)             xorl %edi, %edi
  movl 8(%ebp), %eax          .L2:                           pushl %esi
  paddb (%eax), %xmm0          cmpl $15, -4(%ebp)            pushl %ebx
  movl 16(%ebp), %eax          jle .L5                       subl $12, %esp
  movdqa %xmm0, (%eax)         jmp .L1                       movl 16(%ebp), %ebx
  popl %ebp                   .L5:                           negl %ebx
  ret                          movl -4(%ebp), %eax           andl $15, %ebx
                               movl 16(%ebp), %ecx           cmpl $0, %ebx
                               addl %eax, %ecx               jbe .L20
                               movl -4(%ebp), %eax           .p2align 4,,15
                               movl 8(%ebp), %edx           .L11:
                               addl %eax, %edx               movl 12(%ebp), %edx
                               movl -4(%ebp), %eax           movzbl (%edx,%edi), %eax
                               addl 12(%ebp), %eax           movl 8(%ebp), %edx
                               movzbl (%eax), %eax           addb (%edx,%edi), %al
                               addb (%edx), %al              movl 16(%ebp), %edx
                               movb %al, (%ecx)              movb %al, (%edx,%edi)
                               leal -4(%ebp), %eax           incl %edi
                               incl (%eax)                   cmpl %edi, %ebx
                               jmp .L2                       ja .L11
                              .L1:                           movl $16, -24(%ebp)
                               leave                         subl %edi, -24(%ebp)
                               ret                           cmpl $16, %ebx
                                                             je .L13
                                                            .L4:
                                                             movl $16, -20(%ebp)
                                                             subl %ebx, -20(%ebp)
                                                             movl -20(%ebp), %esi
                                                             shrl $4, %esi
                                                             movl %esi, %eax
                                                             sall $4, %eax
                                                             cmpl $0, %eax
                                                             movl %eax, -16(%ebp)
                                                             jbe .L7
                                                             movl 8(%ebp), %ecx
                                                             movl 12(%ebp), %edx
                                                             movl 16(%ebp), %eax
                                                             addl %ebx, %ecx
                                                             addl %ebx, %edx
```

SIMD intrinsics	-nooptim	-O2 and vectorizer
		addl %ebx, %eax xorl %ebx, %ebx .p2align 4,,15 .L9: movdqu (%ecx), %xmm0 movdqu (%edx), %xmm1 incl %ebx paddb %xmm1, %xmm0 addl $16, %ecx movdqa %xmm0, (%eax) addl $16, %edx addl $16, %eax cmpl %esi, %ebx jb .L9 movl -16(%ebp), %eax subl %eax, -24(%ebp) addl %eax, %edi cmpl %eax, -20(%ebp) je .L13 .L7: movl 16(%ebp), %ebx xorl %esi, %esi movl 12(%ebp), %ecx movl 8(%ebp), %edx addl %edi, %ebx addl %edi, %ecx addl %edi, %edx .p2align 4,,15 .L12: movzbl (%ecx), %eax incl %esi incl %ecx addb (%edx), %al incl %edx movb %al, (%ebx) incl %ebx cmpl %esi, -24(%ebp) jne .L12 .L13: addl $12, %esp popl %ebx popl %esi popl %edi popl %ebp ret .L20: movl $16, -24(%ebp) jmp .L4

3.16.5 Benchmark

```
long long int a[2] __attribute__((aligned));
long long int b[2] __attribute__((aligned));
long long int c[2] __attribute__((aligned));
int i;

for(i = 0; i<2; i++)
  {
      a[i] = 140000 + i;
      b[i] = 140000 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	7.941
GCC 4.0 - not optimized	6.57
GCC 4.1 - not optimized	7.772
ICC 8.1 - not optimized	5.835
GCC 4.0	5.882
GCC 4.1	6.152
ICC 8.1	4.347
GCC SIMD	1.421
ICC SIMD	2.296

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	unrolling

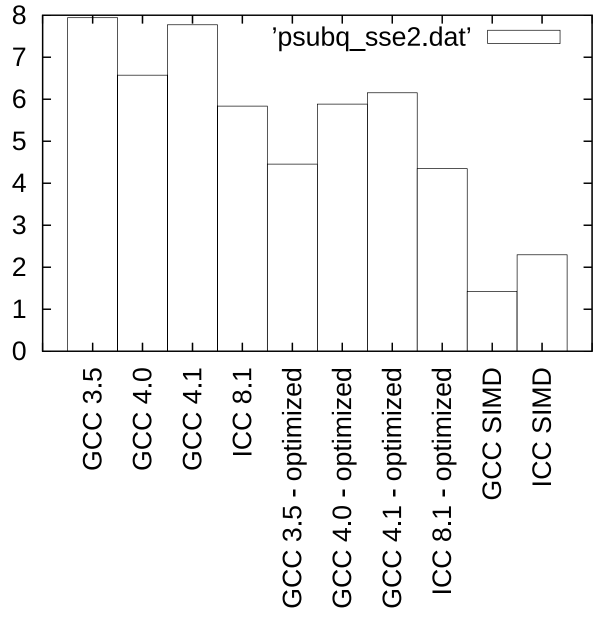

Figure 16: Benchmarks for psubq - SSE2 version

3.17 pmulhw - MMX (64 bits registers) version

3.17.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i, tmp;

  for(i=0; i<4; i++)
    {
      tmp = a[i] * b[i];
      c[i] = tmp >> 16;
    }
}
```

3.17.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i=0;
  int tmp;
  loop_label::
  if(i >= 4)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  tmp = t1 * t2;
  t3 = tmp >> 16;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.17.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_mulhi_pi16(*(__m64 *) a, *(__m64 *) b);
}
```

3.17.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`subl $8, %esp`	`pushl %edi`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`movl 12(%ebp), %edi`
`movl 8(%ebp), %eax`	`.L2:`	`pushl %esi`
`pmulhw (%eax), %mm0`	`cmpl $3, -4(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`pushl %ebx`
`movq %mm0, (%eax)`	`jmp .L1`	`movl $1, %ebx`
`popl %ebp`	`.L5:`	`.p2align 4,,15`
`ret`	`movl -4(%ebp), %eax`	`.L2:`
	`leal (%eax,%eax), %edx`	`movl 8(%ebp), %ecx`
	`movl 8(%ebp), %eax`	`leal (%ebx,%ebx), %eax`
	`movswl (%eax,%edx),%ecx`	`incl %ebx`
	`movl -4(%ebp), %eax`	`movswl -2(%eax,%ecx),%edx`
	`leal (%eax,%eax), %edx`	`movswl -2(%eax,%edi),%ecx`
	`movl 12(%ebp), %eax`	`imull %ecx, %edx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`movswl (%eax,%edx),%eax`	`sarl $16, %edx`
	`imull %ecx, %eax`	`cmpl $5, %ebx`
	`movl %eax, -8(%ebp)`	`movw %dx, -2(%eax,%esi)`
	`movl -4(%ebp), %eax`	`jne .L2`
	`leal (%eax,%eax), %ecx`	`popl %ebx`
	`movl 16(%ebp), %edx`	`popl %esi`
	`movl -8(%ebp), %eax`	`popl %edi`
	`sarl $16, %eax`	`popl %ebp`
	`movw %ax, (%edx,%ecx)`	`ret`
	`leal -4(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

3.17.5 Benchmark

```
  short int a[4] __attribute__((aligned));
  short int b[4] __attribute__((aligned));
  short int c[4] __attribute__((aligned));

  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = -14000 + i;
      b[i] = 14000 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	12.51
GCC 4.0 - not optimized	14.422
GCC 4.1 - not optimized	14.3
ICC 8.1 - not optimized	12.588

GCC 4.0	6.773
GCC 4.1	7.207
ICC 8.1	5.464

GCC SIMD	1.501
ICC SIMD	2.478

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

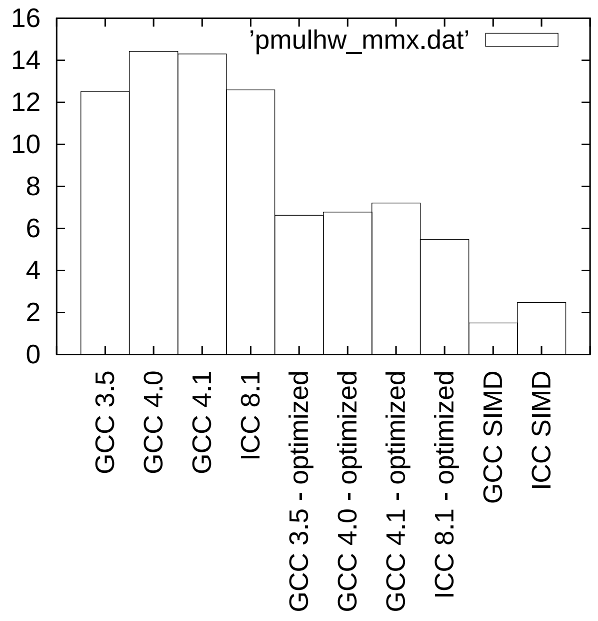

Figure 17: Benchmarks for pmulhw - MMX version

3.18 pmulhw - SSE2 (128 bits registers) version

3.18.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i, tmp;

  for(i=0; i<8; i++)
    {
      tmp = a[i] * b[i];
      c[i] = tmp >> 16;
    }
}
```

3.18.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i=0;
  int tmp;
  loop_label::
  if(i >= 8)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  tmp = t1 * t2;
  t3 = tmp >> 16;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.18.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_mulhi_epi16(*(__m128i *) a, *(__m128i *) b);
}
```

3.18.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                       -O2 and vectorizer
--------------------------------------------------------------------------------------
  pushl %ebp                   pushl %ebp                     pushl %ebp
  movl %esp, %ebp              movl %esp, %ebp                movl %esp, %ebp
  movl 12(%ebp), %eax          subl $8, %esp                  pushl %edi
  movdqa (%eax), %xmm0         movl $0, -4(%ebp)              movl 12(%ebp), %edi
  movl 8(%ebp), %eax          .L2:                            pushl %esi
  pmulhw (%eax), %xmm0         cmpl $7, -4(%ebp)              movl 16(%ebp), %esi
  movl 16(%ebp), %eax          jle .L5                        pushl %ebx
  movdqa %xmm0, (%eax)         jmp .L1                        movl $1, %ebx
  popl %ebp                   .L5:                            .p2align 4,,15
  ret                          movl -4(%ebp), %eax           .L2:
                               leal (%eax,%eax), %edx         movl 8(%ebp), %ecx
                               movl 8(%ebp), %eax             leal (%ebx,%ebx), %eax
                               movswl (%eax,%edx),%ecx        incl %ebx
                               movl -4(%ebp), %eax            movswl -2(%eax,%ecx),%edx
                               leal (%eax,%eax), %edx         movswl -2(%eax,%edi),%ecx
                               movl 12(%ebp), %eax            imull %ecx, %edx
                               movswl (%eax,%edx),%eax        sarl $16, %edx
                               imull %ecx, %eax               cmpl $9, %ebx
                               movl %eax, -8(%ebp)            movw %dx, -2(%eax,%esi)
                               movl -4(%ebp), %eax            jne .L2
                               leal (%eax,%eax), %ecx         popl %ebx
                               movl 16(%ebp), %edx            popl %esi
                               movl -8(%ebp), %eax            popl %edi
                               sarl $16, %eax                 popl %ebp
                               movw %ax, (%edx,%ecx)          ret
                               leal -4(%ebp), %eax
                               incl (%eax)
                               jmp .L2
                              .L1:
                               leave
                               ret
```

3.18.5 Benchmark

```
  short int a[8] __attribute__((aligned));
  short int b[8] __attribute__((aligned));
  short int c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = 14000 + i;
      b[i] = -14000 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	24.026
GCC 4.0 - not optimized	26.318
GCC 4.1 - not optimized	27.367
ICC 8.1 - not optimized	20.515
GCC 4.0	11.754
GCC 4.1	10.745
ICC 8.1	3.322
GCC SIMD	1.74
ICC SIMD	2.376

GCC 4.0 behavior	-O2 optimization, no vectorization
GCC 4.1 behavior	-O2 optimization, no vectorization
ICC behavior	vectorization with `pmulhw`

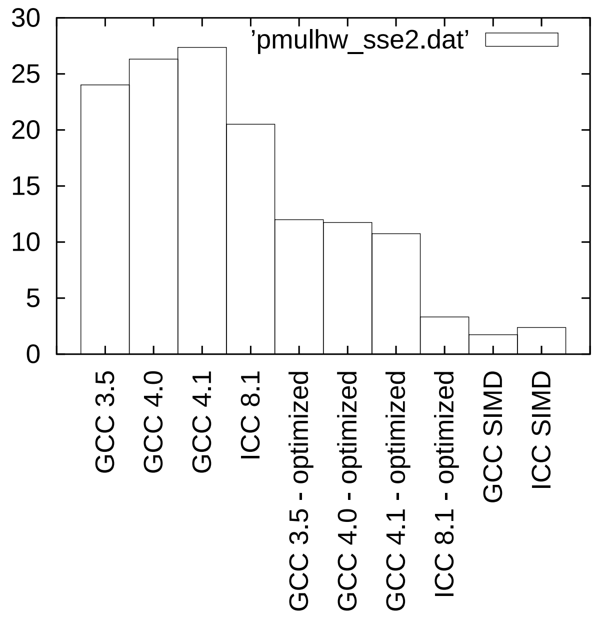

Figure 18: Benchmarks for pmulhw - SSE2 version

3.19 pmulhuw - MMX (64 bits registers) version

3.19.1 C code

```
void test_loop_c(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  int i
  unsigned int tmp;

  for(i=0; i<4; i++)
    {
      tmp = a[i] * b[i];
      c[i] = tmp >> 16;
    }
}
```

3.19.2 GIMPLE code

```
void test_loop_c(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  int i=0;
  int tmp;
  loop_label::
  if(i >= 4)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  tmp = t1 * t2;
  t3 = t3 >> 16;
  c[i] = tmp;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.19.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_mulhi_pu16(*(__m64 *) a, *(__m64 *) b);
}
```

3.19.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`subl $8, %esp`	`pushl %edi`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`movl 12(%ebp), %edi`
`movl 8(%ebp), %eax`	`.L2:`	`pushl %esi`
`pmulhuw (%eax), %mm0`	`cmpl $3, -4(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`pushl %ebx`
`movq %mm0, (%eax)`	`jmp .L1`	`movl $1, %ebx`
`popl %ebp`	`.L5:`	`.p2align 4,,15`
`ret`	`movl -4(%ebp), %eax`	`.L2:`
	`leal (%eax,%eax), %edx`	`movl 8(%ebp), %ecx`
	`movl 8(%ebp), %eax`	`leal (%ebx,%ebx), %eax`
	`movzwl (%eax,%edx), %ecx`	`incl %ebx`
	`movl -4(%ebp), %eax`	`movzwl -2(%eax,%ecx), %edx`
	`leal (%eax,%eax), %edx`	`movzwl -2(%eax,%edi), %ecx`
	`movl 12(%ebp), %eax`	`imull %ecx, %edx`
	`movzwl (%eax,%edx), %eax`	`sarl $16, %edx`
	`imull %ecx, %eax`	`cmpl $5, %ebx`
	`movl %eax, -8(%ebp)`	`movw %dx, -2(%eax,%esi)`
	`movl -4(%ebp), %eax`	`jne .L2`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`leal (%eax,%eax), %ecx`	`popl %ebx`
	`movl 16(%ebp), %edx`	`popl %esi`
	`movl -8(%ebp), %eax`	`popl %edi`
	`sarl $16, %eax`	`popl %ebp`
	`movw %ax, (%edx,%ecx)`	`ret`
	`leal -4(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

3.19.5 Benchmark

```
  short int a[4] __attribute__((aligned));
  short int b[4] __attribute__((aligned));
  short int c[4] __attribute__((aligned));

  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = 14000 + i;
      b[i] = 14000 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	8.953
GCC 4.0 - not optimized	11.695
GCC 4.1 - not optimized	11.729
ICC 8.1 - not optimized	10.252

GCC 4.0	4.56
GCC 4.1	4.714
ICC 8.1	3.6

GCC SIMD	1.27
ICC SIMD	1.247

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

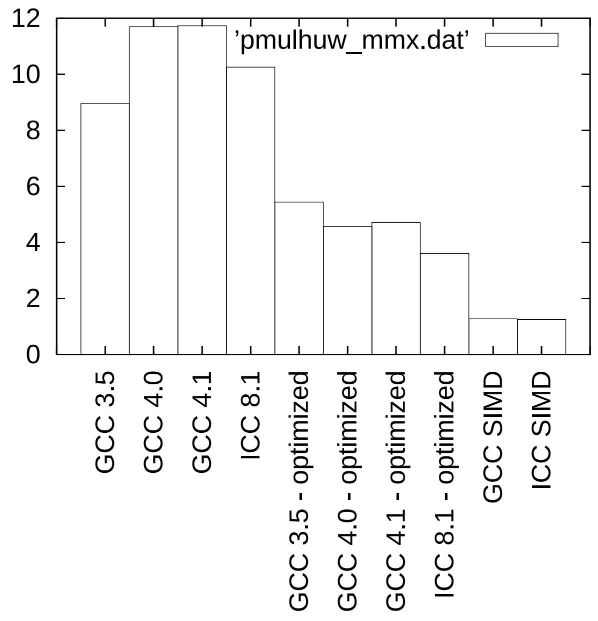

Figure 19: Benchmarks for pmulhuw - MMX version

3.20 pmulhuw - SSE2 (128 bits registers) version

3.20.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i, tmp;

  for(i=0; i<8; i++)
    {
      tmp = a[i] * b[i];
      c[i] = tmp >> 16;
    }
}
```

3.20.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i=0;
  int tmp;
  loop_label::
  if(i >= 8)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  tmp = t1 * t2;
  t3 = tmp >> 16;
  c[i] = t3;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.20.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_mulhi_epu16(*(__m128i *) a, *(__m128i *) b);
}
```

3.20.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl %esp, %ebp
movl 12(%ebp), %eax	subl $8, %esp	pushl %edi
movdqa (%eax), %xmm0	movl $0, -4(%ebp)	movl 12(%ebp), %edi
movl 8(%ebp), %eax	.L2:	pushl %esi
pmulhuw (%eax), %xmm0	cmpl $7, -4(%ebp)	movl 16(%ebp), %esi
movl 16(%ebp), %eax	jle .L5	pushl %ebx
movdqa %xmm0, (%eax)	jmp .L1	movl $1, %ebx
popl %ebp	.L5:	.p2align 4,,15
ret	movl -4(%ebp), %eax	.L2:
	leal (%eax,%eax), %edx	movl 8(%ebp), %ecx
	movl 8(%ebp), %eax	leal (%ebx,%ebx), %eax
	movzwl (%eax,%edx), %ecx	incl %ebx
	movl -4(%ebp), %eax	movzwl -2(%eax,%ecx), %edx
	leal (%eax,%eax), %edx	movzwl -2(%eax,%edi), %ecx
	movl 12(%ebp), %eax	imull %ecx, %edx
	movzwl (%eax,%edx), %eax	sarl $16, %edx
	imull %ecx, %eax	cmpl $9, %ebx
	movl %eax, -8(%ebp)	movw %dx, -2(%eax,%esi)
	movl -4(%ebp), %eax	jne .L2
	leal (%eax,%eax), %ecx	popl %ebx
	movl 16(%ebp), %edx	popl %esi
	movl -8(%ebp), %eax	popl %edi
	sarl $16, %eax	popl %ebp
	movw %ax, (%edx,%ecx)	ret
	leal -4(%ebp), %eax	
	incl (%eax)	
	jmp .L2	
	.L1:	
	leave	
	ret	

3.20.5 Benchmark

```
  short int a[8] __attribute__((aligned));
  short int b[8] __attribute__((aligned));
  short int c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = 14000 + i;
      b[i] = -14000 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	21.68
GCC 4.0 - not optimized	24.087
GCC 4.1 - not optimized	24.761
ICC 8.1 - not optimized	22.152

GCC 4.0	10.287
GCC 4.1	9.995
ICC 8.1	3.466

GCC SIMD	1.245
ICC SIMD	2.471

GCC 4.0 behavior	-O2 optimization, no vectorization
GCC 4.1 behavior	-O2 optimization, no vectorization
ICC behavior	vectorization with `pmulhuw`

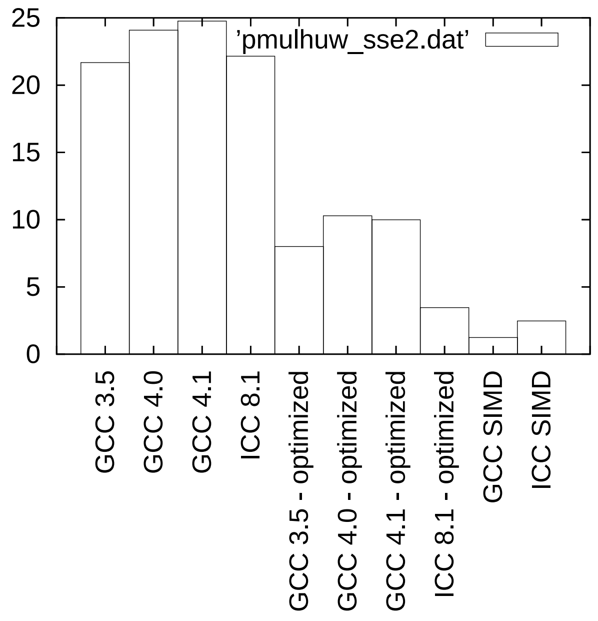

Figure 20: Benchmarks for pmulhuw - SSE2 version

3.21 pmullw - MMX (64 bits registers) version

3.21.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i, tmp;

  for(i=0; i<4; i++)
    {
      tmp = a[i] * b[i];
      c[i] = (tmp << 16) >> 16;
    }
}
```

3.21.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i=0;
  int tmp;
  loop_label::
  if(i >= 4)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  tmp = t1 * t2;
  t3 = tmp << 16;
  t4 = t3 >> 16;
  c[i] = t4;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.21.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_mullo_pi16(*(__m64 *) a, *(__m64 *) b);
}
```

3.21.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`subl $8, %esp`	`pushl %edi`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`movl 12(%ebp), %edi`
`movl 8(%ebp), %eax`	`.L2:`	`pushl %esi`
`pmullw (%eax), %mm0`	`cmpl $3, -4(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`pushl %ebx`
`movq %mm0, (%eax)`	`jmp .L1`	`movl $1, %ebx`
`popl %ebp`	`.L5:`	`.p2align 4,,15`
`ret`	`movl -4(%ebp), %eax`	`.L2:`
	`leal (%eax,%eax), %edx`	`movl 8(%ebp), %ecx`
	`movl 8(%ebp), %eax`	`leal (%ebx,%ebx), %eax`
	`movswl (%eax,%edx),%ecx`	`incl %ebx`
	`movl -4(%ebp), %eax`	`movswl -2(%eax,%ecx),%edx`
	`leal (%eax,%eax), %edx`	`movswl -2(%eax,%edi),%ecx`
	`movl 12(%ebp), %eax`	`imull %ecx, %edx`
	`movswl (%eax,%edx),%eax`	`cmpl $5, %ebx`
	`imull %ecx, %eax`	`movw %dx, -2(%eax,%esi)`
	`movl %eax, -8(%ebp)`	`jne .L2`
	`movl -4(%ebp), %eax`	`popl %ebx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`leal (%eax,%eax), %ecx`	`popl %esi`
	`movl 16(%ebp), %edx`	`popl %edi`
	`movl -8(%ebp), %eax`	`popl %ebp`
	`sall $16, %eax`	`ret`
	`sarl $16, %eax`	
	`movw %ax, (%edx,%ecx)`	
	`leal -4(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

3.21.5 Benchmark

```
  short int a[4] __attribute__((aligned));
  short int b[4] __attribute__((aligned));
  short int c[4] __attribute__((aligned));

  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = 14000 + i;
      b[i] = 14000 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	12.683
GCC 4.0 - not optimized	15.507
GCC 4.1 - not optimized	15.218
ICC 8.1 - not optimized	12.103

GCC 4.0	6.214
GCC 4.1	6.166
ICC 8.1	4.359

GCC SIMD	1.554
ICC SIMD	2.27

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

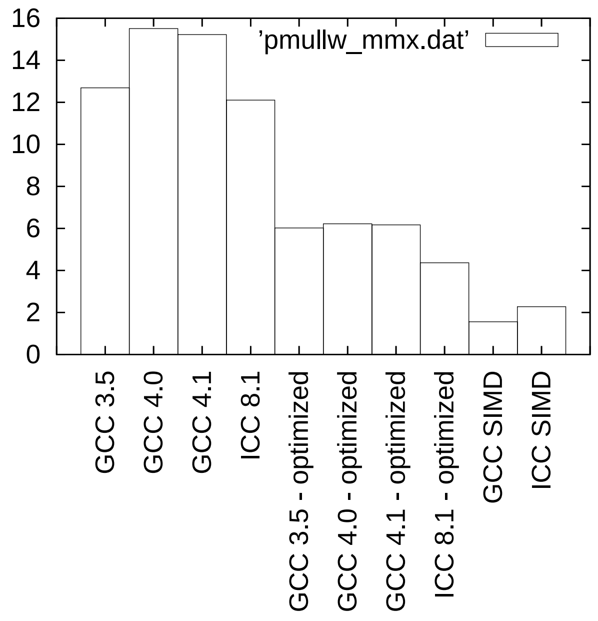

Figure 21: Benchmarks for pmullw - MMX version

3.22 pmullw - SSE2 (128 bits registers) version

3.22.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i, tmp;

  for(i=0; i<8; i++)
    {
      tmp = a[i] * b[i];
      c[i] = (tmp << 16) >> 16;
    }
}
```

3.22.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i=0;
  int tmp;
  loop_label::
  if(i >= 8)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  tmp = t1 * t2;
  t3 = tmp << 16;
  t4 = tmp >> 16;
  c[i] = t4;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

3.22.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_mullo_epi16(*(__m128i *) a, *(__m128i *) b);
}
```

3.22.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl %esp, %ebp
movl 12(%ebp), %eax	subl $8, %esp	pushl %edi
movdqa (%eax), %xmm0	movl $0, -4(%ebp)	movl 12(%ebp), %edi
movl 8(%ebp), %eax	.L2:	pushl %esi
pmullw (%eax), %xmm0	cmpl $7, -4(%ebp)	movl 16(%ebp), %esi
movl 16(%ebp), %eax	jle .L5	pushl %ebx
movdqa %xmm0, (%eax)	jmp .L1	movl $1, %ebx
popl %ebp	.L5:	.p2align 4,,15
ret	movl -4(%ebp), %eax	.L2:
	leal (%eax,%eax), %edx	movl 8(%ebp), %ecx
	movl 8(%ebp), %eax	leal (%ebx,%ebx), %eax
	movswl (%eax,%edx),%ecx	incl %ebx
	movl -4(%ebp), %eax	movswl -2(%eax,%ecx),%edx
	leal (%eax,%eax), %edx	movswl -2(%eax,%edi),%ecx
	movl 12(%ebp), %eax	imull %ecx, %edx
	movswl (%eax,%edx),%eax	cmpl $9, %ebx
	imull %ecx, %eax	movw %dx, -2(%eax,%esi)
	movl %eax, -8(%ebp)	jne .L2
	movl -4(%ebp), %eax	popl %ebx
	leal (%eax,%eax), %ecx	popl %esi
	movl 16(%ebp), %edx	popl %edi
	movl -8(%ebp), %eax	popl %ebp
	sall $16, %eax	ret
	sarl $16, %eax	
	movw %ax, (%edx,%ecx)	
	leal -4(%ebp), %eax	
	incl (%eax)	
	jmp .L2	
	.L1:	
	leave	
	ret	

3.22.5 Benchmark

```
  short int a[8] __attribute__((aligned));
  short int b[8] __attribute__((aligned));
  short int c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = 14000 + i;
      b[i] = 14000 + 2*i;
    }
```

```
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	24.317
GCC 4.0 - not optimized	29.737
GCC 4.1 - not optimized	27.67
ICC 8.1 - not optimized	15.682

GCC 4.0	10.722
GCC 4.1	6.873
ICC 8.1	3.477

GCC SIMD	2.009
ICC SIMD	1.247

GCC 4.0 behavior	-O2 optimization, no vectorization
GCC 4.1 behavior	-O2 optimization, no vectorization
ICC behavior	vectorization with `pmullo`

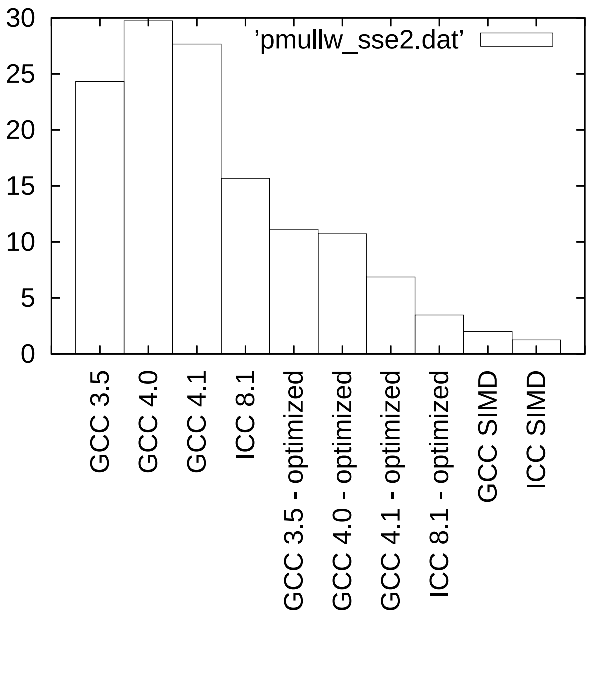

Figure 22: Benchmarks for pmullw - SSE2 version

4 Arithmetic operations with saturation

4.1 paddsb - MMX (64 bits registers) version

4.1.1 C code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i, k;

  for(i=0; i<8; i++)
    {
      k = a[i] + b[i];
      if(k > 127)
        {
          c[i] = 127;
        }
      else if(k<-128)
        {
          c[i] = -128;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.1.2 GIMPLE code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i, k;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 + T2;
  if(k > 127)
    T3 = 127;
  else if(k<-128)
    T3 = -128;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.1.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[8], char b[8], char c[8])
{
  *(__m64 *) c = _mm_adds_pi8(*(__m64*) a,  *(__m64*) b);
}
```

4.1.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %ecx`
`movl 8(%ebp), %eax`	`subl $8, %esp`	`movl %esp, %ebp`

```
SIMD intrinsics             -nooptim                    -O2 and vectorizer
 movl 12(%ebp), %edx         movl $0, -4(%ebp)           pushl %edi
 movq (%eax), %mm0          .L2:                         movl 8(%ebp), %edi
 movl 16(%ebp), %eax         cmpl $7, -4(%ebp)           pushl %esi
 paddsb (%edx), %mm0         jle .L5                     movl 12(%ebp), %esi
 movq %mm0, (%eax)           jmp .L1                     pushl %ebx
 popl %ebp                  .L5:                         movl 16(%ebp), %ebx
 ret                         movl -4(%ebp), %eax         jmp .L2
                             addl 8(%ebp), %eax          .p2align 4,,7
                             movsbl (%eax),%edx         .L14:
                             movl -4(%ebp), %eax         movb $127, -1(%ebx,%ecx)
                             addl 12(%ebp), %eax        .L5:
                             movsbl (%eax),%eax          incl %ecx
                             leal (%eax,%edx), %eax      cmpl $9, %ecx
                             movl %eax, -8(%ebp)         je .L13
                             cmpl $127, -8(%ebp)        .L2:
                             jle .L6                     movsbl -1(%edi,%ecx),%edx
                             movl -4(%ebp), %eax         movsbl -1(%esi,%ecx),%eax
                             addl 16(%ebp), %eax         leal (%edx,%eax), %eax
                             movb $127, (%eax)           cmpl $127, %eax
                             jmp .L4                     jg .L14
                            .L6:                         cmpl $-128, %eax
                             cmpl $-128, -8(%ebp)        jge .L6
                             jge .L8                     movb $-128, -1(%ebx,%ecx)
                             movl -4(%ebp), %eax         incl %ecx
                             addl 16(%ebp), %eax         cmpl $9, %ecx
                             movb $-128, (%eax)          jne .L2
                             jmp .L4                     .p2align 4,,15
                            .L8:                        .L13:
                             movl -4(%ebp), %eax         popl %ebx
                             movl 16(%ebp), %edx         popl %esi
                             addl %eax, %edx             popl %edi
                             movzbl -8(%ebp), %eax       popl %ebp
                             movb %al, (%edx)            ret
                            .L4:                         .p2align 4,,7
                             leal -4(%ebp), %eax        .L6:
                             incl (%eax)                 movb %al, -1(%ebx,%ecx)
                             jmp .L2                     jmp .L5
                            .L1:
                             leave
                             ret
```

4.1.5 Benchmark

```
  char a[8] __attribute__((aligned));
  char b[8] __attribute__((aligned));
  char c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = -60-i;
      b[i] = -60-2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	15.609
GCC 4.0 - not optimized	18.529
GCC 4.1 - not optimized	17.342
ICC 8.1 - not optimized	15.368

GCC 4.0	6.918
GCC 4.1	7.884
ICC 8.1	6.639

GCC SIMD	1.085
ICC SIMD	1.406

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

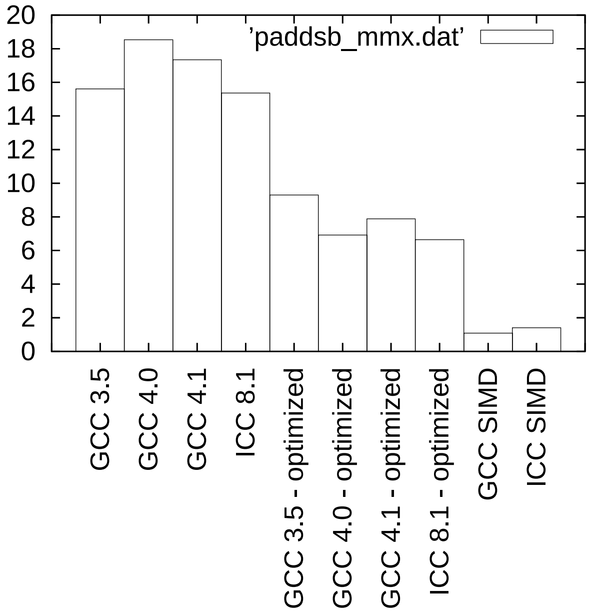

Figure 23: Benchmarks for paddsb - MMX version

4.2 paddsb - SSE2 (128 bits registers) version

4.2.1 C code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i, k;

  for(i=0; i<16; i++)
    {
      k = a[i] + b[i];
      if(k > 127)
        {
          c[i] = 127;
        }
      else if(k<-128)
        {
          c[i] = -128;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.2.2 GIMPLE code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i, k;
  loop_label::
  if(i >= 16)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 + T2;
  if(k > 127)
    T3 = 127;
  else if(k<-128)
    T3 = -128;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.2.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[16], char b[16], char c[16])
{
  *(__m128i *) c = _mm_adds_epi8(*(__m128i*) a,  *(__m128i*) b);
}
```

4.2.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`pushl %edi`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`xorl %edi, %edi`
`movl 8(%ebp), %eax`	`.L2:`	`pushl %esi`
`paddb (%eax), %xmm0`	`cmpl $15, -4(%ebp)`	`pushl %ebx`
`movl 16(%ebp), %eax`	`jle .L5`	`subl $12, %esp`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`movl 16(%ebp), %ebx`
`popl %ebp`	`.L5:`	`negl %ebx`
`ret`	`movl -4(%ebp), %eax`	`andl $15, %ebx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	movl 16(%ebp), %ecx	cmpl $0, %ebx
	addl %eax, %ecx	jbe .L20
	movl -4(%ebp), %eax	.p2align 4,,15
	movl 8(%ebp), %edx	.L11:
	addl %eax, %edx	movl 12(%ebp), %edx
	movl -4(%ebp), %eax	movzbl (%edx,%edi), %eax
	addl 12(%ebp), %eax	movl 8(%ebp), %edx
	movzbl (%eax), %eax	addb (%edx,%edi), %al
	addb (%edx), %al	movl 16(%ebp), %edx
	movb %al, (%ecx)	movb %al, (%edx,%edi)
	leal -4(%ebp), %eax	incl %edi
	incl (%eax)	cmpl %edi, %ebx
	jmp .L2	ja .L11
	.L1:	movl $16, -24(%ebp)
	leave	subl %edi, -24(%ebp)
	ret	cmpl $16, %ebx
		je .L13
		.L4:
		movl $16, -20(%ebp)
		subl %ebx, -20(%ebp)
		movl -20(%ebp), %esi
		shrl $4, %esi
		movl %esi, %eax
		sall $4, %eax
		cmpl $0, %eax
		movl %eax, -16(%ebp)
		jbe .L7
		movl 8(%ebp), %ecx
		movl 12(%ebp), %edx
		movl 16(%ebp), %eax
		addl %ebx, %ecx
		addl %ebx, %edx
		addl %ebx, %eax
		xorl %ebx, %ebx
		.p2align 4,,15
		.L9:
		movdqu (%ecx), %xmm0
		movdqu (%edx), %xmm1
		incl %ebx
		paddb %xmm1, %xmm0
		addl $16, %ecx
		movdqa %xmm0, (%eax)
		addl $16, %edx
		addl $16, %eax
		cmpl %esi, %ebx
		jb .L9
		movl -16(%ebp), %eax
		subl %eax, -24(%ebp)
		addl %eax, %edi
		cmpl %eax, -20(%ebp)
		je .L13
		.L7:

SIMD intrinsics	-nooptim	-O2 and vectorizer
		movl 16(%ebp), %ebx
		xorl %esi, %esi
		movl 12(%ebp), %ecx
		movl 8(%ebp), %edx
		addl %edi, %ebx
		addl %edi, %ecx
		addl %edi, %edx
		.p2align 4,,15
		.L12:
		movzbl (%ecx), %eax
		incl %esi
		incl %ecx
		addb (%edx), %al
		incl %edx
		movb %al, (%ebx)
		incl %ebx
		cmpl %esi, -24(%ebp)
		jne .L12
		.L13:
		addl $12, %esp
		popl %ebx
		popl %esi
		popl %edi
		popl %ebp
		ret
		.L20:
		movl $16, -24(%ebp)
		jmp .L4

4.2.5 Benchmark

```
  char a[16] __attribute__((aligned));
  char b[16] __attribute__((aligned));
  char c[16] __attribute__((aligned));
  int i;

  for(i = 0; i<16; i++)
    {
      a[i] = -60-i;
      b[i] = -60-2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	33.914
GCC 4.0 - not optimized	33.209
GCC 4.1 - not optimized	33.567
ICC 8.1 - not optimized	30.89

GCC 4.0	16.131
GCC 4.1	16.331
ICC 8.1	2.971

GCC SIMD	1.047
ICC SIMD	2.077

GCC 4.0 behavior	-O2 optimization, no vectorization
GCC 4.1 behavior	-O2 optimization, no vectorization
ICC behavior	vectorization with `paddsb`

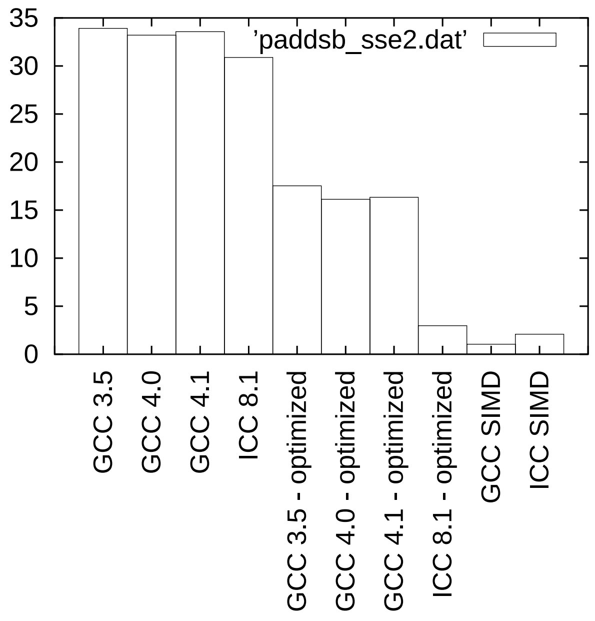

Figure 24: Benchmarks for paddsb - SSE2 version

4.3 paddsw - MMX (64 bits registers) version

4.3.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i, k;

  for(i=0; i<4; i++)
    {
      k = a[i] + b[i];
      if(k > 32767)
        {
          c[i] = 32767;
        }
      else if(k<-32768)
        {
          c[i] = -32768;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.3.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i, k;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 + T2;
  if(k > 32767)
    T3 = 32767;
  else if(k<-32768)
    T3 = -32768;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.3.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_adds_pi16(*(__m64*) a,  *(__m64*) b);
}
```

4.3.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl %esp, %ebp
movl 8(%ebp), %eax	subl $16, %esp	pushl %edi
movl 12(%ebp), %edx	movl $0, -4(%ebp)	movl 12(%ebp), %edi
movq (%eax), %mm0	.L2:	pushl %esi
movl 16(%ebp), %eax	cmpl $3, -4(%ebp)	movl 16(%ebp), %esi
paddsw (%edx), %mm0	jle .L5	pushl %ebx
movq %mm0, (%eax)	jmp .L1	movl $1, %ebx
popl %ebp	.L5:	jmp .L2
ret	movl -4(%ebp), %eax	.p2align 4,,7
	leal (%eax,%eax), %edx	.L18:
	movl 8(%ebp), %eax	jle .L16
	movswl (%eax,%edx),%ecx	.L5:
	movl -4(%ebp), %eax	movw $32767, -2(%esi,%ecx)
	leal (%eax,%eax), %edx	.L6:

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`movl 12(%ebp), %eax`	`incl %ebx`
	`movswl (%eax,%edx),%eax`	`cmpl $5, %ebx`
	`leal (%eax,%ecx), %eax`	`je .L17`
	`cltd`	`.L2:`
	`movl %eax, -16(%ebp)`	`movl 8(%ebp), %edx`
	`movl %edx, -12(%ebp)`	`leal (%ebx,%ebx), %ecx`
	`cmpl $0, -12(%ebp)`	`movswl -2(%edx,%ecx),%eax`
	`js .L6`	`movswl -2(%edi,%ecx),%edx`
	`cmpl $0, -12(%ebp)`	`addl %edx, %eax`
	`jg .L7`	`cltd`
	`cmpl $32767, -16(%ebp)`	`cmpl $0, %edx`
	`jbe .L6`	`jge .L18`
	`.L7:`	`incl %edx`
	`movl -4(%ebp), %eax`	`jle .L19`
	`leal (%eax,%eax), %edx`	`.L7:`
	`movl 16(%ebp), %eax`	`incl %ebx`
	`movw $32767, (%eax,%edx)`	`cmpl $5, %ebx`
	`jmp .L4`	`movw %ax, -2(%esi,%ecx)`
	`.L6:`	`jne .L2`
	`cmpl $-1, -12(%ebp)`	`.p2align 4,,15`
	`jg .L9`	`.L17:`
	`cmpl $-1, -12(%ebp)`	`popl %ebx`
	`jl .L10`	`popl %esi`
	`cmpl $-32768, -16(%ebp)`	`popl %edi`
	`jae .L9`	`popl %ebp`
	`.L10:`	`ret`
	`movl -4(%ebp), %eax`	`.L19:`
	`leal (%eax,%eax), %edx`	`.p2align 4,,2`
	`movl 16(%ebp), %eax`	`jl .L9`
	`movw $-32768, (%eax,%edx)`	`cmpl $-32768, %eax`
	`jmp .L4`	`.p2align 4,,6`
	`.L9:`	`jae .L7`
	`movl -4(%ebp), %eax`	`.L9:`
	`leal (%eax,%eax), %ecx`	`movw`
		`$-32768, -2(%esi,%ecx)`
	`movl 16(%ebp), %edx`	`.p2align 4,,6`
	`movl -16(%ebp), %eax`	`jmp .L6`
	`movw %ax, (%edx,%ecx)`	`.p2align 4,,7`
	`.L4:`	`.L16:`
	`leal -4(%ebp), %eax`	`cmpl $32767, %eax`
	`incl (%eax)`	`.p2align 4,,4`
	`jmp .L2`	`ja .L5`
	`.L1:`	`incl %edx`
	`leave`	`.p2align 4,,5`
	`ret`	`jg .L7`
		`.p2align 4,,8`
		`jmp .L19`

4.3.5 Benchmark

```
short int a[4] __attribute__((aligned));
short int b[4] __attribute__((aligned));
short int c[4] __attribute__((aligned));
int i;

for(i = 0; i<4; i++)
  {
    a[i] = 16000+i;
```

```
      b[i] = 16763+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	12.421
GCC 4.0 - not optimized	12.151
GCC 4.1 - not optimized	12.279
ICC 8.1 - not optimized	13.202
GCC 4.0	6.84
GCC 4.1	6.581
ICC 8.1	5.202
GCC SIMD	1.071
ICC SIMD	1.491

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

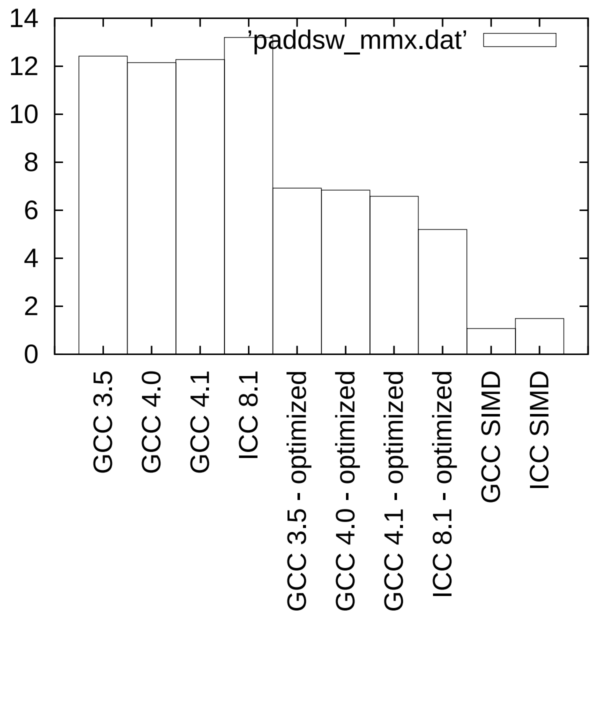

Figure 25: Benchmarks for paddsw - MMX version

4.4 paddsw - SSE2 (128 bits registers) version

4.4.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i, k;
  for(i=0; i<8; i++)
    {
      k = a[i] + b[i];
      if(k > 32767)
        {
          c[i] = 32767;
        }
      else if(k<-32768)
        {
          c[i] = -32768;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.4.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i, k;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 + T2;
  if(k > 32767)
    T3 = 32767;
  else if(k<-32768)
    T3 = -32768;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.4.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_adds_epi16(*(__m128i*) a,  *(__m128i*) b);
}
```

4.4.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`subl $16, %esp`	`pushl %edi`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`movl 12(%ebp), %edi`

```
SIMD intrinsics               -nooptim                      -O2 and vectorizer
 movl 8(%ebp), %eax           .L2:                           pushl %esi
 paddsw (%eax), %xmm0          cmpl $7, -4(%ebp)             movl 16(%ebp), %esi
 movl 16(%ebp), %eax           jle .L5                       pushl %ebx
 movdqa %xmm0, (%eax)          jmp .L1                       movl $1, %ebx
 popl %ebp                    .L5:                           jmp .L2
 ret                           movl -4(%ebp), %eax           .p2align 4,,7
                               leal (%eax,%eax), %edx       .L18:
                               movl 8(%ebp), %eax            jle .L16
                               movswl (%eax,%edx),%ecx      .L5:
                               movl -4(%ebp), %eax           movw $32767, -2(%esi,%ecx)
                               leal (%eax,%eax), %edx       .L6:
                               movl 12(%ebp), %eax           incl %ebx
                               movswl (%eax,%edx),%eax       cmpl $9, %ebx
                               leal (%eax,%ecx), %eax        je .L17
                               cltd                         .L2:
                               movl %eax, -16(%ebp)          movl 8(%ebp), %edx
                               movl %edx, -12(%ebp)          leal (%ebx,%ebx), %ecx
                               cmpl $0, -12(%ebp)            movswl -2(%edx,%ecx),%eax
                               js .L6                        movswl -2(%edi,%ecx),%edx
                               cmpl $0, -12(%ebp)            addl %edx, %eax
                               jg .L7                        cltd
                               cmpl $32767, -16(%ebp)        cmpl $0, %edx
                               jbe .L6                       jge .L18
                              .L7:                           incl %edx
                               movl -4(%ebp), %eax           jle .L19
                               leal (%eax,%eax), %edx       .L7:
                               movl 16(%ebp), %eax           incl %ebx
                               movw $32767, (%eax,%edx)      cmpl $9, %ebx
                               jmp .L4                       movw %ax, -2(%esi,%ecx)
                              .L6:                           jne .L2
                               cmpl $-1, -12(%ebp)           .p2align 4,,15
                               jg .L9                       .L17:
                               cmpl $-1, -12(%ebp)           popl %ebx
                               jl .L10                       popl %esi
                               cmpl $-32768, -16(%ebp)       popl %edi
                               jae .L9                       popl %ebp
                              .L10:                          ret
                               movl -4(%ebp), %eax          .L19:
                               leal (%eax,%eax), %edx        .p2align 4,,2
                               movl 16(%ebp), %eax           jl .L9
                               movw $-32768, (%eax,%edx)     cmpl $-32768, %eax
                               jmp .L4                       .p2align 4,,6
                              .L9:                           jae .L7
                               movl -4(%ebp), %eax          .L9:
                               leal (%eax,%eax), %ecx        movw
                                                            $-32768, -2(%esi,%ecx)
                               movl 16(%ebp), %edx           .p2align 4,,6
                               movl -16(%ebp), %eax          jmp .L6
                               movw %ax, (%edx,%ecx)         .p2align 4,,7
                              .L4:                          .L16:
                               leal -4(%ebp), %eax           cmpl $32767, %eax
                               incl (%eax)                   .p2align 4,,4
```

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`jmp .L2`	`ja .L5`
	`.L1:`	`incl %edx`
	`leave`	`.p2align 4,,5`
	`ret`	`jg .L7`
		`.p2align 4,,8`
		`jmp .L19`

4.4.5 Benchmark

```
 short int a[8] __attribute__((aligned));
 short int b[8] __attribute__((aligned));
 short int c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 16000+i;
     b[i] = 16763+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	26.191
GCC 4.0 - not optimized	23.198
GCC 4.1 - not optimized	22.925
ICC 8.1 - not optimized	25.692

GCC 4.0	11.512
GCC 4.1	11.363
ICC 8.1	11.214

GCC SIMD	1.098
ICC SIMD	1.29

GCC 4.0 behavior	-O2 optimization, no vectorization
GCC 4.1 behavior	-O2 optimization, vectorization
ICC behavior	-O2 optimization, vectorization

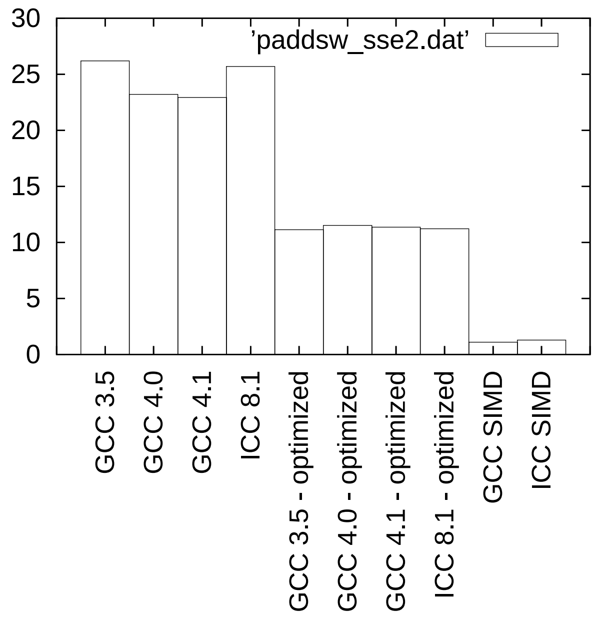

Figure 26: Benchmarks for paddsw - SSE2 version

4.5 paddusb - MMX (64 bits registers) version

4.5.1 C code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i, k;

  for(i=0; i<8; i++)
    {
      k = a[i] + b[i];
      if(k > 255)
        {
          c[i] = 255;
        }
        else
        {
          c[i] = k;
        }
    }
}
```

4.5.2 GIMPLE code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i, k;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 + T2;
  if(k > 255)
```

```
    T3 = 255;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.5.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  *(__m64 *) c = _mm_adds_pu8(*(__m64*) a,  *(__m64*) b);
}
```

4.5.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl $1, %ecx
movl 8(%ebp), %eax	subl $8, %esp	movl %esp, %ebp
movl 12(%ebp), %edx	movl $0, -4(%ebp)	pushl %edi
movq (%eax), %mm0	.L2:	movl 8(%ebp), %edi
movl 16(%ebp), %eax	cmpl $7, -4(%ebp)	pushl %esi
paddusb (%edx), %mm0	jle .L5	movl 12(%ebp), %esi
movq %mm0, (%eax)	jmp .L1	pushl %ebx
popl %ebp	.L5:	movl 16(%ebp), %ebx
ret	movl -4(%ebp), %eax	jmp .L2
	addl 8(%ebp), %eax	.p2align 4,,7
	movzbl (%eax), %edx	.L12:
	movl -4(%ebp), %eax	movb $-1, -1(%ecx,%ebx)
	addl 12(%ebp), %eax	incl %ecx
	movzbl (%eax), %eax	cmpl $9, %ecx
	leal (%eax,%edx), %eax	je .L11
	movl %eax, -8(%ebp)	.L2:
	cmpl $255, -8(%ebp)	movzbl -1(%edi,%ecx), %edx
	jle .L6	movzbl -1(%ecx,%esi), %eax
	movl -4(%ebp), %eax	leal (%edx,%eax), %eax
	addl 16(%ebp), %eax	cmpl $255, %eax
	movb $-1, (%eax)	jg .L12
	jmp .L4	movb %al, -1(%ecx,%ebx)
	.L6:	incl %ecx
	movl -4(%ebp), %eax	cmpl $9, %ecx
	movl 16(%ebp), %edx	jne .L2
	addl %eax, %edx	.L11:
	movzbl -8(%ebp), %eax	popl %ebx
	movb %al, (%edx)	popl %esi
	.L4:	popl %edi
	leal -4(%ebp), %eax	popl %ebp
	incl (%eax)	ret
	jmp .L2	
	.L1:	

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`leave` `ret`	

4.5.5 Benchmark

```
 unsigned char a[8] __attribute__((aligned));
 unsigned char b[8] __attribute__((aligned));
 unsigned char c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 120+i;
     b[i] = 120+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
 test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	13.595
GCC 4.0 - not optimized	17.054
GCC 4.1 - not optimized	17.477
ICC 8.1 - not optimized	13.922

GCC 4.0	5.29
GCC 4.1	5.655
ICC 8.1	5.431

GCC SIMD	0.886
ICC SIMD	1.609

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

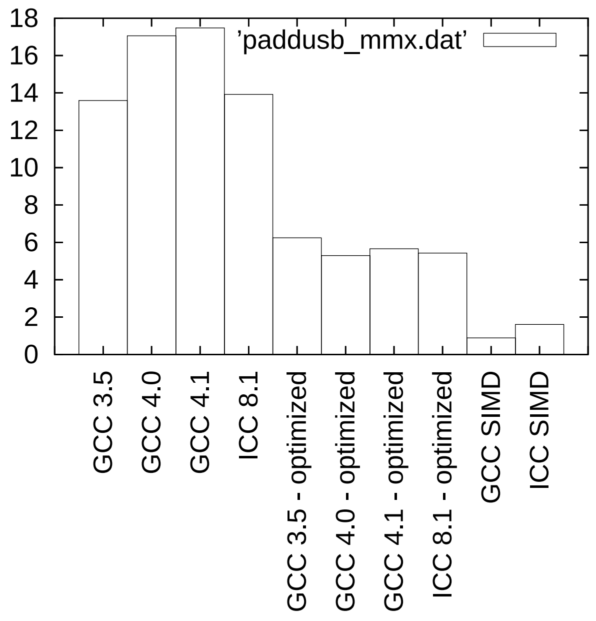

Figure 27: Benchmarks for paddusb - MMX version

4.6 paddusb - SSE2 (128 bits registers) version

4.6.1 C code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i, k;

  for(i=0; i<16; i++)
    {
      k = a[i] + b[i];
      if(k > 255)
        {
          c[i] = 255;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.6.2 GIMPLE code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i, k;
  loop_label::
  if(i >= 16)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 + T2;
  if(k > 255)
```

```
    T3 = 255;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.6.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  *(__m128i *) c = _mm_adds_upi8(*(__m128i*) a,  *(__m128i*) b);
}
```

4.6.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`pushl %edi`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`xorl %edi, %edi`
`movl 8(%ebp), %eax`	`.L2:`	`pushl %esi`
`paddb (%eax), %xmm0`	`cmpl $15, -4(%ebp)`	`pushl %ebx`
`movl 16(%ebp), %eax`	`jle .L5`	`subl $12, %esp`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`movl 16(%ebp), %ebx`
`popl %ebp`	`.L5:`	`negl %ebx`
`ret`	`movl -4(%ebp), %eax`	`andl $15, %ebx`
	`movl 16(%ebp), %ecx`	`cmpl $0, %ebx`
	`addl %eax, %ecx`	`jbe .L20`
	`movl -4(%ebp), %eax`	`.p2align 4,,15`
	`movl 8(%ebp), %edx`	`.L11:`
	`addl %eax, %edx`	`movl 12(%ebp), %edx`
	`movl -4(%ebp), %eax`	`movzbl (%edx,%edi), %eax`
	`addl 12(%ebp), %eax`	`movl 8(%ebp), %edx`
	`movzbl (%eax), %eax`	`addb (%edx,%edi), %al`
	`addb (%edx), %al`	`movl 16(%ebp), %edx`
	`movb %al, (%ecx)`	`movb %al, (%edx,%edi)`
	`leal -4(%ebp), %eax`	`incl %edi`
	`incl (%eax)`	`cmpl %edi, %ebx`
	`jmp .L2`	`ja .L11`
	`.L1:`	`movl $16, -24(%ebp)`
	`leave`	`subl %edi, -24(%ebp)`
	`ret`	`cmpl $16, %ebx`
		`je .L13`
		`.L4:`
		`movl $16, -20(%ebp)`
		`subl %ebx, -20(%ebp)`
		`movl -20(%ebp), %esi`
		`shrl $4, %esi`
		`movl %esi, %eax`
		`sall $4, %eax`

SIMD intrinsics	-nooptim	-O2 and vectorizer

		`cmpl $0, %eax`
		`movl %eax, -16(%ebp)`
		`jbe .L7`
		`movl 8(%ebp), %ecx`
		`movl 12(%ebp), %edx`
		`movl 16(%ebp), %eax`
		`addl %ebx, %ecx`
		`addl %ebx, %edx`
		`addl %ebx, %eax`
		`xorl %ebx, %ebx`
		`.p2align 4,,15`
		`.L9:`
		`movdqu (%ecx), %xmm0`
		`movdqu (%edx), %xmm1`
		`incl %ebx`
		`paddb %xmm1, %xmm0`
		`addl $16, %ecx`
		`movdqa %xmm0, (%eax)`
		`addl $16, %edx`
		`addl $16, %eax`
		`cmpl %esi, %ebx`
		`jb .L9`
		`movl -16(%ebp), %eax`
		`subl %eax, -24(%ebp)`
		`addl %eax, %edi`
		`cmpl %eax, -20(%ebp)`
		`je .L13`
		`.L7:`
		`movl 16(%ebp), %ebx`
		`xorl %esi, %esi`
		`movl 12(%ebp), %ecx`
		`movl 8(%ebp), %edx`
		`addl %edi, %ebx`
		`addl %edi, %ecx`
		`addl %edi, %edx`
		`.p2align 4,,15`
		`.L12:`
		`movzbl (%ecx), %eax`
		`incl %esi`
		`incl %ecx`
		`addb (%edx), %al`
		`incl %edx`
		`movb %al, (%ebx)`
		`incl %ebx`
		`cmpl %esi, -24(%ebp)`
		`jne .L12`
		`.L13:`
		`addl $12, %esp`
		`popl %ebx`
		`popl %esi`
		`popl %edi`
		`popl %ebp`

SIMD intrinsics	-nooptim	-O2 and vectorizer
		`ret`
		`.L20:`
		`movl $16, -24(%ebp)`
		`jmp .L4`

4.6.5 Benchmark

```
 unsigned char a[16] __attribute__((aligned));
 unsigned char b[16] __attribute__((aligned));
 unsigned char c[16] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = 120+i;
     b[i] = 120+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
 test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	29.625
GCC 4.0 - not optimized	34.819
GCC 4.1 - not optimized	34.174
ICC 8.1 - not optimized	29.159

GCC 4.0	10.044
GCC 4.1	9.682
ICC 8.1	2.317

GCC SIMD	0.702
ICC SIMD	1.389

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `paddusb`

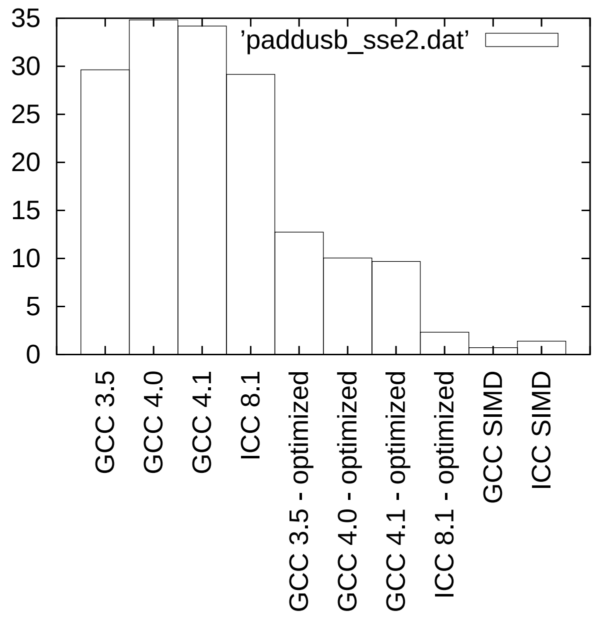

Figure 28: Benchmarks for paddusb - SSE2 version

4.7 paddusw - MMX (64 bits registers) version

4.7.1 C code

```
void test_loop_c(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  int i, k;

  for(i=0; i<4; i++)
    {
      k = a[i] + b[i];
      if(k > 65535)
        {
          c[i] = 65535;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.7.2 GIMPLE code

```
void test_loop_c(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  int i, k;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 + T2;
  if(k > 65535)
```

```
    T3 = 65535;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.7.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  *(__m64 *) c = _mm_adds_pu16(*(__m64*) a,  *(__m64*) b);
}
```

4.7.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl %esp, %ebp
movl 8(%ebp), %eax	subl $8, %esp	pushl %edi
movl 12(%ebp), %edx	movl $0, -4(%ebp)	movl 12(%ebp), %edi
movq (%eax), %mm0	.L2:	pushl %esi
movl 16(%ebp), %eax	cmpl $3, -4(%ebp)	movl 16(%ebp), %esi
paddusw (%edx), %mm0	jle .L5	pushl %ebx
movq %mm0, (%eax)	jmp .L1	movl $1, %ebx
popl %ebp	.L5:	jmp .L2
ret	movl -4(%ebp), %eax	.p2align 4,,7
	leal (%eax,%eax), %edx	.L12:
	movl 8(%ebp), %eax	incl %ebx
	movzwl (%eax,%edx), %ecx	cmpl $5, %ebx
	movl -4(%ebp), %eax	movw $-1, -2(%ecx,%esi)
	leal (%eax,%eax), %edx	je .L11
	movl 12(%ebp), %eax	.L2:
	movzwl (%eax,%edx), %eax	movl 8(%ebp), %eax
	leal (%eax,%ecx), %eax	leal (%ebx,%ebx), %ecx
	movl %eax, -8(%ebp)	movzwl -2(%eax,%ecx), %edx
	cmpl $65535, -8(%ebp)	movzwl -2(%ecx,%edi), %eax
	jle .L6	leal (%edx,%eax), %eax
	movl -4(%ebp), %eax	cmpl $65535, %eax
	leal (%eax,%eax), %edx	jg .L12
	movl 16(%ebp), %eax	incl %ebx
	movw $-1, (%eax,%edx)	cmpl $5, %ebx
	jmp .L4	movw %ax, -2(%ecx,%esi)
	.L6:	jne .L2
	movl -4(%ebp), %eax	.L11:
	leal (%eax,%eax), %ecx	popl %ebx
	movl 16(%ebp), %edx	popl %esi
	movl -8(%ebp), %eax	popl %edi
	movw %ax, (%edx,%ecx)	popl %ebp
	.L4:	ret
	leal -4(%ebp), %eax	

SIMD intrinsics	-nooptim	-O2 and vectorizer

	`incl (%eax)`
	`jmp .L2`
	`.L1:`
	`leave`
	`ret`	

4.7.5 Benchmark

```
 unsigned short int a[4] __attribute__((aligned));
 unsigned short int b[4] __attribute__((aligned));
 unsigned short int c[4] __attribute__((aligned));
 int i;

 for(i = 0; i<4; i++)
   {
     a[i] = 16000+i;
     b[i] = 16763+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	7.273
GCC 4.0 - not optimized	7.311
GCC 4.1 - not optimized	7.725
ICC 8.1 - not optimized	6.098

GCC 4.0	3.667
GCC 4.1	3.788
ICC 8.1	2.465

GCC SIMD	0.832
ICC SIMD	1.221

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

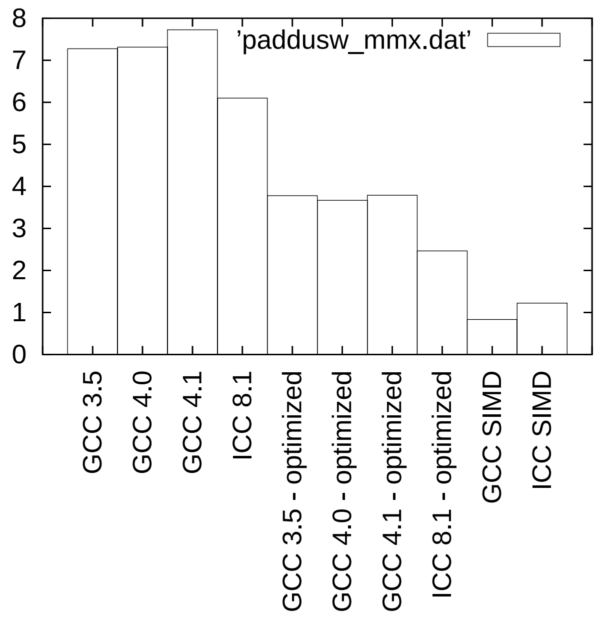

Figure 29: Benchmarks for paddusw - MMX version

4.8 paddusw - SSE2 (128 bits registers) version

4.8.1 C code

```
void test_loop_c(unsigned short int a[8], unsigned short int b[8], unsigned short int c[8])
{
  int i, k;

  for(i=0; i<8; i++)
    {
      k = a[i] + b[i];
      if(k > 65535)
        {
          c[i] = 65535;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.8.2 GIMPLE code

```
void test_loop_c(unsigned short int a[8], unsigned short int b[8], unsigned short int c[8])
{
  int i, k;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 + T2;
  if(k > 65535)
```

```
    T3 = 65535;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.8.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned short int a[8], unsigned short int b[8], unsigned short int c[8])
{
  *(__m128i *) c = _mm_adds_epu16(*(__m128i*) a,  *(__m128i*) b);
}
```

4.8.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl %esp, %ebp
movl 12(%ebp), %eax	subl $8, %esp	pushl %edi
movdqa (%eax), %xmm0	movl $0, -4(%ebp)	movl 12(%ebp), %edi
movl 8(%ebp), %eax	.L2:	pushl %esi
paddusw (%eax), %xmm0	cmpl $7, -4(%ebp)	movl 16(%ebp), %esi
movl 16(%ebp), %eax	jle .L5	pushl %ebx
movdqa %xmm0, (%eax)	jmp .L1	movl $1, %ebx
popl %ebp	.L5:	jmp .L2
ret	movl -4(%ebp), %eax	.p2align 4,,7
	leal (%eax,%eax), %edx	.L12:
	movl 8(%ebp), %eax	incl %ebx
	movzwl (%eax,%edx), %ecx	cmpl $9, %ebx
	movl -4(%ebp), %eax	movw $-1, -2(%ecx,%esi)
	leal (%eax,%eax), %edx	je .L11
	movl 12(%ebp), %eax	.L2:
	movzwl (%eax,%edx), %eax	movl 8(%ebp), %eax
	leal (%eax,%ecx), %eax	leal (%ebx,%ebx), %ecx
	movl %eax, -8(%ebp)	movzwl -2(%eax,%ecx), %edx
	cmpl $65535, -8(%ebp)	movzwl -2(%ecx,%edi), %eax
	jle .L6	leal (%edx,%eax), %eax
	movl -4(%ebp), %eax	cmpl $65535, %eax
	leal (%eax,%eax), %edx	jg .L12
	movl 16(%ebp), %eax	incl %ebx
	movw $-1, (%eax,%edx)	cmpl $9, %ebx
	jmp .L4	movw %ax, -2(%ecx,%esi)
	.L6:	jne .L2
	movl -4(%ebp), %eax	.L11:
	leal (%eax,%eax), %ecx	popl %ebx
	movl 16(%ebp), %edx	popl %esi
	movl -8(%ebp), %eax	popl %edi
	movw %ax, (%edx,%ecx)	popl %ebp
	.L4:	ret
	leal -4(%ebp), %eax	

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

4.8.5 Benchmark

```
 unsigned short int a[8] __attribute__((aligned));
 unsigned short int b[8] __attribute__((aligned));
 unsigned short int c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 32000+i;
     b[i] = 33530+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	13.163
GCC 4.0 - not optimized	13.367
GCC 4.1 - not optimized	13.878
ICC 8.1 - not optimized	12.599

GCC 4.0	5.951
GCC 4.1	5.755
ICC 8.1	2.332

GCC SIMD	0.971
ICC SIMD	1.591

GCC 4.0 behavior	O2 optimization, no vectorization
GCC 4.1 behavior	O2 optimization, no vectorization
ICC behavior	vectorization with `paddusw`

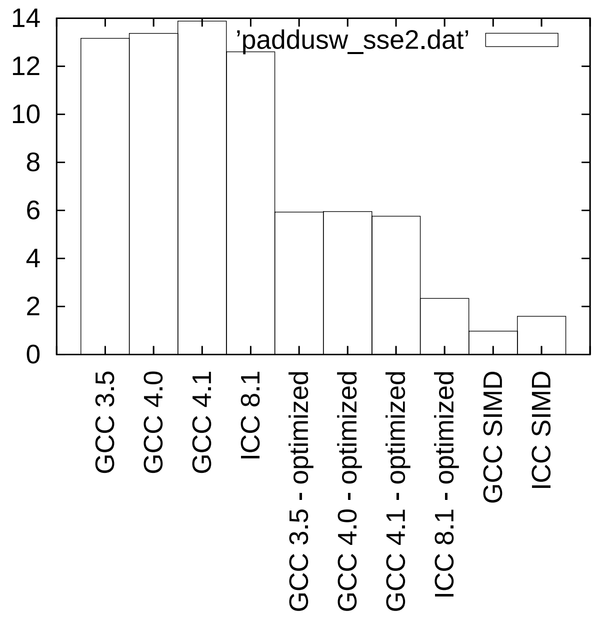

Figure 30: Benchmarks for paddusw - SSE2 version

4.9 psubsb - MMX (64 bits registers) version

4.9.1 C code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i, k;

  for(i=0; i<8; i++)
    {
      k = a[i] - b[i];
      if(k > 127)
        {
          c[i] = 127;
        }
      else if(k<-128)
        {
          c[i] = -128;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.9.2 GIMPLE code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i, k;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 - T2;
  if(k > 127)
    T3 = 127;
  else if(k<-128)
    T3 = -128;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.9.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[8], char b[8], char c[8])
{
  *(__m64 *) c = _mm_subs_pi8(*(__m64*) a,  *(__m64*) b);
}
```

4.9.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %ecx`
`movl 8(%ebp), %eax`	`subl $8, %esp`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`pushl %edi`
`movl 12(%ebp), %eax`	`.L2:`	`movl 8(%ebp), %edi`
`psubsb (%eax), %mm0`	`cmpl $7, -4(%ebp)`	`pushl %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 12(%ebp), %esi`
`movq %mm0, (%eax)`	`jmp .L1`	`pushl %ebx`
`popl %ebp`	`.L5:`	`movl 16(%ebp), %ebx`
`ret`	`movl -4(%ebp), %eax`	`jmp .L2`
	`addl 8(%ebp), %eax`	`.p2align 4,,7`
	`movsbl (%eax),%edx`	`.L14:`
	`movl -4(%ebp), %eax`	`movb $127, -1(%ebx,%ecx)`
	`addl 12(%ebp), %eax`	`.L5:`
	`movsbl (%eax),%eax`	`incl %ecx`
	`subl %eax, %edx`	`cmpl $9, %ecx`
	`movl %edx, %eax`	`je .L13`
	`movl %eax, -8(%ebp)`	`.L2:`
	`cmpl $127, -8(%ebp)`	`movsbl -1(%edi,%ecx),%edx`
	`jle .L6`	`movsbl -1(%esi,%ecx),%eax`
	`movl -4(%ebp), %eax`	`subl %eax, %edx`
	`addl 16(%ebp), %eax`	`cmpl $127, %edx`
	`movb $127, (%eax)`	`jg .L14`
	`jmp .L4`	`cmpl $-128, %edx`
	`.L6:`	`jge .L6`
	`cmpl $-128, -8(%ebp)`	`movb $-128, -1(%ebx,%ecx)`
	`jge .L8`	`incl %ecx`
	`movl -4(%ebp), %eax`	`cmpl $9, %ecx`

```
SIMD intrinsics             -nooptim                     -O2 and vectorizer
                             addl 16(%ebp), %eax          jne .L2
                             movb $-128, (%eax)           .p2align 4,,15
                             jmp .L4                     .L13:
                            .L8:                          popl %ebx
                             movl -4(%ebp), %eax          popl %esi
                             movl 16(%ebp), %edx          popl %edi
                             addl %eax, %edx              popl %ebp
                             movzbl -8(%ebp), %eax        ret
                             movb %al, (%edx)             .p2align 4,,7
                            .L4:                         .L6:
                             leal -4(%ebp), %eax          movb %dl, -1(%ebx,%ecx)
                             incl (%eax)                  jmp .L5
                             jmp .L2
                            .L1:
                             leave
                             ret
```

4.9.5 Benchmark

```
  char a[8] __attribute__((aligned));
  char b[8] __attribute__((aligned));
  char c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = 60+i;
      b[i] = -60-2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	21.63
GCC 4.0 - not optimized	21.847
GCC 4.1 - not optimized	22.841
ICC 8.1 - not optimized	17.739
GCC 4.0	9.779
GCC 4.1	9.545
ICC 8.1	8.569
GCC SIMD	1.51
ICC SIMD	1.725

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

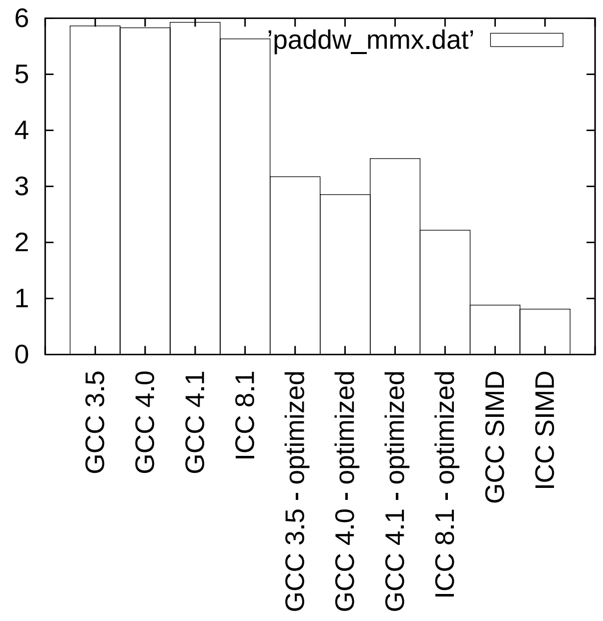

Figure 31: Benchmarks for psubsb - MMX version

4.10 psubsb - SSE2 (128 bits registers) version

4.10.1 C code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i, k;

  for(i=0; i<16; i++)
    {
      k = a[i] - b[i];
      if(k > 127)
        {
          c[i] = 127;
        }
      else if(k<-128)
        {
          c[i] = -128;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.10.2 GIMPLE code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i, k;
  loop_label::
  if(i >= 16)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 - T2;
  if(k > 127)
    T3 = 127;
  else if(k<-128)
    T3 = -128;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.10.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[16], char b[16], char c[16])
{
  *(__m128i *) c = _mm_subs_epi8(*(__m128i*) a,  *(__m128i*) b);
}
```

4.10.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %ecx`
`movl 8(%ebp), %eax`	`subl $8, %esp`	`movl %esp, %ebp`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`pushl %edi`
`movl 12(%ebp), %eax`	`.L2:`	`movl 8(%ebp), %edi`
`psubsb (%eax), %xmm0`	`cmpl $15, -4(%ebp)`	`pushl %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 12(%ebp), %esi`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`pushl %ebx`
`popl %ebp`	`.L5:`	`movl 16(%ebp), %ebx`
`ret`	`movl -4(%ebp), %eax`	`jmp .L2`
	`addl 8(%ebp), %eax`	`.p2align 4,,7`
	`movsbl (%eax),%edx`	`.L14:`
	`movl -4(%ebp), %eax`	`movb $127, -1(%ebx,%ecx)`
	`addl 12(%ebp), %eax`	`.L5:`
	`movsbl (%eax),%eax`	`incl %ecx`
	`subl %eax, %edx`	`cmpl $17, %ecx`
	`movl %edx, %eax`	`je .L13`
	`movl %eax, -8(%ebp)`	`.L2:`
	`cmpl $127, -8(%ebp)`	`movsbl -1(%edi,%ecx),%edx`
	`jle .L6`	`movsbl -1(%esi,%ecx),%eax`
	`movl -4(%ebp), %eax`	`subl %eax, %edx`
	`addl 16(%ebp), %eax`	`cmpl $127, %edx`
	`movb $127, (%eax)`	`jg .L14`
	`jmp .L4`	`cmpl $-128, %edx`
	`.L6:`	`jge .L6`
	`cmpl $-128, -8(%ebp)`	`movb $-128, -1(%ebx,%ecx)`
	`jge .L8`	`incl %ecx`
	`movl -4(%ebp), %eax`	`cmpl $17, %ecx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`addl 16(%ebp), %eax`	`jne .L2`
	`movb $-128, (%eax)`	`.p2align 4,,15`
	`jmp .L4`	`.L13:`
	`.L8:`	`popl %ebx`
	`movl -4(%ebp), %eax`	`popl %esi`
	`movl 16(%ebp), %edx`	`popl %edi`
	`addl %eax, %edx`	`popl %ebp`
	`movzbl -8(%ebp), %eax`	`ret`
	`movb %al, (%edx)`	`.p2align 4,,7`
	`.L4:`	`.L6:`
	`leal -4(%ebp), %eax`	`movb %dl, -1(%ebx,%ecx)`
	`incl (%eax)`	`jmp .L5`
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

4.10.5 Benchmark

```
 char a[16] __attribute__((aligned));
 char b[16] __attribute__((aligned));
 char c[16] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = 60+i;
     b[i] = -60-2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	42.629
GCC 4.0 - not optimized	38.361
GCC 4.1 - not optimized	39.269
ICC 8.1 - not optimized	35.431

GCC 4.0	16.424
GCC 4.1	16.569
ICC 8.1	3.291

GCC SIMD	1.67
ICC SIMD	2.465

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	`psubsb` vectorization

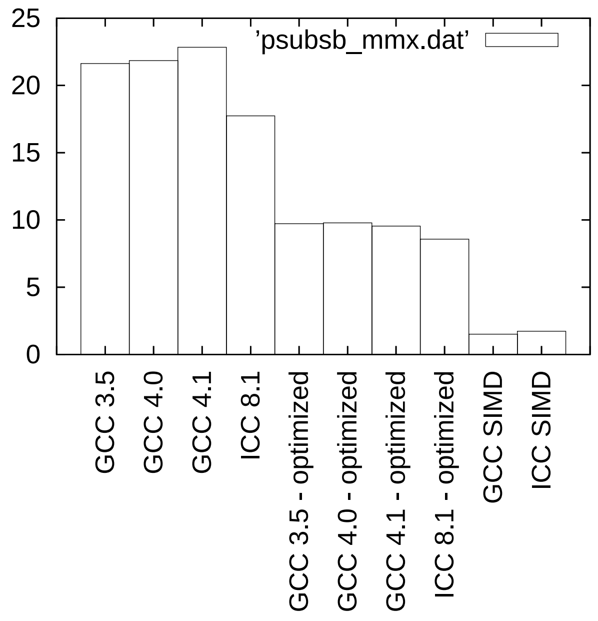

Figure 32: Benchmarks for psubsb - SSE2 version

4.11 psubsw - MMX (64 bits registers) version

4.11.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i, k;

  for(i=0; i<4; i++)
    {
      k = a[i] - b[i];
      if(k > 32767)
        {
          c[i] = 32767;
        }
      else if(k<-32768)
        {
          c[i] = -32768;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.11.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i, k;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 - T2;
  if(k > 32767)
    T3 = 32767;
  else if(k<-32768)
    T3 = -32768;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.11.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_subs_pi16(*(__m64*) a,  *(__m64*) b);
}
```

4.11.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 8(%ebp), %eax`	`subl $16, %esp`	`pushl %edi`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`movl 12(%ebp), %edi`
`movl 12(%ebp), %eax`	`.L2:`	`pushl %esi`
`psubsw (%eax), %xmm0`	`cmpl $7, -4(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`pushl %ebx`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`movl $1, %ebx`
`popl %ebp`	`.L5:`	`jmp .L2`
`ret`	`movl -4(%ebp), %eax`	`.p2align 4,,7`
	`leal (%eax,%eax), %edx`	`.L18:`
	`movl 8(%ebp), %eax`	`jle .L16`
	`movswl (%eax,%edx),%ecx`	`.L5:`
	`movl -4(%ebp), %eax`	`movw $32767, -2(%esi,%ecx)`
	`leal (%eax,%eax), %edx`	`.L6:`
	`movl 12(%ebp), %eax`	`incl %ebx`
	`movswl (%eax,%edx),%eax`	`cmpl $9, %ebx`
	`subl %eax, %ecx`	`je .L17`
	`movl %ecx, %eax`	`.L2:`
	`cltd`	`movl 8(%ebp), %edx`
	`movl %eax, -16(%ebp)`	`leal (%ebx,%ebx), %ecx`
	`movl %edx, -12(%ebp)`	`movswl -2(%edx,%ecx),%eax`
	`cmpl $0, -12(%ebp)`	`movswl -2(%edi,%ecx),%edx`
	`js .L6`	`subl %edx, %eax`
	`cmpl $0, -12(%ebp)`	`cltd`
	`jg .L7`	`cmpl $0, %edx`
	`cmpl $32767, -16(%ebp)`	`jge .L18`
	`jbe .L6`	`incl %edx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`.L7:`	`jle .L19`
	`movl -4(%ebp), %eax`	`.L7:`
	`leal (%eax,%eax), %edx`	`incl %ebx`
	`movl 16(%ebp), %eax`	`cmpl $9, %ebx`
	`movw $32767, (%eax,%edx)`	`movw %ax, -2(%esi,%ecx)`
	`jmp .L4`	`jne .L2`
	`.L6:`	`.p2align 4,,15`
	`cmpl $-1, -12(%ebp)`	`.L17:`
	`jg .L9`	`popl %ebx`
	`cmpl $-1, -12(%ebp)`	`popl %esi`
	`jl .L10`	`popl %edi`
	`cmpl $-32768, -16(%ebp)`	`popl %ebp`
	`jae .L9`	`ret`
	`.L10:`	`.L19:`
	`movl -4(%ebp), %eax`	`.p2align 4,,2`
	`leal (%eax,%eax), %edx`	`jl .L9`
	`movl 16(%ebp), %eax`	`cmpl $-32768, %eax`
	`movw $-32768, (%eax,%edx)`	`.p2align 4,,6`
	`jmp .L4`	`jae .L7`
	`.L9:`	`.L9:`
	`movl -4(%ebp), %eax`	`movw`
		`$-32768, -2(%esi,%ecx)`
	`leal (%eax,%eax), %ecx`	`.p2align 4,,6`
	`movl 16(%ebp), %edx`	`jmp .L6`
	`movl -16(%ebp), %eax`	`.p2align 4,,7`
	`movw %ax, (%edx,%ecx)`	`.L16:`
	`.L4:`	`cmpl $32767, %eax`
	`leal -4(%ebp), %eax`	`.p2align 4,,4`
	`incl (%eax)`	`ja .L5`
	`jmp .L2`	`incl %edx`
	`.L1:`	`.p2align 4,,5`
	`leave`	`jg .L7`
	`ret`	`.p2align 4,,8`
		`jmp .L19`

4.11.5 Benchmark

```
 short int a[4] __attribute__((aligned));
 short int b[4] __attribute__((aligned));
 short int c[4] __attribute__((aligned));
 int i;

 for(i = 0; i<4; i++)
   {
     a[i] = 16000+i;
     b[i] = -16763-2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	14.924
GCC 4.0 - not optimized	15.886
GCC 4.1 - not optimized	16.247
ICC 8.1 - not optimized	16.626

GCC 4.0	8.71
GCC 4.1	8.773
ICC 8.1	7.302

GCC SIMD	1.431
ICC SIMD	1.696

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

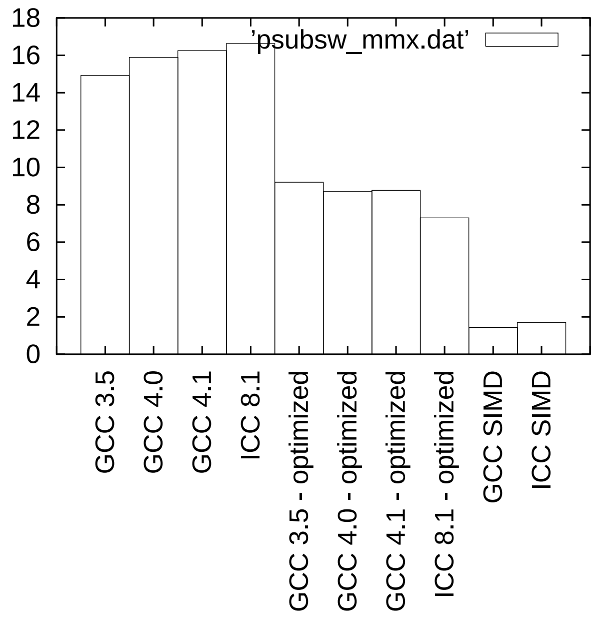

Figure 33: Benchmarks for psubsw - MMX version

4.12 psubsw - SSE2 (128 bits registers) version

4.12.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i, k;

  for(i=0; i<8; i++)
    {
      k = a[i] - b[i];
      if(k > 32767)
        {
          c[i] = 32767;
        }
      else if(k<-32768)
        {
          c[i] = -32768;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.12.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i, k;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 - T2;
  if(k > 32767)
    T3 = 32767;
  else if(k<-32768)
    T3 = -32768;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.12.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_subs_epi16(*(__m128i*) a,  *(__m128i*) b);
}
```

4.12.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 8(%ebp), %eax`	`subl $16, %esp`	`pushl %edi`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`movl 12(%ebp), %edi`
`movl 12(%ebp), %eax`	`.L2:`	`pushl %esi`
`psubsw (%eax), %xmm0`	`cmpl $7, -4(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`pushl %ebx`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`movl $1, %ebx`
`popl %ebp`	`.L5:`	`jmp .L2`
`ret`	`movl -4(%ebp), %eax`	`.p2align 4,,7`
	`leal (%eax,%eax), %edx`	`.L18:`
	`movl 8(%ebp), %eax`	`jle .L16`
	`movswl (%eax,%edx),%ecx`	`.L5:`
	`movl -4(%ebp), %eax`	`movw $32767, -2(%esi,%ecx)`
	`leal (%eax,%eax), %edx`	`.L6:`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`movl 12(%ebp), %eax`	`incl %ebx`
	`movswl (%eax,%edx),%eax`	`cmpl $9, %ebx`
	`subl %eax, %ecx`	`je .L17`
	`movl %ecx, %eax`	`.L2:`
	`cltd`	`movl 8(%ebp), %edx`
	`movl %eax, -16(%ebp)`	`leal (%ebx,%ebx), %ecx`
	`movl %edx, -12(%ebp)`	`movswl -2(%edx,%ecx),%eax`
	`cmpl $0, -12(%ebp)`	`movswl -2(%edi,%ecx),%edx`
	`js .L6`	`subl %edx, %eax`
	`cmpl $0, -12(%ebp)`	`cltd`
	`jg .L7`	`cmpl $0, %edx`
	`cmpl $32767, -16(%ebp)`	`jge .L18`
	`jbe .L6`	`incl %edx`
	`.L7:`	`jle .L19`
	`movl -4(%ebp), %eax`	`.L7:`
	`leal (%eax,%eax), %edx`	`incl %ebx`
	`movl 16(%ebp), %eax`	`cmpl $9, %ebx`
	`movw $32767, (%eax,%edx)`	`movw %ax, -2(%esi,%ecx)`
	`jmp .L4`	`jne .L2`
	`.L6:`	`.p2align 4,,15`
	`cmpl $-1, -12(%ebp)`	`.L17:`
	`jg .L9`	`popl %ebx`
	`cmpl $-1, -12(%ebp)`	`popl %esi`
	`jl .L10`	`popl %edi`
	`cmpl $-32768, -16(%ebp)`	`popl %ebp`
	`jae .L9`	`ret`
	`.L10:`	`.L19:`
	`movl -4(%ebp), %eax`	`.p2align 4,,2`
	`leal (%eax,%eax), %edx`	`jl .L9`
	`movl 16(%ebp), %eax`	`cmpl $-32768, %eax`
	`movw $-32768, (%eax,%edx)`	`.p2align 4,,6`
	`jmp .L4`	`jae .L7`
	`.L9:`	`.L9:`
	`movl -4(%ebp), %eax`	`movw`
		`$-32768, -2(%esi,%ecx)`
	`leal (%eax,%eax), %ecx`	`.p2align 4,,6`
	`movl 16(%ebp), %edx`	`jmp .L6`
	`movl -16(%ebp), %eax`	`.p2align 4,,7`
	`movw %ax, (%edx,%ecx)`	`.L16:`
	`.L4:`	`cmpl $32767, %eax`
	`leal -4(%ebp), %eax`	`.p2align 4,,4`
	`incl (%eax)`	`ja .L5`
	`jmp .L2`	`incl %edx`
	`.L1:`	`.p2align 4,,5`
	`leave`	`jg .L7`
	`ret`	`.p2align 4,,8`
		`jmp .L19`

4.12.5 Benchmark

```
short int a[8] __attribute__((aligned));
short int b[8] __attribute__((aligned));
short int c[8] __attribute__((aligned));
int i;

for(i = 0; i<8; i++)
  {
    a[i] = -16000-i;
```

```
      b[i] = 16763+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	31.387
GCC 4.0 - not optimized	32.422
GCC 4.1 - not optimized	29.993
ICC 8.1 - not optimized	25.893

GCC 4.0	16.76
GCC 4.1	12.558
ICC 8.1	11.345

GCC SIMD	1.502
ICC SIMD	1.647

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

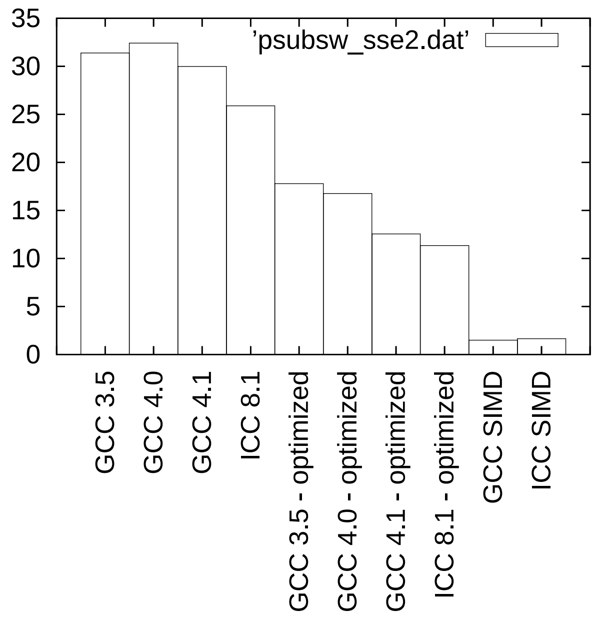

Figure 34: Benchmarks for psubsw - SSE2 version

4.13 psubusb - MMX (64 bits registers) version

4.13.1 C code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i, k;
  for(i=0; i<8; i++)
    {
      k = a[i] - b[i];
      if(k < 0)
        {
          c[i] = 0;
        }
        else
        {
          c[i] = k;
        }
    }
}
```

4.13.2 GIMPLE code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i, k;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 - T2;
  if(k < 0)
    T3 = 0;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.13.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  *(__m64 *) c = _mm_subs_pu8(*(__m64*) a,  *(__m64*) b);
}
```

4.13.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %ecx`
`movl 8(%ebp), %eax`	`subl $8, %esp`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`pushl %edi`
`movl 12(%ebp), %eax`	`.L2:`	`movl 8(%ebp), %edi`
`psubusb (%eax), %mm0`	`cmpl $7, -4(%ebp)`	`pushl %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 12(%ebp), %esi`
`movq %mm0, (%eax)`	`jmp .L1`	`pushl %ebx`
`popl %ebp`	`.L5:`	`movl 16(%ebp), %ebx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
`ret`	`movl -4(%ebp), %eax`	`jmp .L2`
	`addl 8(%ebp), %eax`	`.p2align 4,,7`
	`movzbl (%eax), %edx`	`.L3:`
	`movl -4(%ebp), %eax`	`movb %dl, -1(%ecx,%ebx)`
	`addl 12(%ebp), %eax`	`incl %ecx`
	`movzbl (%eax), %eax`	`cmpl $9, %ecx`
	`subl %eax, %edx`	`je .L11`
	`movl %edx, %eax`	`.L2:`
	`movl %eax, -8(%ebp)`	`movzbl -1(%edi,%ecx), %edx`
	`cmpl $0, -8(%ebp)`	`movzbl -1(%ecx,%esi), %eax`
	`jns .L6`	`subl %eax, %edx`
	`movl -4(%ebp), %eax`	`jns .L3`
	`addl 16(%ebp), %eax`	`movb $0, -1(%ecx,%ebx)`
	`movb $0, (%eax)`	`incl %ecx`
	`jmp .L4`	`cmpl $9, %ecx`
	`.L6:`	`jne .L2`
	`movl -4(%ebp), %eax`	`.p2align 4,,15`
	`movl 16(%ebp), %edx`	`.L11:`
	`addl %eax, %edx`	`popl %ebx`
	`movzbl -8(%ebp), %eax`	`popl %esi`
	`movb %al, (%edx)`	`popl %edi`
	`.L4:`	`popl %ebp`
	`leal -4(%ebp), %eax`	`ret`
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

4.13.5 Benchmark

```
 unsigned char a[8] __attribute__((aligned));
 unsigned char b[8] __attribute__((aligned));
 unsigned char c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 120;
     b[i] = 115+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
 test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	14.952
GCC 4.0 - not optimized	20.128
GCC 4.1 - not optimized	22.258
ICC 8.1 - not optimized	17.414

GCC 4.0	7.014
GCC 4.1	6.938
ICC 8.1	6.115

GCC SIMD	1.072
ICC SIMD	1.545

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

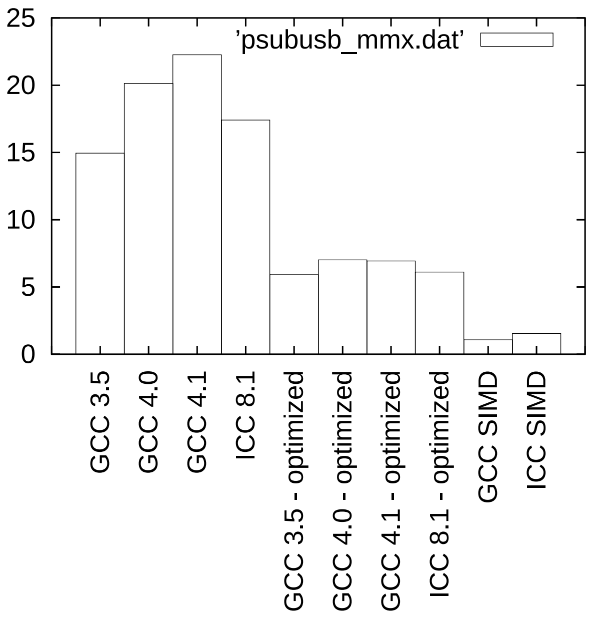

Figure 35: Benchmarks for psubusb - MMX version

4.14 psubusb - SSE2 (128 bits registers) version

4.14.1 C code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i, k;

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

4.14.2 GIMPLE code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i, k;
  loop_label::
  if(i >= 16)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 - T2;
  if(k < 0)
    T3 = 0;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.14.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  *(__m128i *) c = _mm_subs_upi8(*(__m128i*) a,  *(__m128i*) b);
}
```

4.14.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl $1, %ecx
movl 8(%ebp), %eax	subl $8, %esp	movl %esp, %ebp
movdqa (%eax), %xmm0	movl $0, -4(%ebp)	pushl %edi
movl 12(%ebp), %eax	.L2:	movl 8(%ebp), %edi
psubusb (%eax), %xmm0	cmpl $15, -4(%ebp)	pushl %esi
movl 16(%ebp), %eax	jle .L5	movl 12(%ebp), %esi
movdqa %xmm0, (%eax)	jmp .L1	pushl %ebx
popl %ebp	.L5:	movl 16(%ebp), %ebx
ret	movl -4(%ebp), %eax	jmp .L2
	addl 8(%ebp), %eax	.p2align 4,,7
	movzbl (%eax), %edx	.L3:
	movl -4(%ebp), %eax	movb %dl, -1(%ecx,%ebx)
	addl 12(%ebp), %eax	incl %ecx
	movzbl (%eax), %eax	cmpl $17, %ecx
	subl %eax, %edx	je .L11
	movl %edx, %eax	.L2:
	movl %eax, -8(%ebp)	movzbl -1(%edi,%ecx), %edx
	cmpl $0, -8(%ebp)	movzbl -1(%ecx,%esi), %eax
	jns .L6	subl %eax, %edx

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`movl -4(%ebp), %eax`	`jns .L3`
	`addl 16(%ebp), %eax`	`movb $0, -1(%ecx,%ebx)`
	`movb $0, (%eax)`	`incl %ecx`
	`jmp .L4`	`cmpl $17, %ecx`
	`.L6:`	`jne .L2`
	`movl -4(%ebp), %eax`	`.p2align 4,,15`
	`movl 16(%ebp), %edx`	`.L11:`
	`addl %eax, %edx`	`popl %ebx`
	`movzbl -8(%ebp), %eax`	`popl %esi`
	`movb %al, (%edx)`	`popl %edi`
	`.L4:`	`popl %ebp`
	`leal -4(%ebp), %eax`	`ret`
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

4.14.5 Benchmark

```
 unsigned char a[16] __attribute__((aligned));
 unsigned char b[16] __attribute__((aligned));
 unsigned char c[16] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = 120+i;
     b[i] = 120+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	43.083
GCC 4.0 - not optimized	43.884
GCC 4.1 - not optimized	46.142
ICC 8.1 - not optimized	38.024
GCC 4.0	13.761
GCC 4.1	13.847
ICC 8.1	3.139
GCC SIMD	1.357
ICC SIMD	2.226

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `psubusb`

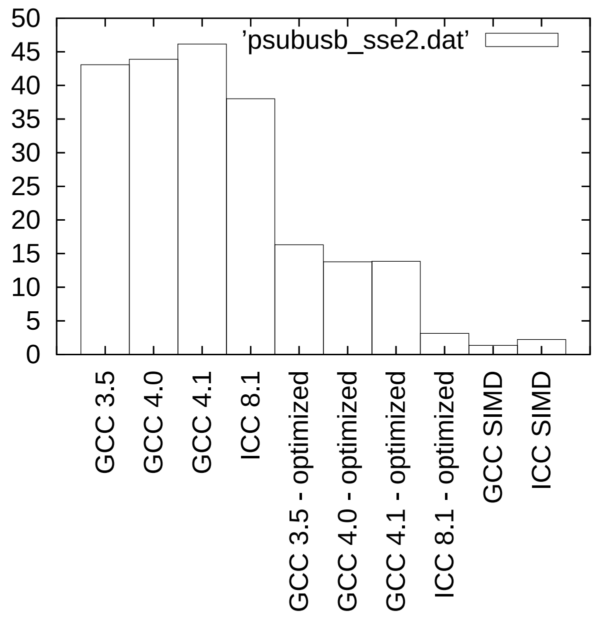

Figure 36: Benchmarks for psubusb - SSE2 version

4.15 psubusw - MMX (64 bits registers) version

4.15.1 C code

```
void test_loop_c(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  int i, k;

  for(i=0; i<4; i++)
    {
      k = a[i] - b[i];
      if(k < 0)
        {
          c[i] = 0;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.15.2 GIMPLE code

```
void test_loop_c(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  int i, k;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 - T2;
  if(k < 0)
```

```
    T3 = 0;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.15.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  *(__m64 *) c = _mm_subs_pu16(*(__m64*) a,  *(__m64*) b);
}
```

4.15.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 8(%ebp), %eax`	`subl $8, %esp`	`pushl %edi`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`movl 12(%ebp), %edi`
`movl 12(%ebp), %eax`	`.L2:`	`pushl %esi`
`psubusw (%eax), %mm0`	`cmpl $3, -4(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`pushl %ebx`
`movq %mm0, (%eax)`	`jmp .L1`	`movl $1, %ebx`
`popl %ebp`	`.L5:`	`jmp .L2`
`ret`	`movl -4(%ebp), %eax`	`.p2align 4,,7`
	`leal (%eax,%eax), %edx`	`.L12:`
	`movl 8(%ebp), %eax`	`incl %ebx`
	`movzwl (%eax,%edx), %ecx`	`cmpl $5, %ebx`
	`movl -4(%ebp), %eax`	`movw $-1, -2(%ecx,%esi)`
	`leal (%eax,%eax), %edx`	`je .L11`
	`movl 12(%ebp), %eax`	`.L2:`
	`movzwl (%eax,%edx), %eax`	`movl 8(%ebp), %eax`
	`subl %eax, %ecx`	`leal (%ebx,%ebx), %ecx`
	`movl %ecx, %eax`	`movzwl -2(%eax,%ecx), %edx`
	`movl %eax, -8(%ebp)`	`movzwl -2(%ecx,%edi), %eax`
	`cmpl $65535, -8(%ebp)`	`subl %eax, %edx`
	`jle .L6`	`cmpl $65535, %edx`
	`movl -4(%ebp), %eax`	`jg .L12`
	`leal (%eax,%eax), %edx`	`incl %ebx`
	`movl 16(%ebp), %eax`	`cmpl $5, %ebx`
	`movw $-1, (%eax,%edx)`	`movw %dx, -2(%ecx,%esi)`
	`jmp .L4`	`jne .L2`
	`.L6:`	`.L11:`
	`movl -4(%ebp), %eax`	`popl %ebx`
	`leal (%eax,%eax), %ecx`	`popl %esi`
	`movl 16(%ebp), %edx`	`popl %edi`
	`movl -8(%ebp), %eax`	`popl %ebp`
	`movw %ax, (%edx,%ecx)`	`ret`
	`.L4:`	

SIMD intrinsics	-nooptim	-O2 and vectorizer

	`leal -4(%ebp), %eax`
	`incl (%eax)`
	`jmp .L2`
	`.L1:`
	`leave`
	`ret`	

4.15.5 Benchmark

```
 unsigned short int a[4] __attribute__((aligned));
 unsigned short int b[4] __attribute__((aligned));
 unsigned short int c[4] __attribute__((aligned));
 int i;

 for(i = 0; i<4; i++)
   {
     a[i] = 16000+i;
     b[i] = 16763+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	10.253
GCC 4.0 - not optimized	13.586
GCC 4.1 - not optimized	13.288
ICC 8.1 - not optimized	20.226

GCC 4.0	7.785
GCC 4.1	7.502
ICC 8.1	4.779

GCC SIMD	1.626
ICC SIMD	2.38

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

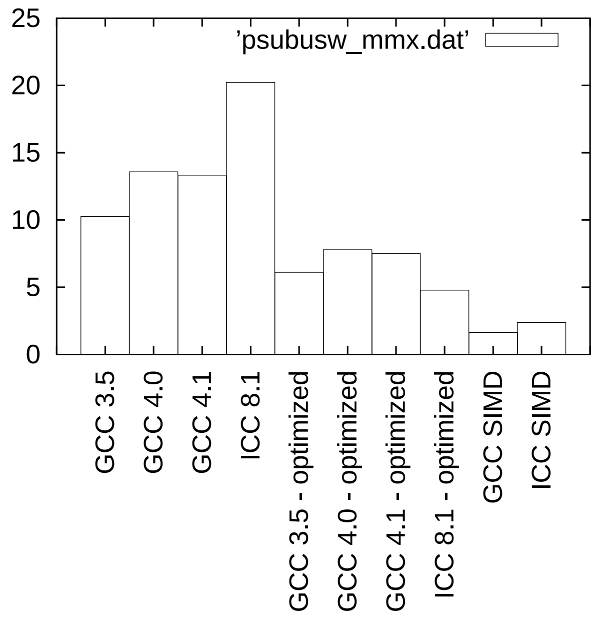

Figure 37: Benchmarks for psubusw - MMX version

4.16 psubusw - SSE2 (128 bits registers) version

4.16.1 C code

```
void test_loop_c(unsigned short int a[8], unsigned short int b[8], unsigned short int c[8])
{
  int i, k;

  for(i=0; i<8; i++)
    {
      k = a[i] - b[i];
      if(k < 0)
        {
          c[i] = 0;
        }
      else
        {
          c[i] = k;
        }
    }
}
```

4.16.2 GIMPLE code

```
void test_loop_c(unsigned short int a[8], unsigned short int b[8], unsigned short int c[8])
{
  int i, k;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  k = T1 + T2;
  if(k < 0)
```

```
    T3 = 0;
  else
    T3 = k;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

4.16.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned short int a[8], unsigned short int b[8], unsigned short int c[8])
{
  *(__m128i *) c = _mm_subs_epu16(*(__m128i*) a,  *(__m128i*) b);
}
```

4.16.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 8(%ebp), %eax`	`subl $8, %esp`	`pushl %edi`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`movl 12(%ebp), %edi`
`movl 12(%ebp), %eax`	`.L2:`	`pushl %esi`
`psubusw (%eax), %xmm0`	`cmpl $7, -4(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`pushl %ebx`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`movl $1, %ebx`
`popl %ebp`	`.L5:`	`jmp .L2`
`ret`	`movl -4(%ebp), %eax`	`.p2align 4,,7`
	`leal (%eax,%eax), %edx`	`.L3:`
	`movl 8(%ebp), %eax`	`incl %ebx`
	`movzwl (%eax,%edx), %ecx`	`cmpl $9, %ebx`
	`movl -4(%ebp), %eax`	`movw %dx, -2(%ecx,%esi)`
	`leal (%eax,%eax), %edx`	`je .L11`
	`movl 12(%ebp), %eax`	`.L2:`
	`movzwl (%eax,%edx), %eax`	`movl 8(%ebp), %eax`
	`subl %eax, %ecx`	`leal (%ebx,%ebx), %ecx`
	`movl %ecx, %eax`	`movzwl -2(%eax,%ecx), %edx`
	`movl %eax, -8(%ebp)`	`movzwl -2(%ecx,%edi), %eax`
	`cmpl $0, -8(%ebp)`	`subl %eax, %edx`
	`jns .L6`	`jns .L3`
	`movl -4(%ebp), %eax`	`incl %ebx`
	`leal (%eax,%eax), %edx`	`cmpl $9, %ebx`
	`movl 16(%ebp), %eax`	`movw $0, -2(%ecx,%esi)`
	`movw $0, (%eax,%edx)`	`jne .L2`
	`jmp .L4`	`.p2align 4,,15`
	`.L6:`	`.L11:`
	`movl -4(%ebp), %eax`	`popl %ebx`
	`leal (%eax,%eax), %ecx`	`popl %esi`
	`movl 16(%ebp), %edx`	`popl %edi`
	`movl -8(%ebp), %eax`	`popl %ebp`
	`movw %ax, (%edx,%ecx)`	`ret`
	`.L4:`	

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`leal -4(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

4.16.5 Benchmark

```
 unsigned short int a[8] __attribute__((aligned));
 unsigned short int b[8] __attribute__((aligned));
 unsigned short int c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 32000+i;
     b[i] = 33530+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	21.762
GCC 4.0 - not optimized	28.867
GCC 4.1 - not optimized	28.481
ICC 8.1 - not optimized	20.422
GCC 4.0	10.119
GCC 4.1	9.493
ICC 8.1	4.105
GCC SIMD	1.799
ICC SIMD	3.026

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `psubusw`

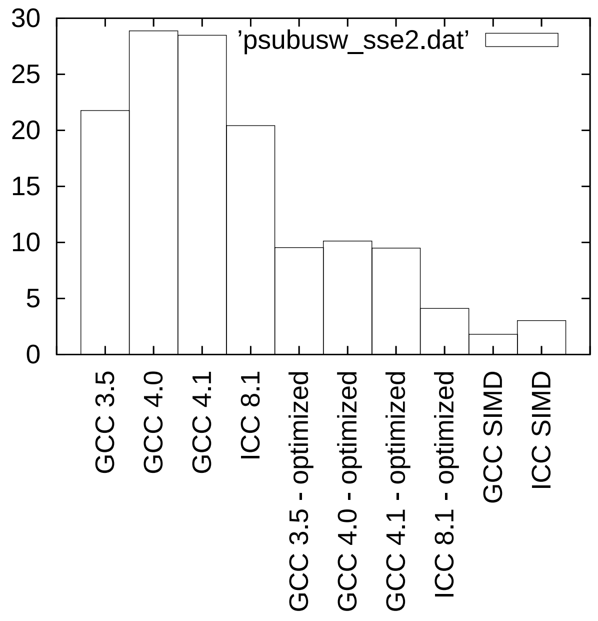

Figure 38: Benchmarks for psubusw - SSE2 version

5 Comparison operations

5.1 pcmpeqb - MMX (64 bits registers) version

5.1.1 C code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      if(a[i] == b[i])
        {
          c[i] = 0xFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.1.2 GIMPLE code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 == T2)
```

```
    T3 = 0xFF;
  else
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.1.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[8], char b[8], char c[8])
{
  *(__m64 *) c = _mm_cmpeq_pi8(*(__m64*) a,  *(__m64*) b);
}
```

5.1.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %edx`
`movl 8(%ebp), %eax`	`subl $4, %esp`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`pushl %esi`
`movl 12(%ebp), %eax`	`.L2:`	`movl 16(%ebp), %ecx`
`pcmpeqb (%eax), %mm0`	`cmpl $7, -4(%ebp)`	`pushl %ebx`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 8(%ebp), %esi`
`movq %mm0, (%eax)`	`jmp .L1`	`movl 12(%ebp), %ebx`
`popl %ebp`	`.L5:`	`.p2align 4,,15`
`ret`	`movl -4(%ebp), %eax`	`.L2:`
	`movl 8(%ebp), %ecx`	`movzbl -1(%edx,%ebx), %eax`
	`addl %eax, %ecx`	`cmpb %al, -1(%esi,%edx)`
	`movl -4(%ebp), %eax`	`setne %al`
	`movl 12(%ebp), %edx`	`decb %al`
	`addl %eax, %edx`	`movb %al, -1(%edx,%ecx)`
	`movzbl (%ecx), %eax`	`incl %edx`
	`cmpb (%edx), %al`	`cmpl $9, %edx`
	`jne .L6`	`jne .L2`
	`movl -4(%ebp), %eax`	`popl %ebx`
	`addl 16(%ebp), %eax`	`popl %esi`
	`movb $-1, (%eax)`	`popl %ebp`
	`jmp .L4`	`ret`
	`.L6:`	
	`movl -4(%ebp), %eax`	
	`addl 16(%ebp), %eax`	
	`movb $0, (%eax)`	
	`.L4:`	
	`leal -4(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

5.1.5 Benchmark

```
  char a[8] __attribute__((aligned));
  char b[8] __attribute__((aligned));
```

```
 char c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 16000+i;
     b[i] = 16000+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	11.747
GCC 4.0 - not optimized	11.66
GCC 4.1 - not optimized	11.698
ICC 8.1 - not optimized	11.086
GCC 4.0	5.707
GCC 4.1	5.993
ICC 8.1	4.392
GCC SIMD	1.062
ICC SIMD	1.16

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

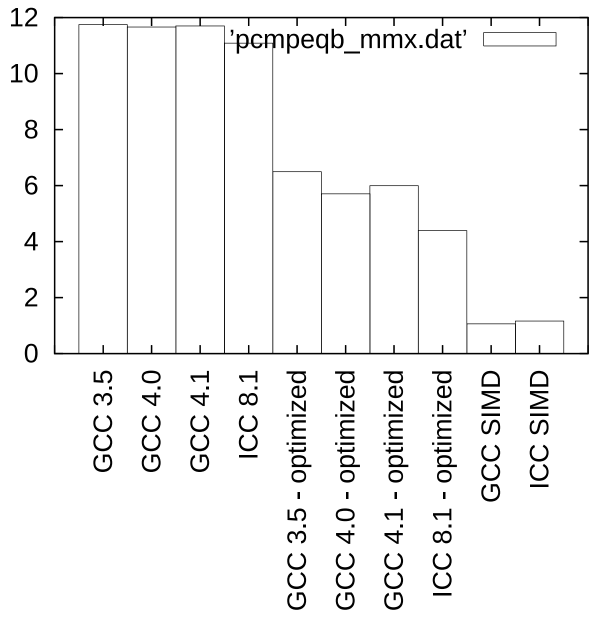

Figure 39: Benchmarks for pcmpeqb - MMX version

5.2 pcmpeqb - SSE2 (128 bits registers) version

5.2.1 C code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i;

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

5.2.2 GIMPLE code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i;
  loop_label::
  if(i >= 16)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 == T2)
    T3 = 0xFF;
  else
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.2.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[16], char b[16], char c[16])
{
  *(__m64 *) c = _mm_cmpeq_epi8(*(__m64*) a,  *(__m64*) b);
}
```

5.2.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %edx`
`movl 8(%ebp), %eax`	`subl $4, %esp`	`movl %esp, %ebp`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`pushl %esi`
`movl 12(%ebp), %eax`	`.L2:`	`movl 16(%ebp), %ecx`

```
SIMD intrinsics              -nooptim                      -O2 and vectorizer
 pcmpeqb (%eax), %xmm0        cmpl $15, -4(%ebp)            pushl %ebx
 movl 16(%ebp), %eax          jle .L5                       movl 8(%ebp), %esi
 movdqa %xmm0, (%eax)         jmp .L1                       movl 12(%ebp), %ebx
 popl %ebp                   .L5:                           .p2align 4,,15
 ret                          movl -4(%ebp), %eax          .L2:
                              movl 8(%ebp), %ecx            movzbl -1(%edx,%ebx), %eax
                              addl %eax, %ecx               cmpb %al, -1(%esi,%edx)
                              movl -4(%ebp), %eax           setne %al
                              movl 12(%ebp), %edx           decb %al
                              addl %eax, %edx               movb %al, -1(%edx,%ecx)
                              movzbl (%ecx), %eax           incl %edx
                              cmpb (%edx), %al              cmpl $17, %edx
                              jne .L6                       jne .L2
                              movl -4(%ebp), %eax           popl %ebx
                              addl 16(%ebp), %eax           popl %esi
                              movb $-1, (%eax)              popl %ebp
                              jmp .L4                       ret
                             .L6:
                              movl -4(%ebp), %eax
                              addl 16(%ebp), %eax
                              movb $0, (%eax)
                             .L4:
                              leal -4(%ebp), %eax
                              incl (%eax)
                              jmp .L2
                             .L1:
                              leave
                              ret
```

5.2.5 Benchmark

```
 char a[16] __attribute__((aligned));
 char b[16] __attribute__((aligned));
 char c[16] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = 100+i;
     b[i] = 100+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	25.175
GCC 4.0 - not optimized	25.721
GCC 4.1 - not optimized	25.821
ICC 8.1 - not optimized	25.195

GCC 4.0	13.308
GCC 4.1	12.852
ICC 8.1	2.245

GCC SIMD	0.782
ICC SIMD	1.395

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	`pcmpeqb` vectorization+

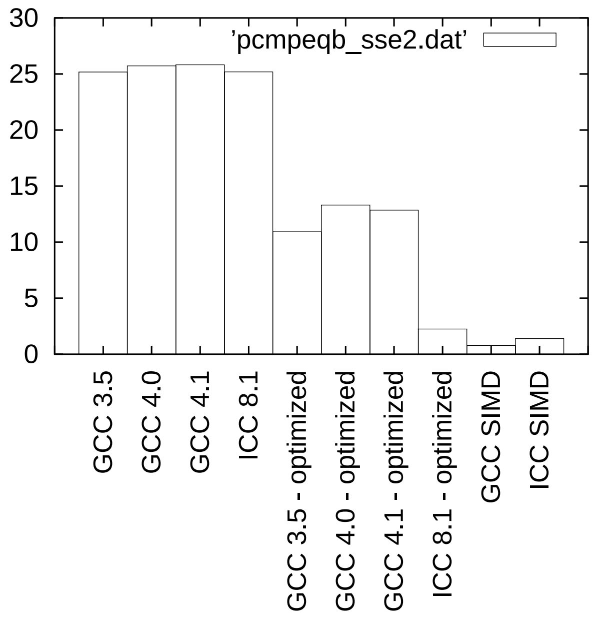

Figure 40: Benchmarks for pcmpeqb - SSE2 version

5.3 pcmpeqw - MMX (64 bits registers) version

5.3.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      if(a[i] == b[i])
        {
          c[i] = 0xFFFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.3.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 == T2)
    T3 = 0xFFFF;
  else
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.3.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_cmpeq_pi16(*(__m64*) a,  *(__m64*) b);
}
```

5.3.4 Assembly code

The assembly codes that are generated are the following:

```
SIMD intrinsics               -nooptim                      -O2 and vectorizer
--------------------------------------------------------------------------------------
  pushl %ebp                    pushl %ebp                    pushl %ebp
  movl %esp, %ebp               movl %esp, %ebp               movl $1, %ecx
  movl 8(%ebp), %eax            pushl %esi                    movl %esp, %ebp
  movq (%eax), %mm0             pushl %ebx                    pushl %edi
  movl 12(%ebp), %eax           subl $4, %esp                 movl 8(%ebp), %edi
  pcmpeqw (%eax), %mm0          movl $0, -12(%ebp)            pushl %esi
  movl 16(%ebp), %eax          .L2:                           movl 12(%ebp), %esi
  movq %mm0, (%eax)             cmpl $3, -12(%ebp)            pushl %ebx
  popl %ebp                     jle .L5                       movl 16(%ebp), %ebx
  ret                           jmp .L1                       .p2align 4,,15
                               .L5:                          .L2:
                                movl -12(%ebp), %eax          leal (%ecx,%ecx), %edx
                                leal (%eax,%eax), %esi        movzwl -2(%edx,%esi), %eax
                                movl 8(%ebp), %ebx            cmpw %ax, -2(%edi,%edx)
                                movl -12(%ebp), %eax          setne %al
                                leal (%eax,%eax), %ecx        incl %ecx
                                movl 12(%ebp), %edx           movzbw %al, %ax
                                movzwl (%ebx,%esi), %eax      decl %eax
                                cmpw (%edx,%ecx), %ax         cmpl $5, %ecx
                                jne .L6                       movw %ax, -2(%edx,%ebx)
                                movl -12(%ebp), %eax          jne .L2
                                leal (%eax,%eax), %edx        popl %ebx
```

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`movl 16(%ebp), %eax`	`popl %esi`
	`movw $-1, (%eax,%edx)`	`popl %edi`
	`jmp .L4`	`popl %ebp`
	`.L6:`	`ret`
	`movl -12(%ebp), %eax`	
	`leal (%eax,%eax), %edx`	
	`movl 16(%ebp), %eax`	
	`movw $0, (%eax,%edx)`	
	`.L4:`	
	`leal -12(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`addl $4, %esp`	
	`popl %ebx`	
	`popl %esi`	
	`popl %ebp`	
	`ret`	

5.3.5 Benchmark

```
 short int a[4] __attribute__((aligned));
 short int b[4] __attribute__((aligned));
 short int c[4] __attribute__((aligned));
 int i;

 for(i = 0; i<4; i++)
   {
     a[i] = 16000+i;
     b[i] = 16000+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	5.472
GCC 4.0 - not optimized	6.383
GCC 4.1 - not optimized	6.759
ICC 8.1 - not optimized	5.424

GCC 4.0	5.348
GCC 4.1	4.865
ICC 8.1	2.309

GCC SIMD	1.077
ICC SIMD	1.15

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

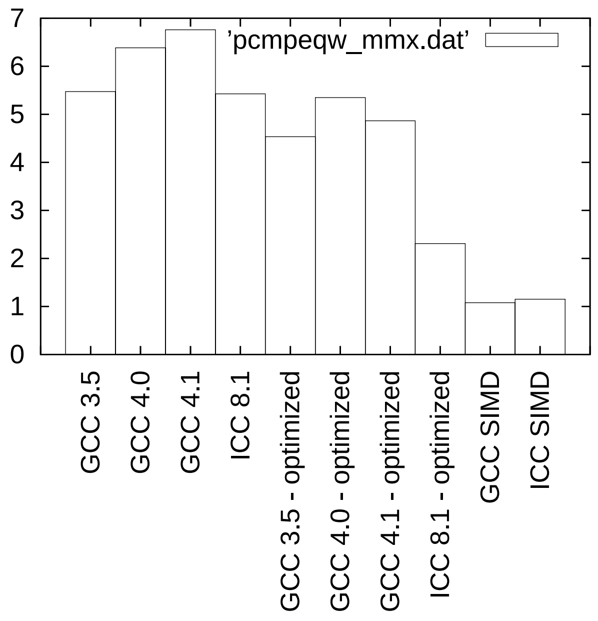

Figure 41: Benchmarks for pcmpeqw - MMX version

5.4 pcmpeqw - SSE2 (128 bits registers) version

5.4.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      if(a[i] == b[i])
        {
          c[i] = 0xFFFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.4.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 == T2)
    T3 = 0xFFFF;
  else
```

```
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.4.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_cmpeq_epi16(*(__m128i*) a,  *(__m128i*) b);
}
```

5.4.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                      -O2 and vectorizer
 pushl %ebp                    pushl %ebp                    pushl %ebp
 movl %esp, %ebp               movl %esp, %ebp               movl $1, %ecx
 movl 8(%ebp), %eax            pushl %esi                    movl %esp, %ebp
 movdqa (%eax), %xmm0          pushl %ebx                    pushl %edi
 movl 12(%ebp), %eax           subl $4, %esp                 movl 8(%ebp), %edi
 pcmpeqw (%eax), %xmm0         movl $0, -12(%ebp)            pushl %esi
 movl 16(%ebp), %eax          .L2:                           movl 12(%ebp), %esi
 movdqa %xmm0, (%eax)          cmpl $7, -12(%ebp)            pushl %ebx
 popl %ebp                     jle .L5                       movl 16(%ebp), %ebx
 ret                           jmp .L1                       .p2align 4,,15
                              .L5:                          .L2:
                               movl -12(%ebp), %eax          leal (%ecx,%ecx), %edx
                               leal (%eax,%eax), %esi        movzwl -2(%edx,%esi), %eax
                               movl 8(%ebp), %ebx            cmpw %ax, -2(%edi,%edx)
                               movl -12(%ebp), %eax          setne %al
                               leal (%eax,%eax), %ecx        incl %ecx
                               movl 12(%ebp), %edx           movzbw %al, %ax
                               movzwl (%ebx,%esi), %eax      decl %eax
                               cmpw (%edx,%ecx), %ax         cmpl $9, %ecx
                               jne .L6                       movw %ax, -2(%edx,%ebx)
                               movl -12(%ebp), %eax          jne .L2
                               leal (%eax,%eax), %edx        popl %ebx
                               movl 16(%ebp), %eax           popl %esi
                               movw $-1, (%eax,%edx)         popl %edi
                               jmp .L4                       popl %ebp
                              .L6:                           ret
                               movl -12(%ebp), %eax
                               leal (%eax,%eax), %edx
                               movl 16(%ebp), %eax
                               movw $0, (%eax,%edx)
                              .L4:
                               leal -12(%ebp), %eax
                               incl (%eax)
                               jmp .L2
                              .L1:
                               addl $4, %esp
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

	`popl %ebx`
	`popl %esi`
	`popl %ebp`
	`ret`	

5.4.5 Benchmark

```
  short int a[8] __attribute__((aligned));
  short int b[8] __attribute__((aligned));
  short int c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<16; i++)
    {
      a[i] = 16000+i;
      b[i] = 16000+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	11.772
GCC 4.0 - not optimized	12.529
GCC 4.1 - not optimized	12.337
ICC 8.1 - not optimized	11.551

GCC 4.0	7.067
GCC 4.1	6.917
ICC 8.1	2.231

GCC SIMD	0.938
ICC SIMD	1.509

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	`pcmpeqw` vectorization+

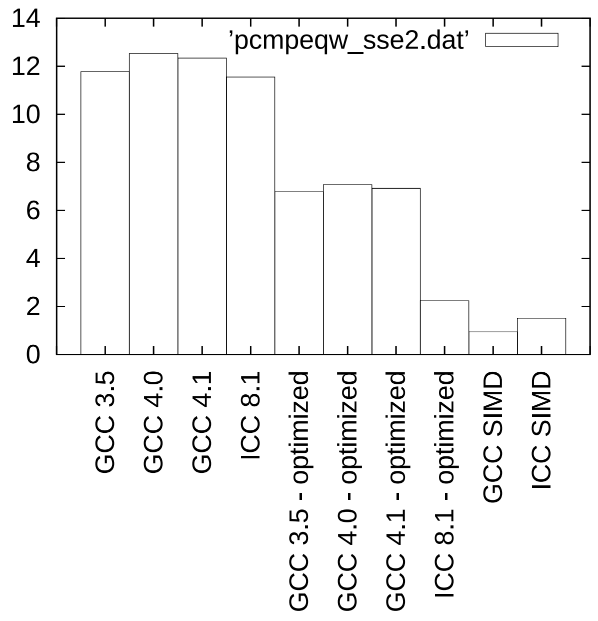

Figure 42: Benchmarks for pcmpeqw - SSE2 version

5.5 pcmpeqd - MMX (64 bits registers) version

5.5.1 C code

```
void test_loop_c(int a[2], int b[2], int c[2])
{
  int i;

  for(i=0; i<2; i++)
    {
      if(a[i] == b[i])
        {
          c[i] = 0xFFFFFFFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.5.2 GIMPLE code

```
void test_loop_c(int a[2], int b[2], int c[2])
{
  int i;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 == T2)
    T3 = 0xFFFFFFFF;
  else
```

```
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.5.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[2], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_cmpeq_pi32(*(__m64*) a,  *(__m64*) b);
}
```

5.5.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics               -nooptim                      -O2 and vectorizer
  pushl %ebp                    pushl %ebp                    pushl %ebp
  movl %esp, %ebp               movl %esp, %ebp               movl $1, %ecx
  movl 8(%ebp), %eax            pushl %esi                    movl %esp, %ebp
  movq (%eax), %mm0             pushl %ebx                    pushl %edi
  movl 12(%ebp), %eax           subl $4, %esp                 movl 8(%ebp), %edi
  pcmpeqd (%eax), %mm0          movl $0, -12(%ebp)            pushl %esi
  movl 16(%ebp), %eax          .L2:                           movl 12(%ebp), %esi
  movq %mm0, (%eax)             cmpl $1, -12(%ebp)            pushl %ebx
  popl %ebp                     jle .L5                       movl 16(%ebp), %ebx
  ret                           jmp .L1                       .p2align 4,,15
                               .L5:                          .L2:
                                movl -12(%ebp), %eax          leal 0(,%ecx,4), %edx
                                leal 0(,%eax,4), %esi         movl -4(%edx,%esi), %eax
                                movl 8(%ebp), %ebx            cmpl %eax, -4(%edi,%edx)
                                movl -12(%ebp), %eax          setne %al
                                leal 0(,%eax,4), %ecx         incl %ecx
                                movl 12(%ebp), %edx           movzbl %al, %eax
                                movl (%ebx,%esi), %eax        decl %eax
                                cmpl (%edx,%ecx), %eax        cmpl $3, %ecx
                                jne .L6                       movl %eax, -4(%edx,%ebx)
                                movl -12(%ebp), %eax          jne .L2
                                leal 0(,%eax,4), %edx         popl %ebx
                                movl 16(%ebp), %eax           popl %esi
                                movl $-1, (%eax,%edx)         popl %edi
                                jmp .L4                       popl %ebp
                               .L6:                           ret
                                movl -12(%ebp), %eax
                                leal 0(,%eax,4), %edx
                                movl 16(%ebp), %eax
                                movl $0, (%eax,%edx)
                               .L4:
                                leal -12(%ebp), %eax
                                incl (%eax)
                                jmp .L2
                               .L1:
                                addl $4, %esp
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

	`popl %ebx`
	`popl %esi`
	`popl %ebp`
	`ret`	

5.5.5 Benchmark

```
 int a[2] __attribute__((aligned));
 int b[2] __attribute__((aligned));
 int c[2] __attribute__((aligned));
 int i;

 for(i = 0; i<2; i++)
   {
     a[i] = 16000+i;
     b[i] = 16000+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	4.838
GCC 4.0 - not optimized	4.356
GCC 4.1 - not optimized	4.432
ICC 8.1 - not optimized	4.559

GCC 4.0	2.803
GCC 4.1	3.287
ICC 8.1	1.738

GCC SIMD	1.076
ICC SIMD	1.305

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

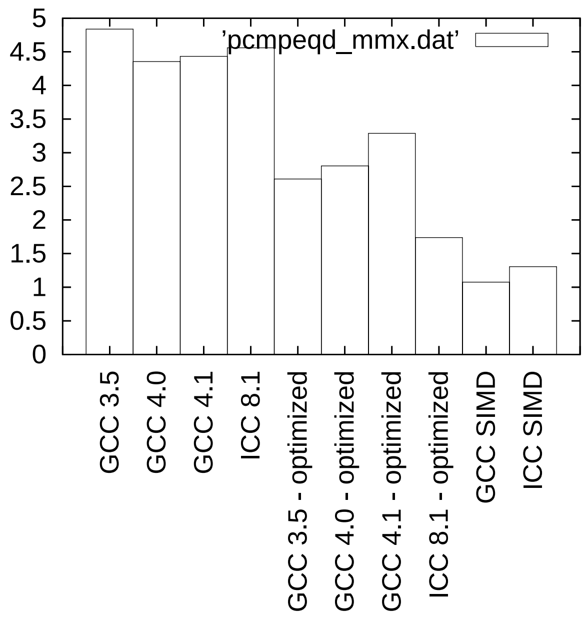

Figure 43: Benchmarks for pcmpeqd - MMX version

5.6 pcmpeqd - SSE2 (128 bits registers) version

5.6.1 C code

```
void test_loop_c(int a[4], int b[4], int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      if(a[i] == b[i])
        {
          c[i] = 0xFFFFFFFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.6.2 GIMPLE code

```
void test_loop_c(int a[4], int b[4], int c[4])
{
  int i;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 == T2)
    T3 = 0xFFFFFFFF;
  else
```

```
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.6.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(int a[4], int b[4], int c[4])
{
  *(__m128i *) c = _mm_cmpeq_epi32(*(__m128i*) a,  *(__m128i*) b);
}
```

5.6.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                     -O2 and vectorizer
-------------------------------------------------------------------------------------
  pushl %ebp                   pushl %ebp                   pushl %ebp
  movl %esp, %ebp              movl %esp, %ebp              movl $1, %ecx
  movl 8(%ebp), %eax           pushl %esi                   movl %esp, %ebp
  movdqa (%eax), %xmm0         pushl %ebx                   pushl %edi
  movl 12(%ebp), %eax          subl $4, %esp                movl 8(%ebp), %edi
  pcmpeqd (%eax), %xmm0        movl $0, -12(%ebp)           pushl %esi
  movl 16(%ebp), %eax         .L2:                          movl 12(%ebp), %esi
  movdqa %xmm0, (%eax)         cmpl $3, -12(%ebp)           pushl %ebx
  popl %ebp                    jle .L5                      movl 16(%ebp), %ebx
  ret                          jmp .L1                      .p2align 4,,15
                              .L5:                         .L2:
                               movl -12(%ebp), %eax         leal 0(,%ecx,4), %edx
                               leal 0(,%eax,4), %esi        movl -4(%edx,%esi), %eax
                               movl 8(%ebp), %ebx           cmpl %eax, -4(%edi,%edx)
                               movl -12(%ebp), %eax         setne %al
                               leal 0(,%eax,4), %ecx        incl %ecx
                               movl 12(%ebp), %edx          movzbl %al, %eax
                               movl (%ebx,%esi), %eax       decl %eax
                               cmpl (%edx,%ecx), %eax       cmpl $5, %ecx
                               jne .L6                      movl %eax, -4(%edx,%ebx)
                               movl -12(%ebp), %eax         jne .L2
                               leal 0(,%eax,4), %edx        popl %ebx
                               movl 16(%ebp), %eax          popl %esi
                               movl $-1, (%eax,%edx)        popl %edi
                               jmp .L4                      popl %ebp
                              .L6:                          ret
                               movl -12(%ebp), %eax
                               leal 0(,%eax,4), %edx
                               movl 16(%ebp), %eax
                               movl $0, (%eax,%edx)
                              .L4:
                               leal -12(%ebp), %eax
                               incl (%eax)
                               jmp .L2
                              .L1:
                               addl $4, %esp
```

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`popl %ebx`	
	`popl %esi`	
	`popl %ebp`	
	`ret`	

5.6.5 Benchmark

```
 int a[4] __attribute__((aligned));
 int b[4] __attribute__((aligned));
 int c[4] __attribute__((aligned));
 int i;

 for(i = 0; i<4; i++)
   {
     a[i] = 16000+i;
     b[i] = 16000+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	7.928
GCC 4.0 - not optimized	6.645
GCC 4.1 - not optimized	6.687
ICC 8.1 - not optimized	5.164

GCC 4.0	4.648
GCC 4.1	4.821
ICC 8.1	2.765

GCC SIMD	0.934
ICC SIMD	1.508

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	`pcmpeqd` vectorization+

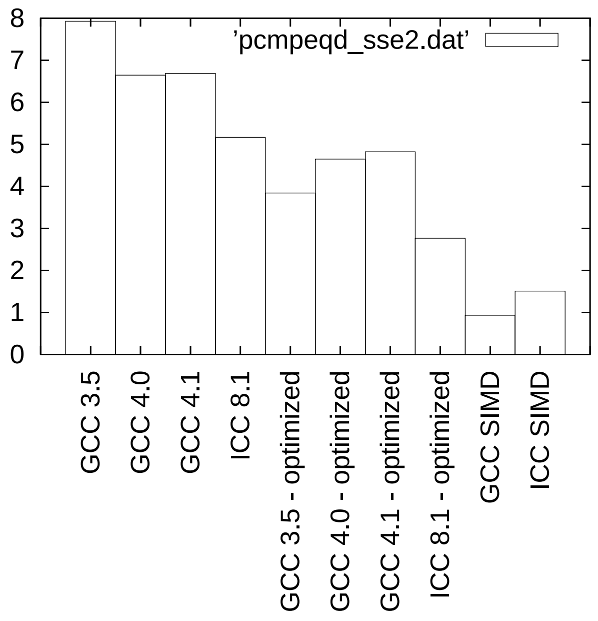

Figure 44: Benchmarks for pcmpeqd - SSE2 version

5.7 pcmpgtb - MMX (64 bits registers) version

5.7.1 C code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      if(a[i] > b[i])
        {
          c[i] = 0xFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.7.2 GIMPLE code

```
void test_loop_c(char a[8], char b[8], char c[8])
{
  int i;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = 0xFF;
  else
```

```
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.7.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[8], char b[8], char c[8])
{
  *(__m64 *) c = _mm_cmpgt_pi8(*(__m64*) a,  *(__m64*) b);
}
```

5.7.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %edx`
`movl 8(%ebp), %eax`	`subl $4, %esp`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`pushl %esi`
`movl 12(%ebp), %eax`	`.L2:`	`movl 16(%ebp), %ecx`
`pcmpgtb (%eax), %mm0`	`cmpl $7, -4(%ebp)`	`pushl %ebx`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 8(%ebp), %esi`
`movq %mm0, (%eax)`	`jmp .L1`	`movl 12(%ebp), %ebx`
`popl %ebp`	`.L5:`	`.p2align 4,,15`
`ret`	`movl -4(%ebp), %eax`	`.L2:`
	`movl 8(%ebp), %ecx`	`movzbl -1(%edx,%ebx), %eax`
	`addl %eax, %ecx`	`cmpb %al, -1(%esi,%edx)`
	`movl -4(%ebp), %eax`	`setle %al`
	`movl 12(%ebp), %edx`	`decb %al`
	`addl %eax, %edx`	`movb %al, -1(%edx,%ecx)`
	`movzbl (%ecx), %eax`	`incl %edx`
	`cmpb (%edx), %al`	`cmpl $9, %edx`
	`jle .L6`	`jne .L2`
	`movl -4(%ebp), %eax`	`popl %ebx`
	`addl 16(%ebp), %eax`	`popl %esi`
	`movb $-1, (%eax)`	`popl %ebp`
	`jmp .L4`	`ret`
	`.L6:`	
	`movl -4(%ebp), %eax`	
	`addl 16(%ebp), %eax`	
	`movb $0, (%eax)`	
	`.L4:`	
	`leal -4(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

5.7.5 Benchmark

```
  char a[8] __attribute__((aligned));
  char b[8] __attribute__((aligned));
  char c[8] __attribute__((aligned));
  int i;
```

```
 for(i = 0; i<8; i++)
   {
     a[i] = 16000+i;
     b[i] = 16000+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	12.457
GCC 4.0 - not optimized	15.7
GCC 4.1 - not optimized	16.166
ICC 8.1 - not optimized	11.672
GCC 4.0	6.084
GCC 4.1	6.106
ICC 8.1	4.303
GCC SIMD	0.929
ICC SIMD	1.304

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

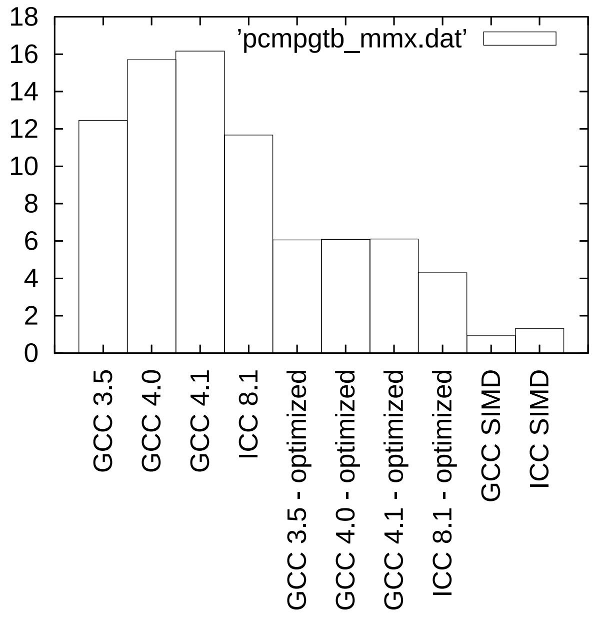

Figure 45: Benchmarks for pcmpgtb - MMX version

5.8 pcmpgtb - SSE2 (128 bits registers) version

5.8.1 C code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i;

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

5.8.2 GIMPLE code

```
void test_loop_c(char a[16], char b[16], char c[16])
{
  int i;
  loop_label::
  if(i >= 16)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = 0xFF;
  else
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.8.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(char a[16], char b[16], char c[16])
{
  *(__m64 *) c = _mm_cmpgt_epi8(*(__m64*) a,  *(__m64*) b);
}
```

5.8.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %edx`
`movl 8(%ebp), %eax`	`subl $4, %esp`	`movl %esp, %ebp`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`pushl %esi`
`movl 12(%ebp), %eax`	`.L2:`	`movl 16(%ebp), %ecx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pcmpgtb (%eax), %xmm0`	`cmpl $15, -4(%ebp)`	`pushl %ebx`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 8(%ebp), %esi`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`movl 12(%ebp), %ebx`
`popl %ebp`	`.L5:`	`.p2align 4,,15`
`ret`	`movl -4(%ebp), %eax`	`.L2:`
	`movl 8(%ebp), %ecx`	`movzbl -1(%edx,%ebx), %eax`
	`addl %eax, %ecx`	`cmpb %al, -1(%esi,%edx)`
	`movl -4(%ebp), %eax`	`setle %al`
	`movl 12(%ebp), %edx`	`decb %al`
	`addl %eax, %edx`	`movb %al, -1(%edx,%ecx)`
	`movzbl (%ecx), %eax`	`incl %edx`
	`cmpb (%edx), %al`	`cmpl $17, %edx`
	`jle .L6`	`jne .L2`
	`movl -4(%ebp), %eax`	`popl %ebx`
	`addl 16(%ebp), %eax`	`popl %esi`
	`movb $-1, (%eax)`	`popl %ebp`
	`jmp .L4`	`ret`
	`.L6:`	
	`movl -4(%ebp), %eax`	
	`addl 16(%ebp), %eax`	
	`movb $0, (%eax)`	
	`.L4:`	
	`leal -4(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

5.8.5 Benchmark

```
 char a[16] __attribute__((aligned));
 char b[16] __attribute__((aligned));
 char c[16] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = 100+i;
     b[i] = 100+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	26.822
GCC 4.0 - not optimized	23.95
GCC 4.1 - not optimized	30.131
ICC 8.1 - not optimized	30.276

GCC 4.0	15.809
GCC 4.1	16.852
ICC 8.1	3.208

GCC SIMD	0.955
ICC SIMD	2.164

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	`pcmpgtb` vectorization+

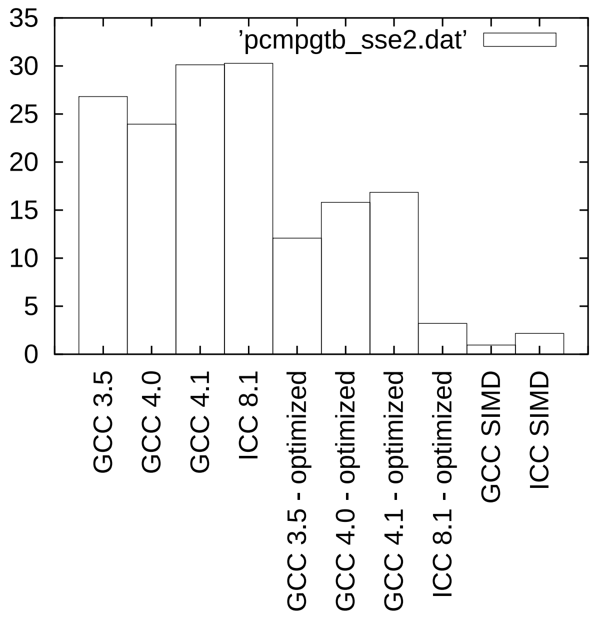

Figure 46: Benchmarks for pcmpgtb - SSE2 version

5.9 pcmpgtw - MMX (64 bits registers) version

5.9.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      if(a[i] > b[i])
        {
          c[i] = 0xFFFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.9.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = 0xFFFF;
  else
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.9.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_cmpgt_pi16(*(__m64*) a,  *(__m64*) b);
}
```

5.9.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %ecx`
`movl 8(%ebp), %eax`	`pushl %esi`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`pushl %ebx`	`pushl %edi`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`movl 8(%ebp), %edi`
`pcmpgtw (%eax), %mm0`	`movl $0, -12(%ebp)`	`pushl %esi`
`movl 16(%ebp), %eax`	`.L2:`	`movl 12(%ebp), %esi`
`movq %mm0, (%eax)`	`cmpl $3, -12(%ebp)`	`pushl %ebx`
`popl %ebp`	`jle .L5`	`movl 16(%ebp), %ebx`
`ret`	`jmp .L1`	`.p2align 4,,15`
	`.L5:`	`.L2:`
	`movl -12(%ebp), %eax`	`leal (%ecx,%ecx), %edx`
	`leal (%eax,%eax), %esi`	`movzwl -2(%edx,%esi), %eax`
	`movl 8(%ebp), %ebx`	`cmpw %ax, -2(%edi,%edx)`
	`movl -12(%ebp), %eax`	`setle %al`
	`leal (%eax,%eax), %ecx`	`incl %ecx`
	`movl 12(%ebp), %edx`	`movzbw %al, %ax`
	`movzwl (%ebx,%esi), %eax`	`decl %eax`
	`cmpw (%edx,%ecx), %ax`	`cmpl $5, %ecx`
	`jle .L6`	`movw %ax, -2(%edx,%ebx)`
	`movl -12(%ebp), %eax`	`jne .L2`
	`leal (%eax,%eax), %edx`	`popl %ebx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`movl 16(%ebp), %eax`	`popl %esi`
	`movw $-1, (%eax,%edx)`	`popl %edi`
	`jmp .L4`	`popl %ebp`
	`.L6:`	`ret`
	`movl -12(%ebp), %eax`	
	`leal (%eax,%eax), %edx`	
	`movl 16(%ebp), %eax`	
	`movw $0, (%eax,%edx)`	
	`.L4:`	
	`leal -12(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`addl $4, %esp`	
	`popl %ebx`	
	`popl %esi`	
	`popl %ebp`	
	`ret`	

5.9.5 Benchmark

```
  short int a[4] __attribute__((aligned));
  short int b[4] __attribute__((aligned));
  short int c[4] __attribute__((aligned));
  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = 16000+i;
      b[i] = 16000+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	9.123
GCC 4.0 - not optimized	9.416
GCC 4.1 - not optimized	9.351
ICC 8.1 - not optimized	8.118

GCC 4.0	6.197
GCC 4.1	6.334
ICC 8.1	3.715

GCC SIMD	1.422
ICC SIMD	1.68

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

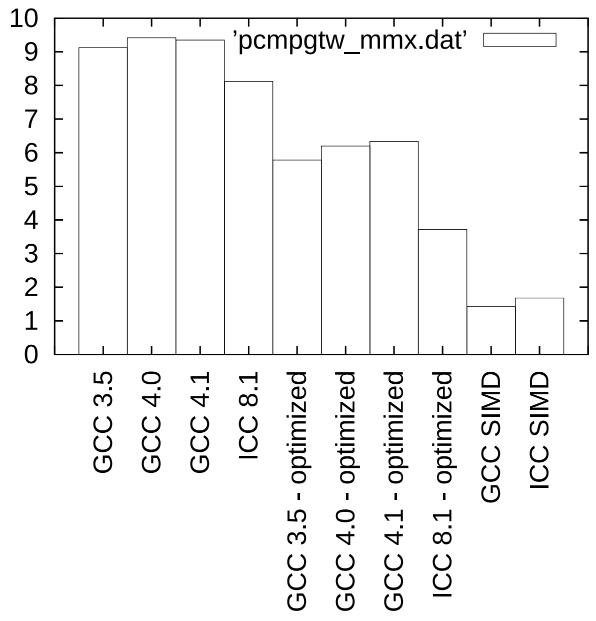

Figure 47: Benchmarks for pcmpgtw - MMX version

5.10 pcmpgtw - SSE2 (128 bits registers) version

5.10.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      if(a[i] > b[i])
        {
          c[i] = 0xFFFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.10.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = 0xFFFF;
  else
```

```
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.10.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_cmpgt_epi16(*(__m128i*) a,  *(__m128i*) b);
}
```

5.10.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                      -O2 and vectorizer
----------------------------------------------------------------------------------------
  pushl %ebp                   pushl %ebp                    pushl %ebp
  movl %esp, %ebp              movl %esp, %ebp               movl $1, %ecx
  movl 8(%ebp), %eax           pushl %esi                    movl %esp, %ebp
  movdqa (%eax), %xmm0         pushl %ebx                    pushl %edi
  movl 12(%ebp), %eax          subl $4, %esp                 movl 8(%ebp), %edi
  pcmpgtw (%eax), %xmm0        movl $0, -12(%ebp)            pushl %esi
  movl 16(%ebp), %eax         .L2:                           movl 12(%ebp), %esi
  movdqa %xmm0, (%eax)         cmpl $7, -12(%ebp)            pushl %ebx
  popl %ebp                    jle .L5                       movl 16(%ebp), %ebx
  ret                          jmp .L1                       .p2align 4,,15
                              .L5:                          .L2:
                               movl -12(%ebp), %eax          leal (%ecx,%ecx), %edx
                               leal (%eax,%eax), %esi        movzwl -2(%edx,%esi), %eax
                               movl 8(%ebp), %ebx            cmpw %ax, -2(%edi,%edx)
                               movl -12(%ebp), %eax          setle %al
                               leal (%eax,%eax), %ecx        incl %ecx
                               movl 12(%ebp), %edx           movzbw %al, %ax
                               movzwl (%ebx,%esi), %eax      decl %eax
                               cmpw (%edx,%ecx), %ax         cmpl $9, %ecx
                               jle .L6                       movw %ax, -2(%edx,%ebx)
                               movl -12(%ebp), %eax          jne .L2
                               leal (%eax,%eax), %edx        popl %ebx
                               movl 16(%ebp), %eax           popl %esi
                               movw $-1, (%eax,%edx)         popl %edi
                               jmp .L4                       popl %ebp
                              .L6:                           ret
                               movl -12(%ebp), %eax
                               leal (%eax,%eax), %edx
                               movl 16(%ebp), %eax
                               movw $0, (%eax,%edx)
                              .L4:
                               leal -12(%ebp), %eax
                               incl (%eax)
                               jmp .L2
                              .L1:
                               addl $4, %esp
```

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`popl %ebx`	
	`popl %esi`	
	`popl %ebp`	
	`ret`	

5.10.5 Benchmark

```
  short int a[8] __attribute__((aligned));
  short int b[8] __attribute__((aligned));
  short int c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<16; i++)
    {
      a[i] = 16000+i;
      b[i] = 16000+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	15.04
GCC 4.0 - not optimized	16.446
GCC 4.1 - not optimized	16.182
ICC 8.1 - not optimized	14.203

GCC 4.0	9.48
GCC 4.1	9.463
ICC 8.1	3.081

GCC SIMD	1.337
ICC SIMD	2.3

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	`pcmpgtw` vectorization+

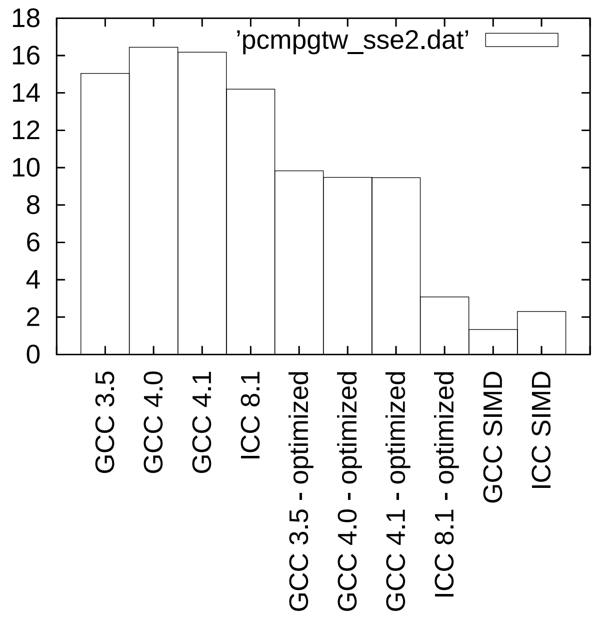

Figure 48: Benchmarks for pcmpgtw - SSE2 version

5.11 pcmpgtd - MMX (64 bits registers) version

5.11.1 C code

```
void test_loop_c(int a[2], int b[2], int c[2])
{
  int i;

  for(i=0; i<2; i++)
    {
      if(a[i] > b[i])
        {
          c[i] = 0xFFFFFFFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.11.2 GIMPLE code

```
void test_loop_c(int a[2], int b[2], int c[2])
{
  int i;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = 0xFFFFFFFF;
  else
```

```
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.11.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[2], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_cmpgt_pi32(*(__m64*) a,  *(__m64*) b);
}
```

5.11.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %ecx`
`movl 8(%ebp), %eax`	`pushl %esi`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`pushl %ebx`	`pushl %edi`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`movl 8(%ebp), %edi`
`pcmpgtd (%eax), %mm0`	`movl $0, -12(%ebp)`	`pushl %esi`
`movl 16(%ebp), %eax`	`.L2:`	`movl 12(%ebp), %esi`
`movq %mm0, (%eax)`	`cmpl $1, -12(%ebp)`	`pushl %ebx`
`popl %ebp`	`jle .L5`	`movl 16(%ebp), %ebx`
`ret`	`jmp .L1`	`.p2align 4,,15`
	`.L5:`	`.L2:`
	`movl -12(%ebp), %eax`	`leal 0(,%ecx,4), %edx`
	`leal 0(,%eax,4), %esi`	`movl -4(%edx,%esi), %eax`
	`movl 8(%ebp), %ebx`	`cmpl %eax, -4(%edi,%edx)`
	`movl -12(%ebp), %eax`	`setle %al`
	`leal 0(,%eax,4), %ecx`	`incl %ecx`
	`movl 12(%ebp), %edx`	`movzbl %al, %eax`
	`movl (%ebx,%esi), %eax`	`decl %eax`
	`cmpl (%edx,%ecx), %eax`	`cmpl $3, %ecx`
	`jle .L6`	`movl %eax, -4(%edx,%ebx)`
	`movl -12(%ebp), %eax`	`jne .L2`
	`leal 0(,%eax,4), %edx`	`popl %ebx`
	`movl 16(%ebp), %eax`	`popl %esi`
	`movl $-1, (%eax,%edx)`	`popl %edi`
	`jmp .L4`	`popl %ebp`
	`.L6:`	`ret`
	`movl -12(%ebp), %eax`	
	`leal 0(,%eax,4), %edx`	
	`movl 16(%ebp), %eax`	
	`movl $0, (%eax,%edx)`	
	`.L4:`	
	`leal -12(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`addl $4, %esp`	

SIMD intrinsics	-nooptim	-O2 and vectorizer
	popl %ebx popl %esi popl %ebp ret	

5.11.5 Benchmark

```
 int a[2] __attribute__((aligned));
 int b[2] __attribute__((aligned));
 int c[2] __attribute__((aligned));
 int i;

 for(i = 0; i<2; i++)
   {
     a[i] = 16000+i;
     b[i] = 16000+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	6.027
GCC 4.0 - not optimized	5.489
GCC 4.1 - not optimized	5.259
ICC 8.1 - not optimized	4.779

GCC 4.0	3.762
GCC 4.1	4.098
ICC 8.1	2.974

GCC SIMD	1.343
ICC SIMD	2.291

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

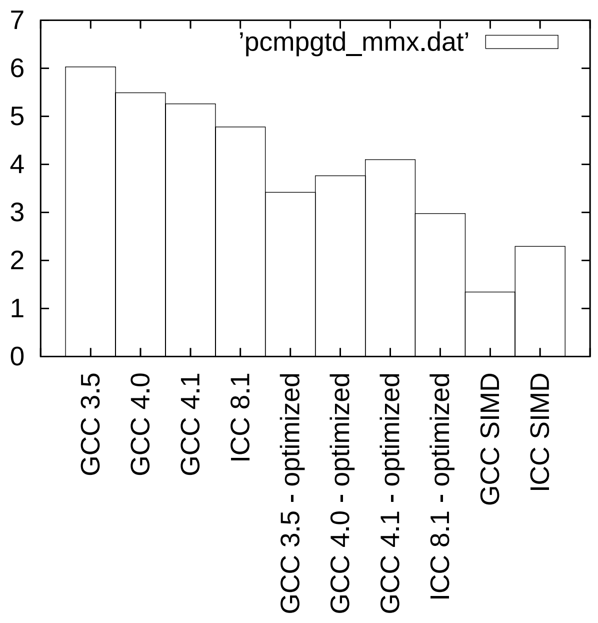

Figure 49: Benchmarks for pcmpgtd - MMX version

5.12 pcmpgtd - SSE2 (128 bits registers) version

5.12.1 C code

```
void test_loop_c(int a[4], int b[4], int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      if(a[i] > b[i])
        {
          c[i] = 0xFFFFFFFF;
        }
      else
        {
          c[i] = 0;
        }
    }
}
```

5.12.2 GIMPLE code

```
void test_loop_c(int a[4], int b[4], int c[4])
{
  int i;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = 0xFFFFFFFF;
  else
```

```
    T3 = 0;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.12.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(int a[4], int b[4], int c[4])
{
  *(__m128i *) c = _mm_cmpgt_epi32(*(__m128i*) a,  *(__m128i*) b);
}
```

5.12.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                      -O2 and vectorizer
--------------------------------------------------------------------------------------
  pushl %ebp                   pushl %ebp                    pushl %ebp
  movl %esp, %ebp              movl %esp, %ebp               movl $1, %ecx
  movl 8(%ebp), %eax           pushl %esi                    movl %esp, %ebp
  movdqa (%eax), %xmm0         pushl %ebx                    pushl %edi
  movl 12(%ebp), %eax          subl $4, %esp                 movl 8(%ebp), %edi
  pcmpgtd (%eax), %xmm0        movl $0, -12(%ebp)            pushl %esi
  movl 16(%ebp), %eax         .L2:                           movl 12(%ebp), %esi
  movdqa %xmm0, (%eax)         cmpl $3, -12(%ebp)            pushl %ebx
  popl %ebp                    jle .L5                       movl 16(%ebp), %ebx
  ret                          jmp .L1                       .p2align 4,,15
                              .L5:                          .L2:
                               movl -12(%ebp), %eax          leal 0(,%ecx,4), %edx
                               leal 0(,%eax,4), %esi         movl -4(%edx,%esi), %eax
                               movl 8(%ebp), %ebx            cmpl %eax, -4(%edi,%edx)
                               movl -12(%ebp), %eax          setle %al
                               leal 0(,%eax,4), %ecx         incl %ecx
                               movl 12(%ebp), %edx           movzbl %al, %eax
                               movl (%ebx,%esi), %eax        decl %eax
                               cmpl (%edx,%ecx), %eax        cmpl $5, %ecx
                               jle .L6                       movl %eax, -4(%edx,%ebx)
                               movl -12(%ebp), %eax          jne .L2
                               leal 0(,%eax,4), %edx         popl %ebx
                               movl 16(%ebp), %eax           popl %esi
                               movl $-1, (%eax,%edx)         popl %edi
                               jmp .L4                       popl %ebp
                              .L6:                           ret
                               movl -12(%ebp), %eax
                               leal 0(,%eax,4), %edx
                               movl 16(%ebp), %eax
                               movl $0, (%eax,%edx)
                              .L4:
                               leal -12(%ebp), %eax
                               incl (%eax)
                               jmp .L2
                              .L1:
                               addl $4, %esp
```

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`popl %ebx`	
	`popl %esi`	
	`popl %ebp`	
	`ret`	

5.12.5 Benchmark

```
 int a[4] __attribute__((aligned));
 int b[4] __attribute__((aligned));
 int c[4] __attribute__((aligned));
 int i;

 for(i = 0; i<4; i++)
   {
     a[i] = 16000+i;
     b[i] = 16000+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	10.812
GCC 4.0 - not optimized	9.005
GCC 4.1 - not optimized	9.331
ICC 8.1 - not optimized	7.376

GCC 4.0	6.272
GCC 4.1	6.151
ICC 8.1	3.834

GCC SIMD	1.517
ICC SIMD	2.167

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	`pcmpgtd` vectorization+

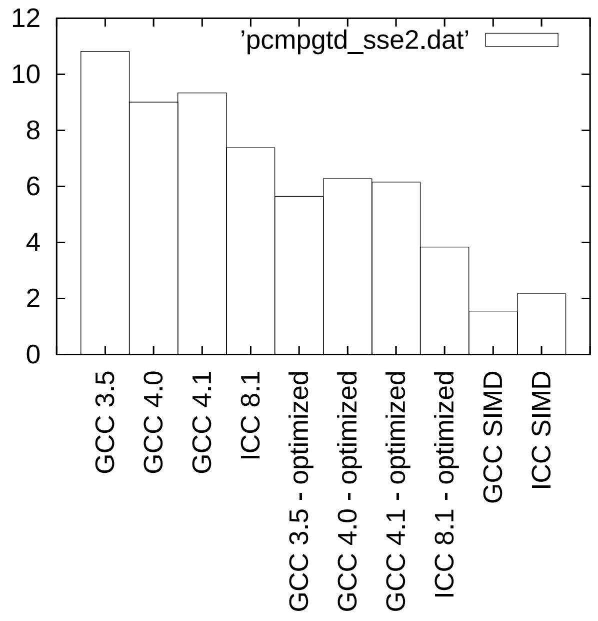

Figure 50: Benchmarks for pcmpgtd - SSE2 version

5.13 pmaxub - MMX (64 bits registers) version

5.13.1 C code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      if(a[i] > b[i])
        {
          c[i] = a[i];
        }
      else
        {
          c[i] = b[i];
        }
    }
}
```

5.13.2 GIMPLE code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = T1;
  else
```

```
    T3 = T2;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.13.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  *(__m64 *) c = _mm_max_pu8(*(__m64*) a,  *(__m64*) b);
}
```

5.13.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %eax`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`pushl %edi`
`movl 8(%ebp), %eax`	`.L2:`	`movl 8(%ebp), %edi`
`pmaxub (%eax), %mm0`	`cmpl $7, -4(%ebp)`	`pushl %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 12(%ebp), %esi`
`movq %mm0, (%eax)`	`jmp .L1`	`pushl %ebx`
`popl %ebp`	`.L5:`	`movl 16(%ebp), %ebx`
`ret`	`movl -4(%ebp), %eax`	`jmp .L2`
	`movl 8(%ebp), %ecx`	`.p2align 4,,7`
	`addl %eax, %ecx`	`.L12:`
	`movl -4(%ebp), %eax`	`movb %cl, -1(%eax,%ebx)`
	`movl 12(%ebp), %edx`	`incl %eax`
	`addl %eax, %edx`	`cmpl $9, %eax`
	`movzbl (%ecx), %eax`	`je .L11`
	`cmpb (%edx), %al`	`.L2:`
	`jbe .L6`	`movzbl -1(%edi,%eax), %ecx`
	`movl -4(%ebp), %eax`	`movzbl -1(%esi,%eax), %edx`
	`movl 16(%ebp), %edx`	`cmpb %dl, %cl`
	`addl %eax, %edx`	`ja .L12`
	`movl -4(%ebp), %eax`	`movb %dl, -1(%eax,%ebx)`
	`addl 8(%ebp), %eax`	`incl %eax`
	`movzbl (%eax), %eax`	`cmpl $9, %eax`
	`movb %al, (%edx)`	`jne .L2`
	`jmp .L4`	`.L11:`
	`.L6:`	`popl %ebx`
	`movl -4(%ebp), %eax`	`popl %esi`
	`movl 16(%ebp), %edx`	`popl %edi`
	`addl %eax, %edx`	`popl %ebp`
	`movl -4(%ebp), %eax`	`ret`
	`addl 12(%ebp), %eax`	
	`movzbl (%eax), %eax`	
	`movb %al, (%edx)`	
	`.L4:`	
	`leal -4(%ebp), %eax`	

SIMD intrinsics	-nooptim	-O2 and vectorizer
	` incl (%eax)`	
	` jmp .L2`	
	`.L1:`	
	` leave`	
	` ret`	

5.13.5 Benchmark

```
  unsigned char a[8] __attribute__((aligned));
  unsigned char b[8] __attribute__((aligned));
  unsigned char c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = 110+i;
      b[i] = 110+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	16.934
GCC 4.0 - not optimized	20.144
GCC 4.1 - not optimized	20.265
ICC 8.1 - not optimized	18.111

GCC 4.0	6.985
GCC 4.1	6.948
ICC 8.1	6.408

GCC SIMD	1.439
ICC SIMD	1.774

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

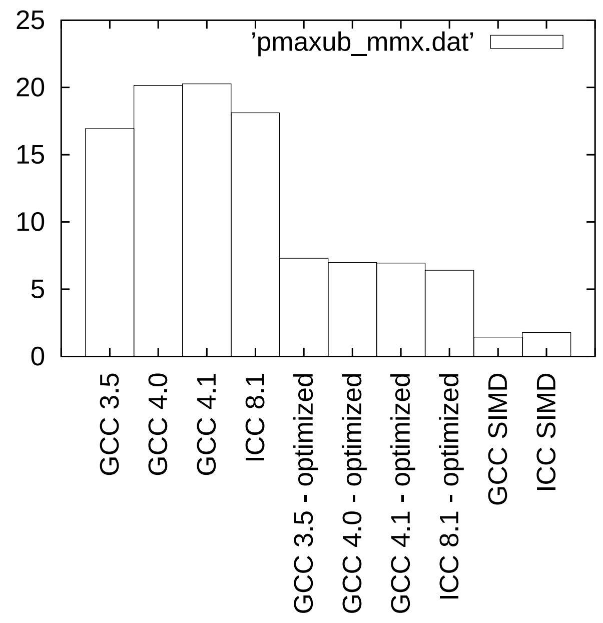

Figure 51: Benchmarks for pmaxub - MMX version

5.14 pmaxub - SSE2 (128 bits registers) version

5.14.1 C code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i;

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

5.14.2 GIMPLE code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i;
  loop_label::
  if(i >= 16)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = T1;
  else
```

```
    T3 = T2;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.14.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  *(__m64 *) c = _mm_max_epu8(*(__m64*) a,  *(__m64*) b);
}
```

5.14.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                      -O2 and vectorizer
  pushl %ebp                    pushl %ebp                    pushl %ebp
  movl %esp, %ebp               movl %esp, %ebp               movl $1, %eax
  movl 12(%ebp), %eax           subl $4, %esp                 movl %esp, %ebp
  movdqa (%eax), %xmm0          movl $0, -4(%ebp)             pushl %edi
  movl 8(%ebp), %eax           .L2:                           movl 8(%ebp), %edi
  pmaxub (%eax), %xmm0          cmpl $15, -4(%ebp)            pushl %esi
  movl 16(%ebp), %eax           jle .L5                       movl 12(%ebp), %esi
  movdqa %xmm0, (%eax)          jmp .L1                       pushl %ebx
  popl %ebp                    .L5:                           movl 16(%ebp), %ebx
  ret                           movl -4(%ebp), %eax           jmp .L2
                                movl 8(%ebp), %ecx            .p2align 4,,7
                                addl %eax, %ecx              .L12:
                                movl -4(%ebp), %eax           movb %cl, -1(%eax,%ebx)
                                movl 12(%ebp), %edx           incl %eax
                                addl %eax, %edx               cmpl $17, %eax
                                movzbl (%ecx), %eax           je .L11
                                cmpb (%edx), %al             .L2:
                                jle .L6                       movzbl -1(%edi,%eax), %ecx
                                movl -4(%ebp), %eax           movzbl -1(%esi,%eax), %edx
                                movl 16(%ebp), %edx           cmpb %dl, %cl
                                addl %eax, %edx               jg .L12
                                movl -4(%ebp), %eax           movb %dl, -1(%eax,%ebx)
                                addl 8(%ebp), %eax            incl %eax
                                movzbl (%eax), %eax           cmpl $17, %eax
                                movb %al, (%edx)              jne .L2
                                jmp .L4                      .L11:
                               .L6:                           popl %ebx
                                movl -4(%ebp), %eax           popl %esi
                                movl 16(%ebp), %edx           popl %edi
                                addl %eax, %edx               popl %ebp
                                movl -4(%ebp), %eax           ret
                                addl 12(%ebp), %eax
                                movzbl (%eax), %eax
                                movb %al, (%edx)
                               .L4:
                                leal -4(%ebp), %eax
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

	`incl (%eax)`
	`jmp .L2`
	`.L1:`
	`leave`
	`ret`	

5.14.5 Benchmark

```
 unsigned char a[16] __attribute__((aligned));
 unsigned char b[16] __attribute__((aligned));
 unsigned char c[16] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = 100+i;
     b[i] = 100+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	35.218
GCC 4.0 - not optimized	27.098
GCC 4.1 - not optimized	28.426
ICC 8.1 - not optimized	29.588

GCC 4.0	9.415
GCC 4.1	9.171
ICC 8.1	7.099

GCC SIMD	1.219
ICC SIMD	1.674

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

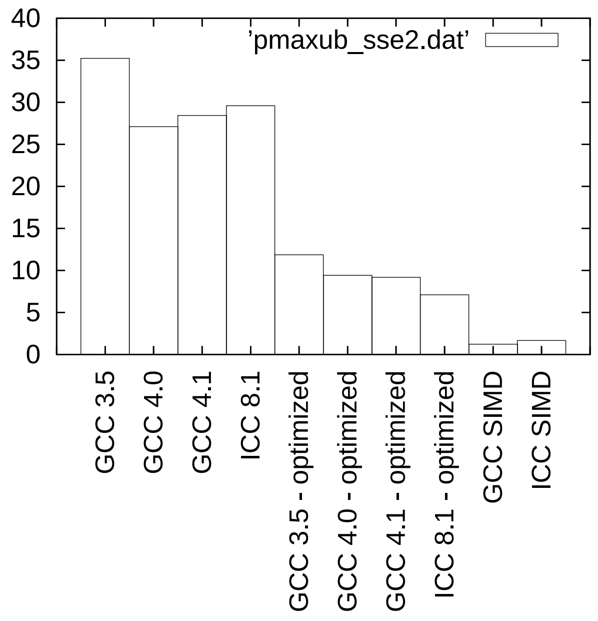

Figure 52: Benchmarks for pmaxub - SSE2 version

5.15 pminub - MMX (64 bits registers) version

5.15.1 C code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      if(a[i] < b[i])
        {
          c[i] = a[i];
        }
      else
        {
          c[i] = b[i];
        }
    }
}
```

5.15.2 GIMPLE code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 < T2)
    T3 = T1;
  else
```

```
    T3 = T2;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.15.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  *(__m64 *) c = _mm_min_pu8(*(__m64*) a,  *(__m64*) b);
}
```

5.15.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %eax`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`pushl %edi`
`movl 8(%ebp), %eax`	`.L2:`	`movl 8(%ebp), %edi`
`pminub (%eax), %mm0`	`cmpl $7, -4(%ebp)`	`pushl %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 12(%ebp), %esi`
`movq %mm0, (%eax)`	`jmp .L1`	`pushl %ebx`
`popl %ebp`	`.L5:`	`movl 16(%ebp), %ebx`
`ret`	`movl -4(%ebp), %eax`	`jmp .L2`
	`movl 8(%ebp), %ecx`	`.p2align 4,,7`
	`addl %eax, %ecx`	`.L12:`
	`movl -4(%ebp), %eax`	`movb %cl, -1(%eax,%ebx)`
	`movl 12(%ebp), %edx`	`incl %eax`
	`addl %eax, %edx`	`cmpl $9, %eax`
	`movzbl (%ecx), %eax`	`je .L11`
	`cmpb (%edx), %al`	`.L2:`
	`jae .L6`	`movzbl -1(%edi,%eax), %ecx`
	`movl -4(%ebp), %eax`	`movzbl -1(%esi,%eax), %edx`
	`movl 16(%ebp), %edx`	`cmpb %dl, %cl`
	`addl %eax, %edx`	`jb .L12`
	`movl -4(%ebp), %eax`	`movb %dl, -1(%eax,%ebx)`
	`addl 8(%ebp), %eax`	`incl %eax`
	`movzbl (%eax), %eax`	`cmpl $9, %eax`
	`movb %al, (%edx)`	`jne .L2`
	`jmp .L4`	`.L11:`
	`.L6:`	`popl %ebx`
	`movl -4(%ebp), %eax`	`popl %esi`
	`movl 16(%ebp), %edx`	`popl %edi`
	`addl %eax, %edx`	`popl %ebp`
	`movl -4(%ebp), %eax`	`ret`
	`addl 12(%ebp), %eax`	
	`movzbl (%eax), %eax`	
	`movb %al, (%edx)`	
	`.L4:`	
	`leal -4(%ebp), %eax`	

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

5.15.5 Benchmark

```
  unsigned char a[8] __attribute__((aligned));
  unsigned char b[8] __attribute__((aligned));
  unsigned char c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = 110+i;
      b[i] = 110+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	13.965
GCC 4.0 - not optimized	14.746
GCC 4.1 - not optimized	14.798
ICC 8.1 - not optimized	14.242

GCC 4.0	5.503
GCC 4.1	5.622
ICC 8.1	5.623

GCC SIMD	0.879
ICC SIMD	1.922

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

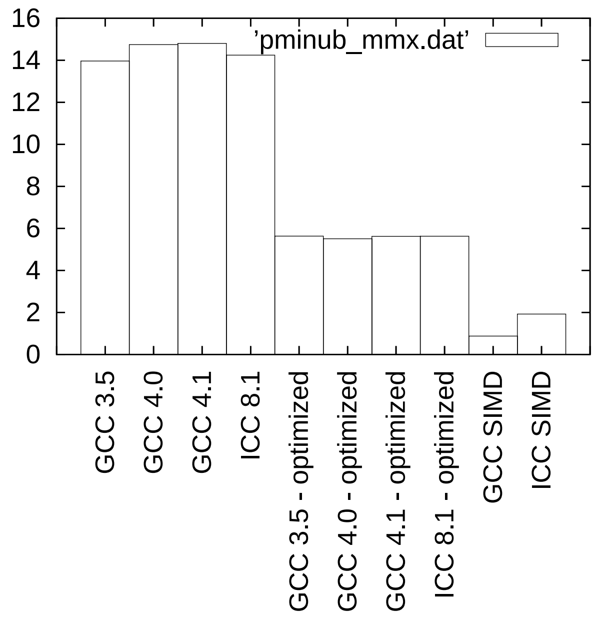

Figure 53: Benchmarks for pminub - MMX version

5.16 pminub - SSE2 (128 bits registers) version

5.16.1 C code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i;

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

5.16.2 GIMPLE code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i;
  loop_label::
  if(i >= 16)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 < T2)
    T3 = T1;
  else
```

```
    T3 = T2;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.16.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  *(__m64 *) c = _mm_min_epu8(*(__m64*) a,  *(__m64*) b);
}
```

5.16.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl $1, %eax
movl 12(%ebp), %eax	subl $4, %esp	movl %esp, %ebp
movdqa (%eax), %xmm0	movl $0, -4(%ebp)	pushl %edi
movl 8(%ebp), %eax	.L2:	movl 8(%ebp), %edi
pminub (%eax), %xmm0	cmpl $15, -4(%ebp)	pushl %esi
movl 16(%ebp), %eax	jle .L5	movl 12(%ebp), %esi
movdqa %xmm0, (%eax)	jmp .L1	pushl %ebx
popl %ebp	.L5:	movl 16(%ebp), %ebx
ret	movl -4(%ebp), %eax	jmp .L2
	movl 8(%ebp), %ecx	.p2align 4,,7
	addl %eax, %ecx	.L12:
	movl -4(%ebp), %eax	movb %cl, -1(%eax,%ebx)
	movl 12(%ebp), %edx	incl %eax
	addl %eax, %edx	cmpl $17, %eax
	movzbl (%ecx), %eax	je .L11
	cmpb (%edx), %al	.L2:
	jge .L6	movzbl -1(%edi,%eax), %ecx
	movl -4(%ebp), %eax	movzbl -1(%esi,%eax), %edx
	movl 16(%ebp), %edx	cmpb %dl, %cl
	addl %eax, %edx	jl .L12
	movl -4(%ebp), %eax	movb %dl, -1(%eax,%ebx)
	addl 8(%ebp), %eax	incl %eax
	movzbl (%eax), %eax	cmpl $17, %eax
	movb %al, (%edx)	jne .L2
	jmp .L4	.L11:
	.L6:	popl %ebx
	movl -4(%ebp), %eax	popl %esi
	movl 16(%ebp), %edx	popl %edi
	addl %eax, %edx	popl %ebp
	movl -4(%ebp), %eax	ret
	addl 12(%ebp), %eax	
	movzbl (%eax), %eax	
	movb %al, (%edx)	
	.L4:	
	leal -4(%ebp), %eax	

SIMD intrinsics	-nooptim	-O2 and vectorizer

	`incl (%eax)`
	`jmp .L2`
	`.L1:`
	`leave`
	`ret`	

5.16.5 Benchmark

```
 unsigned char a[16] __attribute__((aligned));
 unsigned char b[16] __attribute__((aligned));
 unsigned char c[16] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = 100+i;
     b[i] = 100+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	28.634
GCC 4.0 - not optimized	28.154
GCC 4.1 - not optimized	27.502
ICC 8.1 - not optimized	30.327

GCC 4.0	9.702
GCC 4.1	10.124
ICC 8.1	6.913

GCC SIMD	0.935
ICC SIMD	1.702

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

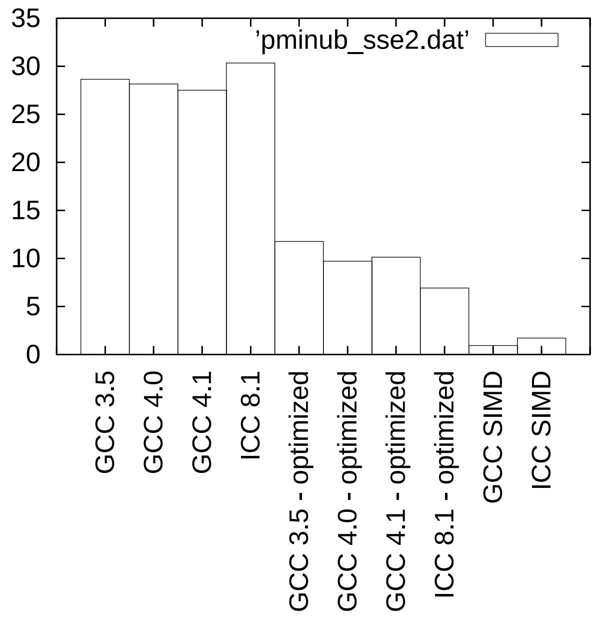

Figure 54: Benchmarks for pminub - SSE2 version

5.17 pmaxsw - MMX (64 bits registers) version

5.17.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      if(a[i] > b[i])
        {
          c[i] = a[i];
        }
      else
        {
          c[i] = b[i];
        }
    }
}
```

5.17.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = T1;
  else
```

```
    T3 = T2;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.17.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_max_pi16(*(__m64*) a,  *(__m64*) b);
}
```

5.17.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`pushl %esi`	`pushl %edi`
`movq (%eax), %mm0`	`pushl %ebx`	`movl 12(%ebp), %edi`
`movl 8(%ebp), %eax`	`subl $4, %esp`	`pushl %esi`
`pmaxsw (%eax), %mm0`	`movl $0, -12(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`.L2:`	`pushl %ebx`
`movq %mm0, (%eax)`	`cmpl $3, -12(%ebp)`	`movl $1, %ebx`
`popl %ebp`	`jle .L5`	`jmp .L2`
`ret`	`jmp .L1`	`.p2align 4,,7`
	`.L5:`	`.L12:`
	`movl -12(%ebp), %eax`	`incl %ebx`
	`leal (%eax,%eax), %esi`	`cmpl $5, %ebx`
	`movl 8(%ebp), %ebx`	`movw %cx, -2(%eax,%esi)`
	`movl -12(%ebp), %eax`	`je .L11`
	`leal (%eax,%eax), %ecx`	`.L2:`
	`movl 12(%ebp), %edx`	`movl 8(%ebp), %edx`
	`movzwl (%ebx,%esi), %eax`	`leal (%ebx,%ebx), %eax`
	`cmpw (%edx,%ecx), %ax`	`movswl -2(%edx,%eax),%ecx`
	`jle .L6`	`movswl -2(%edi,%eax),%edx`
	`movl -12(%ebp), %eax`	`cmpw %dx, %cx`
	`leal (%eax,%eax), %ebx`	`jg .L12`
	`movl 16(%ebp), %ecx`	`incl %ebx`
	`movl -12(%ebp), %eax`	`cmpl $5, %ebx`
	`leal (%eax,%eax), %edx`	`movw %dx, -2(%eax,%esi)`
	`movl 8(%ebp), %eax`	`jne .L2`
	`movzwl (%eax,%edx), %eax`	`.L11:`
	`movw %ax, (%ecx,%ebx)`	`popl %ebx`
	`jmp .L4`	`popl %esi`
	`.L6:`	`popl %edi`
	`movl -12(%ebp), %eax`	`popl %ebp`
	`leal (%eax,%eax), %ebx`	`ret`
	`movl 16(%ebp), %ecx`	
	`movl -12(%ebp), %eax`	
	`leal (%eax,%eax), %edx`	
	`movl 12(%ebp), %eax`	

SIMD intrinsics	-nooptim	-O2 and vectorizer

```
 movzwl (%eax,%edx), %eax
 movw %ax, (%ecx,%ebx)
.L4:
 leal -12(%ebp), %eax
 incl (%eax)
 jmp .L2
.L1:
 addl $4, %esp
 popl %ebx
 popl %esi
 popl %ebp
 ret
```

5.17.5 Benchmark

```
  short int a[4] __attribute__((aligned));
  short int b[4] __attribute__((aligned));
  short int c[4] __attribute__((aligned));
  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = 16000+i;
      b[i] = 16000+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	9.435
GCC 4.0 - not optimized	10.254
GCC 4.1 - not optimized	10.37
ICC 8.1 - not optimized	9.582
GCC 4.0	5.681
GCC 4.1	5.283
ICC 8.1	4.411
GCC SIMD	1.263
ICC SIMD	1.755

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

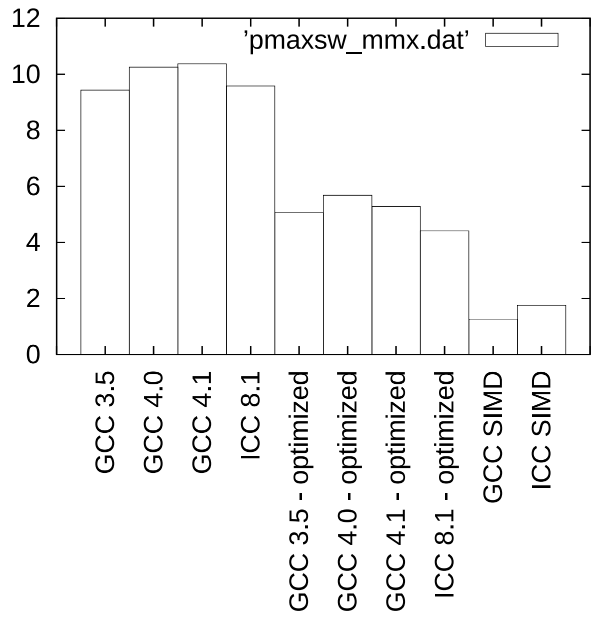

Figure 55: Benchmarks for pmaxsw - MMX version

5.18 pmaxsw - SSE2 (128 bits registers) version

5.18.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      if(a[i] > b[i])
        {
          c[i] = a[i];
        }
      else
        {
          c[i] = b[i];
        }
    }
}
```

5.18.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 > T2)
    T3 = T1;
  else
```

```
    T3 = T2;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.18.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_max_epi16(*(__m128i*) a,  *(__m128i*) b);
}
```

5.18.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                      -O2 and vectorizer
  pushl %ebp                   pushl %ebp                    pushl %ebp
  movl %esp, %ebp              movl %esp, %ebp               movl %esp, %ebp
  movl 12(%ebp), %eax          pushl %esi                    pushl %edi
  movdqa (%eax), %xmm0         pushl %ebx                    movl 12(%ebp), %edi
  movl 8(%ebp), %eax           subl $4, %esp                 pushl %esi
  pmaxsw (%eax), %xmm0         movl $0, -12(%ebp)            movl 16(%ebp), %esi
  movl 16(%ebp), %eax         .L2:                           pushl %ebx
  movdqa %xmm0, (%eax)         cmpl $7, -12(%ebp)            movl $1, %ebx
  popl %ebp                    jle .L5                       jmp .L2
  ret                          jmp .L1                       .p2align 4,,7
                              .L5:                          .L12:
                               movl -12(%ebp), %eax          incl %ebx
                               leal (%eax,%eax), %esi        cmpl $9, %ebx
                               movl 8(%ebp), %ebx            movw %cx, -2(%eax,%esi)
                               movl -12(%ebp), %eax          je .L11
                               leal (%eax,%eax), %ecx       .L2:
                               movl 12(%ebp), %edx           movl 8(%ebp), %edx
                               movzwl (%ebx,%esi), %eax      leal (%ebx,%ebx), %eax
                               cmpw (%edx,%ecx), %ax         movswl -2(%edx,%eax),%ecx
                               jle .L6                       movswl -2(%edi,%eax),%edx
                               movl -12(%ebp), %eax          cmpw %dx, %cx
                               leal (%eax,%eax), %ebx        jg .L12
                               movl 16(%ebp), %ecx           incl %ebx
                               movl -12(%ebp), %eax          cmpl $9, %ebx
                               leal (%eax,%eax), %edx        movw %dx, -2(%eax,%esi)
                               movl 8(%ebp), %eax            jne .L2
                               movzwl (%eax,%edx), %eax     .L11:
                               movw %ax, (%ecx,%ebx)         popl %ebx
                               jmp .L4                       popl %esi
                              .L6:                           popl %edi
                               movl -12(%ebp), %eax          popl %ebp
                               leal (%eax,%eax), %ebx        ret
                               movl 16(%ebp), %ecx
                               movl -12(%ebp), %eax
                               leal (%eax,%eax), %edx
                               movl 12(%ebp), %eax
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

```
 movzwl (%eax,%edx), %eax
 movw %ax, (%ecx,%ebx)
.L4:
 leal -12(%ebp), %eax
 incl (%eax)
 jmp .L2
.L1:
 addl $4, %esp
 popl %ebx
 popl %esi
 popl %ebp
 ret
```

5.18.5 Benchmark

```
 short int a[8] __attribute__((aligned));
 short int b[8] __attribute__((aligned));
 short int c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = 16000+i;
     b[i] = 16000+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	16.545
GCC 4.0 - not optimized	18.687
GCC 4.1 - not optimized	19.325
ICC 8.1 - not optimized	16.997
GCC 4.0	8.578
GCC 4.1	8.448
ICC 8.1	3.113
GCC SIMD	1.443
ICC SIMD	2.354

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	`pmaxsw` vectorization+

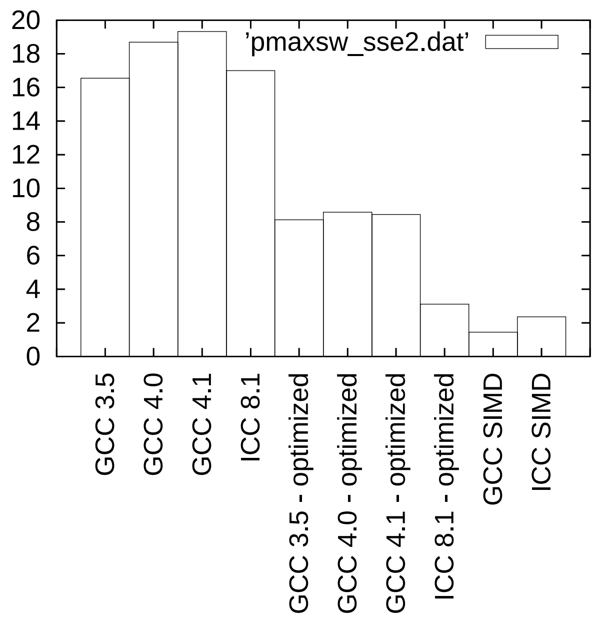

Figure 56: Benchmarks for pmaxw - SSE2 version

5.19 pminsw - MMX (64 bits registers) version

5.19.1 C code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      if(a[i] < b[i])
        {
          c[i] = a[i];
        }
      else
        {
          c[i] = b[i];
        }
    }
}
```

5.19.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], short int c[4])
{
  int i;
  loop_label::
  if(i >= 4)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 < T2)
    T3 = T1;
  else
```

```
    T3 = T2;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.19.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], short int c[4])
{
  *(__m64 *) c = _mm_min_pi16(*(__m64*) a,  *(__m64*) b);
}
```

5.19.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	movl %esp, %ebp
movl 12(%ebp), %eax	pushl %esi	pushl %edi
movq (%eax), %mm0	pushl %ebx	movl 12(%ebp), %edi
movl 8(%ebp), %eax	subl $4, %esp	pushl %esi
pminsw (%eax), %mm0	movl $0, -12(%ebp)	movl 16(%ebp), %esi
movl 16(%ebp), %eax	.L2:	pushl %ebx
movq %mm0, (%eax)	cmpl $3, -12(%ebp)	movl $1, %ebx
popl %ebp	jle .L5	jmp .L2
ret	jmp .L1	.p2align 4,,7
	.L5:	.L12:
	movl -12(%ebp), %eax	incl %ebx
	leal (%eax,%eax), %esi	cmpl $5, %ebx
	movl 8(%ebp), %ebx	movw %cx, -2(%eax,%esi)
	movl -12(%ebp), %eax	je .L11
	leal (%eax,%eax), %ecx	.L2:
	movl 12(%ebp), %edx	movl 8(%ebp), %edx
	movzwl (%ebx,%esi), %eax	leal (%ebx,%ebx), %eax
	cmpw (%edx,%ecx), %ax	movswl -2(%edx,%eax),%ecx
	jge .L6	movswl -2(%edi,%eax),%edx
	movl -12(%ebp), %eax	cmpw %dx, %cx
	leal (%eax,%eax), %ebx	jl .L12
	movl 16(%ebp), %ecx	incl %ebx
	movl -12(%ebp), %eax	cmpl $5, %ebx
	leal (%eax,%eax), %edx	movw %dx, -2(%eax,%esi)
	movl 8(%ebp), %eax	jne .L2
	movzwl (%eax,%edx), %eax	.L11:
	movw %ax, (%ecx,%ebx)	popl %ebx
	jmp .L4	popl %esi
	.L6:	popl %edi
	movl -12(%ebp), %eax	popl %ebp
	leal (%eax,%eax), %ebx	ret
	movl 16(%ebp), %ecx	
	movl -12(%ebp), %eax	
	leal (%eax,%eax), %edx	
	movl 12(%ebp), %eax	

SIMD intrinsics	-nooptim	-O2 and vectorizer
	movzwl (%eax,%edx), %eax	
	movw %ax, (%ecx,%ebx)	
	.L4:	
	leal -12(%ebp), %eax	
	incl (%eax)	
	jmp .L2	
	.L1:	
	addl $4, %esp	
	popl %ebx	
	popl %esi	
	popl %ebp	
	ret	

5.19.5 Benchmark

```
  short int a[4] __attribute__((aligned));
  short int b[4] __attribute__((aligned));
  short int c[4] __attribute__((aligned));
  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = 16000+i;
      b[i] = 16000+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	6.904
GCC 4.0 - not optimized	7.701
GCC 4.1 - not optimized	8.056
ICC 8.1 - not optimized	7.667

GCC 4.0	4.009
GCC 4.1	3.886
ICC 8.1	3.236

GCC SIMD	1.002
ICC SIMD	1.367

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

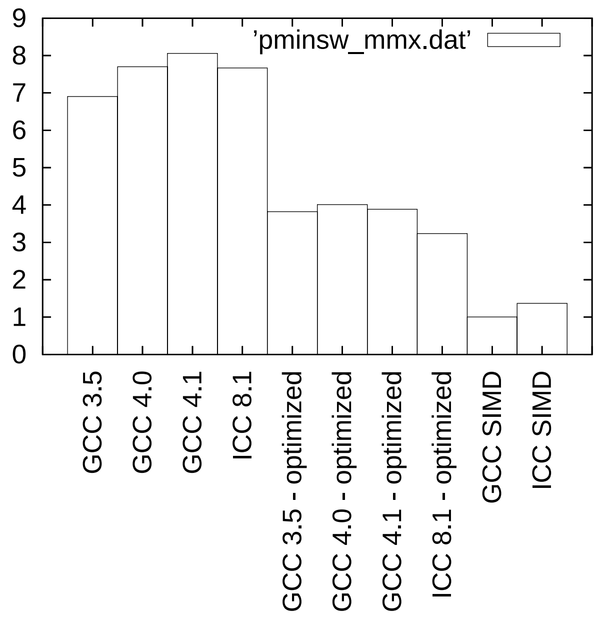

Figure 57: Benchmarks for pminsw - MMX version

5.20 pminsw - SSE2 (128 bits registers) version

5.20.1 C code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      if(a[i] < b[i])
        {
          c[i] = a[i];
        }
      else
        {
          c[i] = b[i];
        }
    }
}
```

5.20.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], short int c[8])
{
  int i;
  loop_label::
  if(i >= 8)
    goto break_label;
  T1 = a[i];
  T2 = b[i];
  if(T1 < T2)
    T3 = T1;
  else
```

```
    T3 = T2;
  c[i] = T3;
  i = i + 1;
  goto loop_label;
  break_label::;
}
```

5.20.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], short int c[8])
{
  *(__m128i *) c = _mm_min_epi16(*(__m128i*) a,  *(__m128i*) b);
}
```

5.20.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics              -nooptim                      -O2 and vectorizer
 ---------------------------------------------------------------------------------------
  pushl %ebp                   pushl %ebp                    pushl %ebp
  movl %esp, %ebp              movl %esp, %ebp               movl %esp, %ebp
  movl 12(%ebp), %eax          pushl %esi                    pushl %edi
  movdqa (%eax), %xmm0         pushl %ebx                    movl 12(%ebp), %edi
  movl 8(%ebp), %eax           subl $4, %esp                 pushl %esi
  pminsw (%eax), %xmm0         movl $0, -12(%ebp)            movl 16(%ebp), %esi
  movl 16(%ebp), %eax         .L2:                           pushl %ebx
  movdqa %xmm0, (%eax)         cmpl $7, -12(%ebp)            movl $1, %ebx
  popl %ebp                    jle .L5                       jmp .L2
  ret                          jmp .L1                       .p2align 4,,7
                              .L5:                          .L12:
                               movl -12(%ebp), %eax          incl %ebx
                               leal (%eax,%eax), %esi        cmpl $9, %ebx
                               movl 8(%ebp), %ebx            movw %cx, -2(%eax,%esi)
                               movl -12(%ebp), %eax          je .L11
                               leal (%eax,%eax), %ecx       .L2:
                               movl 12(%ebp), %edx           movl 8(%ebp), %edx
                               movzwl (%ebx,%esi), %eax      leal (%ebx,%ebx), %eax
                               cmpw (%edx,%ecx), %ax         movswl -2(%edx,%eax),%ecx
                               jge .L6                       movswl -2(%edi,%eax),%edx
                               movl -12(%ebp), %eax          cmpw %dx, %cx
                               leal (%eax,%eax), %ebx        jl .L12
                               movl 16(%ebp), %ecx           incl %ebx
                               movl -12(%ebp), %eax          cmpl $9, %ebx
                               leal (%eax,%eax), %edx        movw %dx, -2(%eax,%esi)
                               movl 8(%ebp), %eax            jne .L2
                               movzwl (%eax,%edx), %eax     .L11:
                               movw %ax, (%ecx,%ebx)         popl %ebx
                               jmp .L4                       popl %esi
                              .L6:                           popl %edi
                               movl -12(%ebp), %eax          popl %ebp
                               leal (%eax,%eax), %ebx        ret
                               movl 16(%ebp), %ecx
                               movl -12(%ebp), %eax
                               leal (%eax,%eax), %edx
                               movl 12(%ebp), %eax
```

SIMD intrinsics	-nooptim	-O2 and vectorizer
	` movzwl (%eax,%edx), %eax`	
	` movw %ax, (%ecx,%ebx)`	
	`.L4:`	
	` leal -12(%ebp), %eax`	
	` incl (%eax)`	
	` jmp .L2`	
	`.L1:`	
	` addl $4, %esp`	
	` popl %ebx`	
	` popl %esi`	
	` popl %ebp`	
	` ret`	

5.20.5 Benchmark

```
  short int a[8] __attribute__((aligned));
  short int b[8] __attribute__((aligned));
  short int c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<16; i++)
    {
      a[i] = 16000+i;
      b[i] = 16000+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	13.205
GCC 4.0 - not optimized	14.717
GCC 4.1 - not optimized	14.749
ICC 8.1 - not optimized	14.342

GCC 4.0	6.633
GCC 4.1	6.029
ICC 8.1	2.989

GCC SIMD	1.139
ICC SIMD	2.025

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	`pminsw` vectorization+

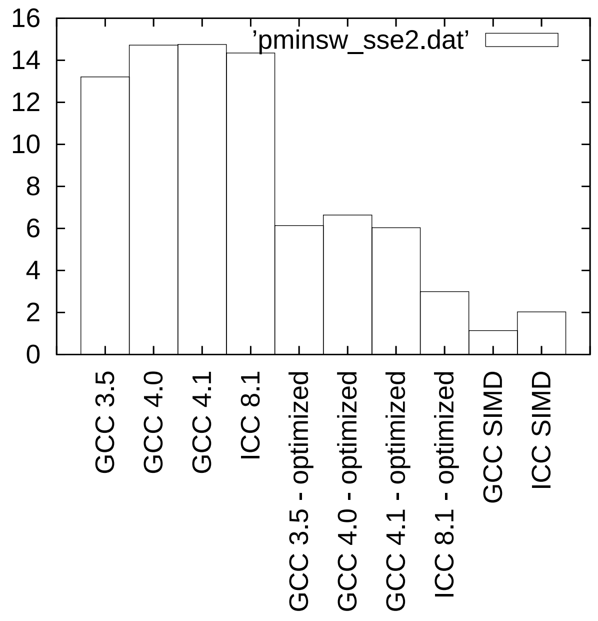

Figure 58: Benchmarks for pminw - SSE2 version

6 Complex operations

6.1 pavgb - MMX (64 bits registers) version

6.1.1 C code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i;

  for(i=0; i<8; i++)
    {
      c[i] = (a[i] + b[i] + 1) >> 1;
    }
}
```

6.1.2 GIMPLE code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  int i=0;
  loop_label::
  if(i >= 8)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  t4 = t3 + 1;
  t5 = t4 >> 1;
  c[i] = t5;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

6.1.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  *(__m64 *) c = _mm_avg_pu8(*(__m64*) a,  *(__m64*) b);
}
```

6.1.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %ecx`
`movl 12(%ebp), %eax`	`subl $4, %esp`	`movl %esp, %ebp`
`movq (%eax), %mm0`	`movl $0, -4(%ebp)`	`pushl %edi`
`movl 8(%ebp), %eax`	`.L2:`	`movl 8(%ebp), %edi`
`pavgb (%eax), %mm0`	`cmpl $7, -4(%ebp)`	`pushl %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 12(%ebp), %esi`
`movq %mm0, (%eax)`	`jmp .L1`	`pushl %ebx`
`popl %ebp`	`.L5:`	`movl 16(%ebp), %ebx`
`ret`	`movl -4(%ebp), %eax`	`.p2align 4,,15`
	`movl 16(%ebp), %ecx`	`.L2:`
	`addl %eax, %ecx`	`movzbl -1(%edi,%ecx), %eax`
	`movl -4(%ebp), %eax`	`movzbl -1(%esi,%ecx), %edx`
	`addl 8(%ebp), %eax`	`leal 1(%eax,%edx), %eax`
	`movzbl (%eax), %edx`	`sarl %eax`
	`movl -4(%ebp), %eax`	`movb %al, -1(%ebx,%ecx)`
	`addl 12(%ebp), %eax`	`incl %ecx`
	`movzbl (%eax), %eax`	`cmpl $9, %ecx`
	`leal (%eax,%edx), %eax`	`jne .L2`
	`incl %eax`	`popl %ebx`
	`sarl %eax`	`popl %esi`
	`movb %al, (%ecx)`	`popl %edi`
	`leal -4(%ebp), %eax`	`popl %ebp`
	`incl (%eax)`	`ret`
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

6.1.5 Benchmark

```
  unsigned char a[8] __attribute__((aligned));
  unsigned char b[8] __attribute__((aligned));
  unsigned char c[8] __attribute__((aligned));
  int i;

  for(i = 0; i<8; i++)
    {
      a[i] = i;
      b[i] = 10+2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	11.14
GCC 4.0 - not optimized	15.06
GCC 4.1 - not optimized	14.687
ICC 8.1 - not optimized	11.162
GCC 4.0	4.932
GCC 4.1	5.546
ICC 8.1	4.685
GCC SIMD	0.95
ICC SIMD	1.962

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

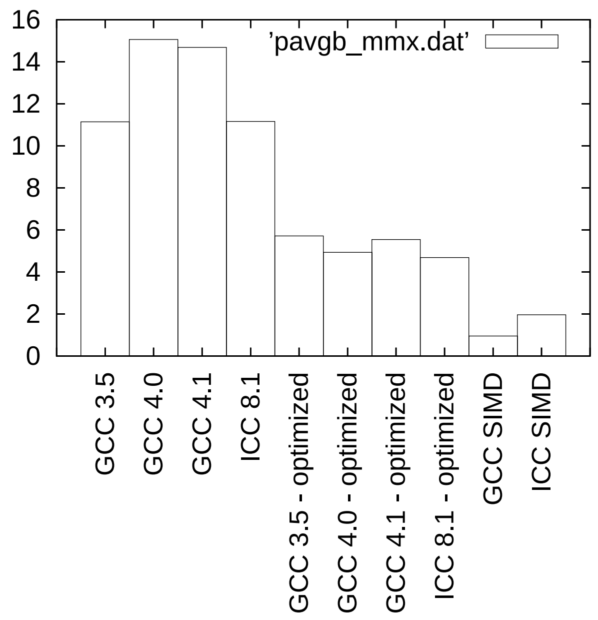

Figure 59: Benchmarks for pavgb - MMX version

6.2 pavgb - SSE2 (128 bits registers) version

6.2.1 C code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i;

  for(i=0; i<16; i++)
    {
      c[i] = (a[i] + b[i] + 1) >> 1;
    }
}
```

6.2.2 GIMPLE code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  int i=0;
  loop_label::
  if(i >= 16)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  t4 = t3 + 1;
  t5 = t4 >> 1;
  c[i] = t5;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

6.2.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[16], unsigned char b[16], unsigned char c[16])
{
  *(__m128i *) c = _mm_avg_epu8(*(__m128i *) a,  *(__m128i *) b);
}
```

6.2.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl $1, %ecx`
`movl 12(%ebp), %eax`	`subl $8, %esp`	`movl %esp, %ebp`
`movdqa (%eax), %xmm0`	`movl $0, -4(%ebp)`	`pushl %edi`
`movl 8(%ebp), %eax`	`.L2:`	`movl 8(%ebp), %edi`
`pavgb (%eax), %xmm0`	`cmpl $15, -4(%ebp)`	`pushl %esi`
`movl 16(%ebp), %eax`	`jle .L5`	`movl 12(%ebp), %esi`
`movdqa %xmm0, (%eax)`	`jmp .L1`	`pushl %ebx`
`popl %ebp`	`.L5:`	`movl 16(%ebp), %ebx`
`ret`	`movl -4(%ebp), %eax`	`.p2align 4,,15`
	`movl 16(%ebp), %ecx`	`.L2:`
	`addl %eax, %ecx`	`movzbl -1(%edi,%ecx), %eax`
	`movl -4(%ebp), %eax`	`movzbl -1(%esi,%ecx), %edx`
	`addl 8(%ebp), %eax`	`leal 1(%eax,%edx), %eax`
	`movzbl (%eax), %edx`	`sarl %eax`
	`movl -4(%ebp), %eax`	`movb %al, -1(%ebx,%ecx)`
	`addl 12(%ebp), %eax`	`incl %ecx`
	`movzbl (%eax), %eax`	`cmpl $17, %ecx`
	`leal (%eax,%edx), %eax`	`jne .L2`
	`incl %eax`	`popl %ebx`
	`sarl %eax`	`popl %esi`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`movb %al, (%ecx)`	`popl %edi`
	`leal -4(%ebp), %eax`	`popl %ebp`
	`incl (%eax)`	`ret`
	`jmp .L2`	
	`.L1:`	
	`leave`	
	`ret`	

6.2.5 Benchmark

```
 unsigned char a[16] __attribute__((aligned));
 unsigned char b[16] __attribute__((aligned));
 unsigned char c[16] __attribute__((aligned));
 int i;

 for(i = 0; i<16; i++)
   {
     a[i] = i;
     b[i] = 10+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
 test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	26.782
GCC 4.0 - not optimized	29.993
GCC 4.1 - not optimized	30.277
ICC 8.1 - not optimized	22.596

GCC 4.0	9.766
GCC 4.1	9.561
ICC 8.1	2.348

GCC SIMD	1.1
ICC SIMD	1.709

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `pavgb`

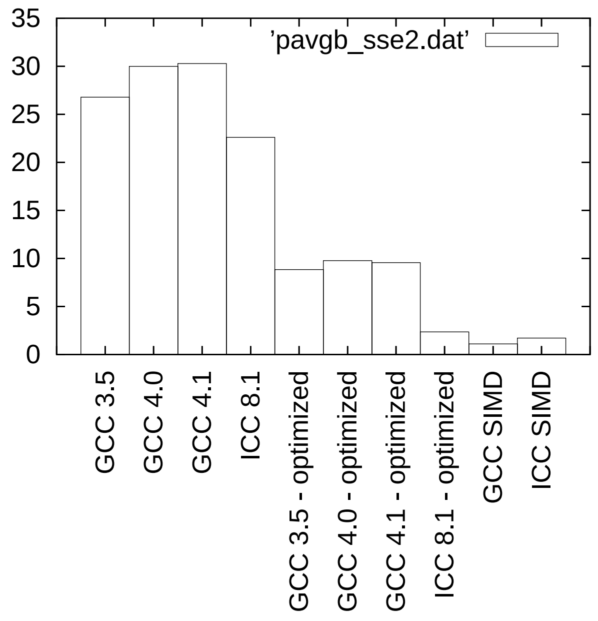

Figure 60: Benchmarks for pavgb - SSE2 version

6.3 pavgw - MMX (64 bits registers) version

6.3.1 C code

```
void test_loop_c(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      c[i] = (a[i] + b[i] + 1) >> 1;
    }
}
```

6.3.2 GIMPLE code

```
void test_loop_c(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  int i=0;
  loop_label::
  if(i >= 4)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  t4 = t3 + 1;
  t5 = t4 >> 1;
  c[i] = t5;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

6.3.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned short int a[4], unsigned short int b[4], unsigned short int c[4])
{
  *(__m64 *) c = _mm_avg_pu16(*(__m64 *) a, *(__m64 *) b);
}
```

6.3.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`pushl %esi`	`pushl %edi`
`movq (%eax), %mm0`	`pushl %ebx`	`movl 12(%ebp), %edi`
`movl 8(%ebp), %eax`	`subl $8, %esp`	`pushl %esi`
`pavgw (%eax), %mm0`	`movl $0, -12(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`.L2:`	`pushl %ebx`
`movq %mm0, (%eax)`	`cmpl $3, -12(%ebp)`	`movl $1, %ebx`
`popl %ebp`	`jle .L5`	`.p2align 4,,15`
`ret`	`jmp .L1`	`.L2:`
	`.L5:`	`movl 8(%ebp), %ecx`
	`movl -12(%ebp), %eax`	`leal (%ebx,%ebx), %eax`
	`leal (%eax,%eax), %ebx`	`incl %ebx`
	`movl 16(%ebp), %esi`	`movzwl -2(%eax,%ecx), %edx`
	`movl -12(%ebp), %eax`	`movzwl -2(%eax,%edi), %ecx`
	`leal (%eax,%eax), %edx`	`leal 1(%edx,%ecx), %edx`
	`movl 8(%ebp), %eax`	`sarl %edx`
	`movzwl (%eax,%edx), %ecx`	`cmpl $5, %ebx`
	`movl -12(%ebp), %eax`	`movw %dx, -2(%eax,%esi)`
	`leal (%eax,%eax), %edx`	`jne .L2`
	`movl 12(%ebp), %eax`	`popl %ebx`
	`movzwl (%eax,%edx), %eax`	`popl %esi`
	`leal (%eax,%ecx), %eax`	`popl %edi`
	`incl %eax`	`popl %ebp`
	`sarl %eax`	`ret`
	`movw %ax, (%esi,%ebx)`	
	`leal -12(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`addl $8, %esp`	
	`popl %ebx`	
	`popl %esi`	
	`popl %ebp`	
	`ret`	

6.3.5 Benchmark

```
  unsigned short int a[4] __attribute__((aligned));
  unsigned short int b[4] __attribute__((aligned));
  unsigned short int c[4] __attribute__((aligned));

  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = 140 + i;
      b[i] = 140 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	6.027
GCC 4.0 - not optimized	6.053
GCC 4.1 - not optimized	6.722
ICC 8.1 - not optimized	5.87
GCC 4.0	3.882
GCC 4.1	3.754
ICC 8.1	2.518
GCC SIMD	0.877
ICC SIMD	1.356

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

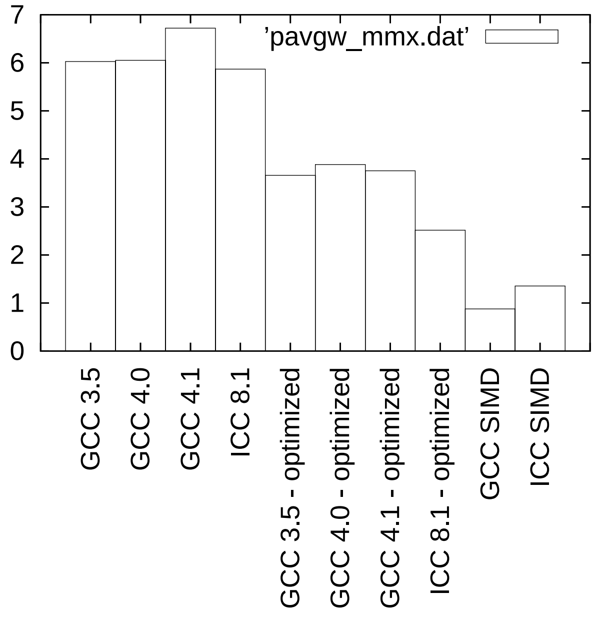

Figure 61: Benchmarks for pavgw - MMX version

6.4 pavgw - SSE2 (128 bits registers) version

6.4.1 C code

```
void test_loop_c(unsigned short int a[8], unsigned short int b[8], unsigned short int c[8])
{
  int i;
  for(i=0; i<8; i++)
    {
      c[i] = (a[i] + b[i] + 1) >> 1;
    }
}
```

6.4.2 GIMPLE code

```
void test_loop_c(unsigned short int a[8], unsigned short int b[8], unsigned short int c[8])
{
  int i=0;
  loop_label::
  if(i >= 8)
    goto break_label;
  t1 = a[i];
  t2 = b[i];
  t3 = t1 + t2;
  t4 = t3 + 1;
  t5 = t4 >> 1;
  c[i] = t5;
  i = i + 1;
  goto loop_label;

  break_label:;
}
```

6.4.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned short int a[8], unsigned short int b[8], unsigned short int c[8])
{
  *(__m128i *) c = _mm_avg_epu16(*(__m128i *) a, *(__m128i *) b);
}
```

6.4.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
`pushl %ebp`	`pushl %ebp`	`pushl %ebp`
`movl %esp, %ebp`	`movl %esp, %ebp`	`movl %esp, %ebp`
`movl 12(%ebp), %eax`	`pushl %esi`	`pushl %edi`
`movdqa (%eax), %xmm0`	`pushl %ebx`	`movl 12(%ebp), %edi`
`movl 8(%ebp), %eax`	`subl $4, %esp`	`pushl %esi`
`pavgw (%eax), %xmm0`	`movl $0, -12(%ebp)`	`movl 16(%ebp), %esi`
`movl 16(%ebp), %eax`	`.L2:`	`pushl %ebx`
`movdqa %xmm0, (%eax)`	`cmpl $7, -12(%ebp)`	`movl $1, %ebx`
`popl %ebp`	`jle .L5`	`.p2align 4,,15`
`ret`	`jmp .L1`	`.L2:`
	`.L5:`	`movl 8(%ebp), %ecx`
	`movl -12(%ebp), %eax`	`leal (%ebx,%ebx), %eax`
	`leal (%eax,%eax), %ebx`	`incl %ebx`
	`movl 16(%ebp), %esi`	`movzwl -2(%eax,%ecx), %edx`
	`movl -12(%ebp), %eax`	`movzwl -2(%eax,%edi), %ecx`
	`leal (%eax,%eax), %edx`	`leal 1(%edx,%ecx), %edx`
	`movl 8(%ebp), %eax`	`sarl %edx`

SIMD intrinsics	-nooptim	-O2 and vectorizer
	`movzwl (%eax,%edx), %ecx`	`cmpl $9, %ebx`
	`movl -12(%ebp), %eax`	`movw %dx, -2(%eax,%esi)`
	`leal (%eax,%eax), %edx`	`jne .L2`
	`movl 12(%ebp), %eax`	`popl %ebx`
	`movzwl (%eax,%edx), %eax`	`popl %esi`
	`leal (%eax,%ecx), %eax`	`popl %edi`
	`incl %eax`	`popl %ebp`
	`sarl %eax`	`ret`
	`movw %ax, (%esi,%ebx)`	
	`leal -12(%ebp), %eax`	
	`incl (%eax)`	
	`jmp .L2`	
	`.L1:`	
	`addl $4, %esp`	
	`popl %ebx`	
	`popl %esi`	
	`popl %ebp`	
	`ret`	

6.4.5 Benchmark

```
 unsigned short int a[8] __attribute__((aligned));
 unsigned short int b[8] __attribute__((aligned));
 unsigned short int c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 140 + i;
     b[i] = 140 + 2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	10.172
GCC 4.0 - not optimized	12.623
GCC 4.1 - not optimized	12.963
ICC 8.1 - not optimized	11.346
GCC 4.0	5.82
GCC 4.1	6.203
ICC 8.1	3.205
GCC SIMD	0.847
ICC SIMD	2.603

GCC 4.0 behavior	unrolling and vectorization
GCC 4.1 behavior	unrolling and vectorization
ICC behavior	vectorization with `pavgw`

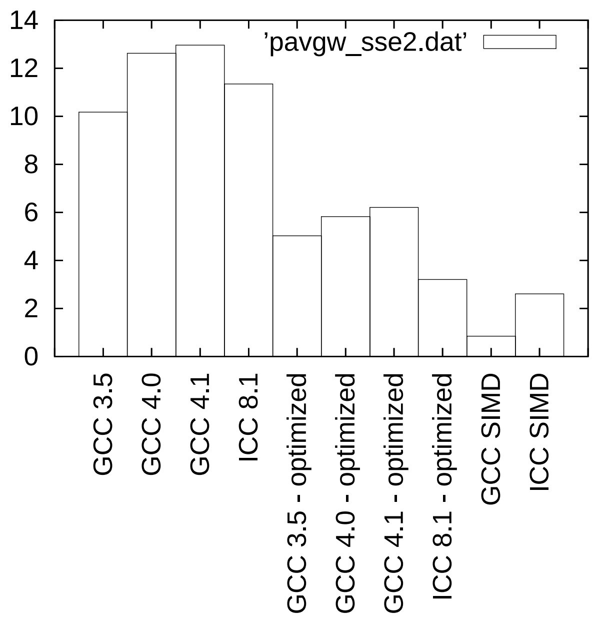

Figure 62: Benchmarks for pavgw - SSE2 version

6.5 pmaddwd - MMX (64 bits registers) version

6.5.1 C code

```
void test_loop_c(short int a[4], short int b[4], int c[2])
{
  int i;

  for(i=0; i<2; i++)
    {
      c[i] = a[2*i] * b[2*i] + a[2*i+1] * b[2*i + 1];
    }
}
```

6.5.2 GIMPLE code

```
void test_loop_c(short int a[4], short int b[4], int c[2])
{
  int i=0;
  loop_label::
  if(i >= 2)
    goto break_label;
  t1 = 2*i;
  t2 = a[t1];
  t3 = b[t2];
  t4 = t2 + t3;
  t5 = t1 + 1;
  t6 = a[t5];
  t7 = b[t6];
  t8 = t5 + t6;
  t9 = t4 + t8;
  c[i] = t9;
  i = i + 1;
  goto loop_label;
  break_label:;
}
```

6.5.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[4], short int b[4], int c[2])
{
  *(__m64 *) c = _mm_madd_pi16(*(__m64 *) a, *(__m64 *) b);
}
```

6.5.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics             -nooptim                     -O2 and vectorizer
  pushl %ebp                  pushl %ebp                   pushl %ebp
  movl %esp, %ebp             movl %esp, %ebp              movl %esp, %ebp
  movl 8(%ebp), %eax          pushl %esi                   pushl %edi
  movq (%eax), %mm1           pushl %ebx                   movl 16(%ebp), %edi
  movl 12(%ebp), %eax         subl $8, %esp                pushl %esi
  movq (%eax), %mm0           movl $0, -12(%ebp)           movl $1, %esi
  movl 16(%ebp), %eax        .L2:                          pushl %ebx
  pmaddwd %mm0, %mm1          cmpl $1, -12(%ebp)           .p2align 4,,15
  movq %mm1, (%eax)           jle .L5                     .L2:
  popl %ebp                   jmp .L1                      movl 12(%ebp), %eax
  ret                        .L5:                          leal 0(,%esi,4), %edx
                              movl -12(%ebp), %eax         movl 8(%ebp), %ebx
                              leal 0(,%eax,4), %ebx        addl %edx, %ebx
                              movl 16(%ebp), %esi          addl %eax, %edx
                              movl -12(%ebp), %eax         movswl -4(%edx),%eax
                              leal 0(,%eax,4), %edx        movswl -4(%ebx),%ecx
                              movl 8(%ebp), %eax           movswl -2(%edx),%edx
                              movswl (%eax,%edx),%ecx      imull %eax, %ecx
                              movl -12(%ebp), %eax         movswl -2(%ebx),%eax
                              leal 0(,%eax,4), %edx        imull %edx, %eax
                              movl 12(%ebp), %eax          addl %eax, %ecx
                              movswl (%eax,%edx),%eax      movl %ecx, -4(%edi,%esi,4)
                              imull %eax, %ecx             incl %esi
                              movl -12(%ebp), %eax         cmpl $3, %esi
                              sall $2, %eax                jne .L2
                              addl 8(%ebp), %eax           popl %ebx
                              addl $2, %eax                popl %esi
                              movswl (%eax),%edx           popl %edi
                              movl -12(%ebp), %eax         popl %ebp
                              sall $2, %eax                ret
                              addl 12(%ebp), %eax
                              addl $2, %eax
                              movswl (%eax),%eax
                              imull %edx, %eax
                              leal (%eax,%ecx), %eax
                              movl %eax, (%esi,%ebx)
                              leal -12(%ebp), %eax
                              incl (%eax)
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

	`jmp .L2`
	`.L1:`
	`addl $8, %esp`
	`popl %ebx`
	`popl %esi`
	`popl %ebp`
	`ret`	

6.5.5 Benchmark

```
 short int a[4] __attribute__((aligned));
 short int b[4] __attribute__((aligned));
 int c[2] __attribute__((aligned));

 int i;

 for(i = 0; i<4; i++)
   {
     a[i] = 140 + i;
     b[i] = 140 + 2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
     test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	9.026
GCC 4.0 - not optimized	8.629
GCC 4.1 - not optimized	8.565
ICC 8.1 - not optimized	7.079

GCC 4.0	5.288
GCC 4.1	5.86
ICC 8.1	3.88

GCC SIMD	1.812
ICC SIMD	1.742

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

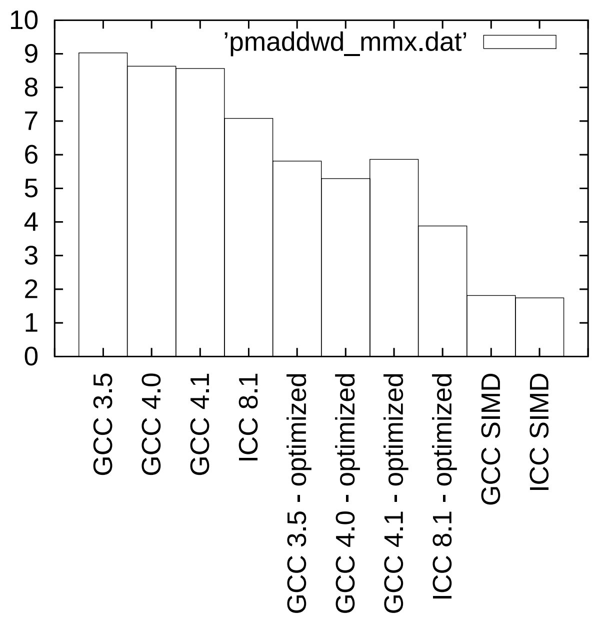

Figure 63: Benchmarks for pmaddwd - MMX version

6.6 pmaddwd - SSE2 (128 bits registers) version

6.6.1 C code

```
void test_loop_c(short int a[8], short int b[8], int c[4])
{
  int i;

  for(i=0; i<4; i++)
    {
      c[i] = a[2*i] * b[2*i] + a[2*i+1] * b[2*i + 1];
    }
}
```

6.6.2 GIMPLE code

```
void test_loop_c(short int a[8], short int b[8], int c[4])
{
  int i=0;
  loop_label::
  if(i >= 4)
    goto break_label;
  t1 = 2*i;
  t2 = a[t1];
  t3 = b[t2];
  t4 = t2 + t3;
  t5 = t1 + 1;
  t6 = a[t5];
  t7 = b[t6];
  t8 = t5 + t6;
  t9 = t4 + t8;
  c[i] = t9;
  i = i + 1;
  goto loop_label;
  break_label:;
}
```

6.6.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(short int a[8], short int b[8], int c[4])
{
  *(__m128i *) c = _mm_madd_epi16(*(__m128i *) a, *(__m128i *) b);
}
```

6.6.4 Assembly code

The assembly codes that are generated are the following:

```
 SIMD intrinsics             -nooptim                    -O2 and vectorizer
  pushl %ebp                  pushl %ebp                  pushl %ebp
  movl %esp, %ebp             movl %esp, %ebp             movl %esp, %ebp
  movl 8(%ebp), %eax          pushl %esi                  pushl %edi
  movdqa (%eax), %xmm1        pushl %ebx                  movl 16(%ebp), %edi
  movl 12(%ebp), %eax         subl $8, %esp               pushl %esi
  movdqa (%eax), %xmm0        movl $0, -12(%ebp)          movl $1, %esi
  movl 16(%ebp), %eax        .L2:                         pushl %ebx
  pmaddwd %xmm0, %xmm1        cmpl $3, -12(%ebp)          .p2align 4,,15
  movdqa %xmm1, (%eax)        jle .L5                    .L2:
  popl %ebp                   jmp .L1                     movl 12(%ebp), %eax
  ret                        .L5:                         leal 0(,%esi,4), %edx
                              movl -12(%ebp), %eax        movl 8(%ebp), %ebx
                              leal 0(,%eax,4), %ebx       addl %edx, %ebx
                              movl 16(%ebp), %esi         addl %eax, %edx
                              movl -12(%ebp), %eax        movswl -4(%edx),%eax
                              leal 0(,%eax,4), %edx       movswl -4(%ebx),%ecx
                              movl 8(%ebp), %eax          movswl -2(%edx),%edx
                              movswl (%eax,%edx),%ecx     imull %eax, %ecx
                              movl -12(%ebp), %eax        movswl -2(%ebx),%eax
                              leal 0(,%eax,4), %edx       imull %edx, %eax
                              movl 12(%ebp), %eax         addl %eax, %ecx
                              movswl (%eax,%edx),%eax     movl %ecx, -4(%edi,%esi,4)
                              imull %eax, %ecx            incl %esi
                              movl -12(%ebp), %eax        cmpl $5, %esi
                              sall $2, %eax               jne .L2
                              addl 8(%ebp), %eax          popl %ebx
                              addl $2, %eax               popl %esi
                              movswl (%eax),%edx          popl %edi
                              movl -12(%ebp), %eax        popl %ebp
                              sall $2, %eax               ret
                              addl 12(%ebp), %eax
                              addl $2, %eax
                              movswl (%eax),%eax
                              imull %edx, %eax
                              leal (%eax,%ecx), %eax
                              movl %eax, (%esi,%ebx)
                              leal -12(%ebp), %eax
                              incl (%eax)
```

SIMD intrinsics	-nooptim	-O2 and vectorizer

```
 jmp .L2
.L1:
 addl $8, %esp
 popl %ebx
 popl %esi
 popl %ebp
 ret
```

6.6.5 Benchmark

```
  short int a[8] __attribute__((aligned));
  short int b[8] __attribute__((aligned));
  int c[4] __attribute__((aligned));

  int i;

  for(i = 0; i<4; i++)
    {
      a[i] = 140 + i;
      b[i] = 140 + 2*i;
    }
 for(i=0; i<30000000; i++)
    {
      test_loop_c(a, b, c);
    }

 for(i=0; i<30000000; i++)
    {
      test_loop_simd(a, b, c);
    }
```

GCC 3.5 - not optimized	15.474
GCC 4.0 - not optimized	14.757
GCC 4.1 - not optimized	15.342
ICC 8.1 - not optimized	12.2
GCC 4.0	9.636
GCC 4.1	9.683
ICC 8.1	6.356
GCC SIMD	1.563
ICC SIMD	2.36

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

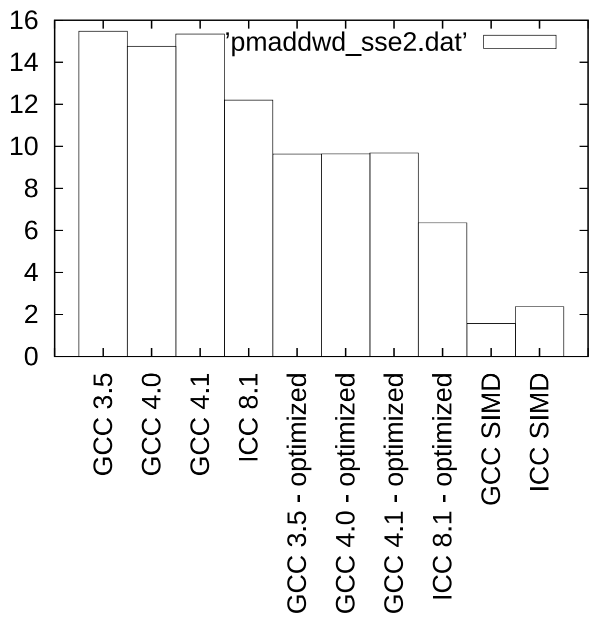

Figure 64: Benchmarks for pmaddwd - SSE2 version

6.7 psadbw - MMX (64 bits registers) version

6.7.1 C code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned short int c[4])
{
  int i;
  unsigned char tmparray[8];

  for(i=0; i<4; i++)
    {
      c[i] = 0;
    }

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

6.7.2 GIMPLE code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[4])
{
  int i=0, j=0, k=0;
  unsigned char tmparray[8];
  loop_label1::
  if(i >= 4)
    goto break_label1;
  c[i] = 0;
  i = i + 1;
  goto loop_label1;
  break_label1::
  loop_label2::
  if(j >= 8)
    goto break_label2;
  t1 = a[j];
  t2 = b[j];
  t3 = a[j] - b[j];
  if(t3 < 0)
    t4 = -t3;
  else
    t4 = t3;
  tmparray[j] = t4;
  j = j + 1;
  break_label2::
  t5 = 0;
  loop_label3::
   if(k >= 8)
    goto break_label3;
  t6 = tmparray[k];
  t5 = t5 + t6
  break_label3::
    c[0] = t5;
}
```

6.7.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  *(__m64 *) c = _mm_sad_pu8(*(__m64*) a,  *(__m64*) b);
}
```

6.7.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	xorl %eax, %eax
movl 8(%ebp), %eax	subl $12, %esp	movl %esp, %ebp
movq (%eax), %mm0	movl $1, -4(%ebp)	pushl %edi
movl 12(%ebp), %eax	.L2:	movl 12(%ebp), %edi
psadbw (%eax), %mm0	cmpl $3, -4(%ebp)	pushl %esi
movl 16(%ebp), %eax	jle .L5	movl 16(%ebp), %esi
movq %mm0, (%eax)	jmp .L3	pushl %ebx
popl %ebp	.L5:	.p2align 4,,15
ret	movl -4(%ebp), %eax	.L2:
	leal (%eax,%eax), %edx	movw $0, 2(%esi,%eax,2)
	movl 16(%ebp), %eax	incl %eax
	movw $0, (%eax,%edx)	cmpl $3, %eax
	leal -4(%ebp), %eax	jne .L2
	incl (%eax)	movl $1, %ecx
	jmp .L2	.p2align 4,,15
	.L3:	.L4:
	movl $0, -4(%ebp)	movl 8(%ebp), %edx

SIMD intrinsics	-nooptim	-O2 and vectorizer
	.L6:	movzbl -1(%edx,%ecx), %eax
	cmpl $7, -4(%ebp)	movzbl -1(%edi,%ecx), %edx
	jle .L9	incl %ecx
	jmp .L7	subl %edx, %eax
	.L9:	cltd
	movl -4(%ebp), %eax	xorl %edx, %eax
	addl 8(%ebp), %eax	subl %edx, %eax
	movzbl (%eax), %edx	addl %ebx, %eax
	movl -4(%ebp), %eax	cmpl $9, %ecx
	addl 12(%ebp), %eax	movzwl %ax, %ebx
	movzbl (%eax), %eax	jne .L4
	subl %eax, %edx	movw %bx, (%esi)
	movl %edx, %eax	popl %ebx
	movl %eax, -12(%ebp)	popl %esi
	cmpl $0, -12(%ebp)	popl %edi
	jns .L10	popl %ebp
	negl -12(%ebp)	ret
	.L10:	
	movzwl -6(%ebp), %edx	
	movl -12(%ebp), %eax	
	leal (%eax,%edx), %eax	
	movw %ax, -6(%ebp)	
	leal -4(%ebp), %eax	
	incl (%eax)	
	jmp .L6	
	.L7:	
	movl 16(%ebp), %edx	
	movzwl -6(%ebp), %eax	
	movw %ax, (%edx)	
	leave	
	ret	

6.7.5 Benchmark

```
 unsigned char a[8] __attribute__((aligned));
 unsigned char b[8] __attribute__((aligned));
 unsigned char c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 120;
     b[i] = 115+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
 test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	39.163
GCC 4.0 - not optimized	30.713
GCC 4.1 - not optimized	31.682
ICC 8.1 - not optimized	40.501

GCC 4.0	14.876
GCC 4.1	14.99
ICC 8.1	15.884

GCC SIMD	1.421
ICC SIMD	1.857

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

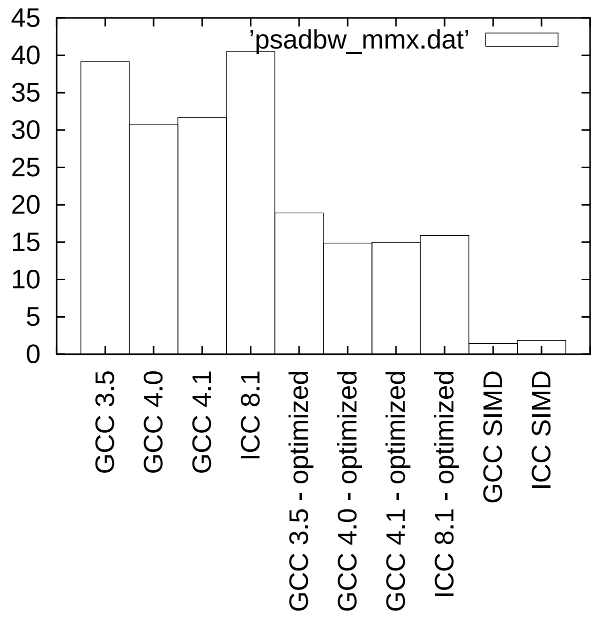

Figure 65: Benchmarks for psadwb - MMX version

6.8 psadbw - SSE2 (128 bits registers) version

6.8.1 C code

```
void test_loop_c(unsigned char a[16], unsigned char b[16], unsigned short int c[8])
{
  int i;
  unsigned char tmparray[16];

  for(i=0; i<4; i++)
    {
      c[i] = 0;
      c[i+4] = 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];
    }
}
```

6.8.2 GIMPLE code

```
void test_loop_c(unsigned char a[8], unsigned char b[8], unsigned char c[4])
{
  int i=0, j=0, k=0;
  unsigned char tmparray[8];
  loop_label1::
  if(i >= 4)
    goto break_label1;
  c[i] = 0;
  i = i + 1;
  goto loop_label1;
  break_label1::
  loop_label2::
  if(j >= 8)
    goto break_label2;
  t1 = a[j];
  t2 = b[j];
  t3 = a[j] - b[j];
  if(t3 < 0)
    t4 = -t3;
  else
    t4 = t3;
  tmparray[j] = t4;
  j = j + 1;
  break_label2::
  t5 = 0;
  t6 = 0;
  goto loop_label2;
  loop_label3::
   if(k >= 8)
    goto break_label3;
  t7 = tmparray[k];
  t8 = k+8;
  t9 = tmparray[t8];
  t5 = t5 + t7;
  t6 = t6 + t9
  goto loop_label3;
  break_label3::
  c[0] = t5;
  c[4] = t6;
}
```

6.8.3 Code with SIMD extensions

This code uses the SIMD intrinsics:

```
void test_loop_simd(unsigned char a[8], unsigned char b[8], unsigned char c[8])
{
  *(__m128i *) c = _mm_sad_epu8(*(__m128i *) a,  *(__m128i *) b);
}
```

6.8.4 Assembly code

The assembly codes that are generated are the following:

SIMD intrinsics	-nooptim	-O2 and vectorizer
pushl %ebp	pushl %ebp	pushl %ebp
movl %esp, %ebp	movl %esp, %ebp	xorl %edx, %edx
movl 8(%ebp), %eax	subl $16, %esp	movl %esp, %ebp
movdqa (%eax), %xmm0	movl $1, -4(%ebp)	pushl %edi
movl 12(%ebp), %eax	.L2:	movl 16(%ebp), %eax
psadbw (%eax), %xmm0	cmpl $3, -4(%ebp)	pushl %esi
movl 16(%ebp), %eax	jle .L5	movl 8(%ebp), %edi
movdqa %xmm0, (%eax)	jmp .L3	movl 12(%ebp), %esi
popl %ebp	.L5:	pushl %ebx
ret	movl -4(%ebp), %eax	.p2align 4,,15
	leal (%eax,%eax), %edx	.L2:
	movl 16(%ebp), %eax	incl %edx
	movw $0, (%eax,%edx)	movw $0, 2(%eax)
	movl -4(%ebp), %eax	movw $0, 10(%eax)
	addl %eax, %eax	addl $2, %eax
	addl 16(%ebp), %eax	cmpl $3, %edx
	addl $8, %eax	jne .L2
	movw $0, (%eax)	movl $1, %ecx
	leal -4(%ebp), %eax	.p2align 4,,15
	incl (%eax)	.L4:
	jmp .L2	movzbl -1(%esi,%ecx), %edx
	.L3:	movzbl -1(%edi,%ecx), %eax
	movl $0, -4(%ebp)	incl %ecx
	.L6:	subl %edx, %eax
	cmpl $7, -4(%ebp)	cltd
	jle .L9	xorl %edx, %eax
	jmp .L7	subl %edx, %eax
	.L9:	addl %ebx, %eax
	movl -4(%ebp), %eax	cmpl $9, %ecx
	addl 8(%ebp), %eax	movzwl %ax, %ebx
	movzbl (%eax), %edx	jne .L4
	movl -4(%ebp), %eax	movl 16(%ebp), %eax
	addl 12(%ebp), %eax	xorl %ecx, %ecx
	movzbl (%eax), %eax	movw %bx, (%eax)
	subl %eax, %edx	xorl %ebx, %ebx
	movl %edx, %eax	.p2align 4,,15
	movl %eax, -12(%ebp)	.L6:
	cmpl $0, -12(%ebp)	movzbl 8(%ecx,%esi), %edx
	jns .L10	movzbl 8(%ecx,%edi), %eax
	negl -12(%ebp)	incl %ecx
	.L10:	subl %edx, %eax
	movzwl -6(%ebp), %edx	cltd
	movl -12(%ebp), %eax	xorl %edx, %eax
	leal (%eax,%edx), %eax	subl %edx, %eax
	movw %ax, -6(%ebp)	addl %ebx, %eax
	leal -4(%ebp), %eax	cmpl $8, %ecx
	incl (%eax)	movzwl %ax, %ebx
	jmp .L6	jne .L6
	.L7:	movl 16(%ebp), %eax
	movl 16(%ebp), %edx	movw %bx, 8(%eax)
	movzwl -6(%ebp), %eax	popl %ebx
	movw %ax, (%edx)	popl %esi

SIMD intrinsics	-nooptim	-O2 and vectorizer
	movw $0, -6(%ebp)	popl %edi
	movl $8, -4(%ebp)	popl %ebp
	.L11:	ret
	cmpl $15, -4(%ebp)	
	jle .L14	
	jmp .L12	
	.L14:	
	movl -4(%ebp), %eax	
	addl 8(%ebp), %eax	
	movzbl (%eax), %edx	
	movl -4(%ebp), %eax	
	addl 12(%ebp), %eax	
	movzbl (%eax), %eax	
	subl %eax, %edx	
	movl %edx, %eax	
	movl %eax, -16(%ebp)	
	cmpl $0, -16(%ebp)	
	jns .L15	
	negl -16(%ebp)	
	.L15:	
	movzwl -6(%ebp), %edx	
	movl -16(%ebp), %eax	
	leal (%eax,%edx), %eax	
	movw %ax, -6(%ebp)	
	leal -4(%ebp), %eax	
	incl (%eax)	
	jmp .L11	
	.L12:	
	movl 16(%ebp), %edx	
	addl $8, %edx	
	movzwl -6(%ebp), %eax	
	movw %ax, (%edx)	
	leave	
	ret	

6.8.5 Benchmark

```
 unsigned char a[16] __attribute__((aligned));
 unsigned char b[16] __attribute__((aligned));
 unsigned short int c[8] __attribute__((aligned));
 int i;

 for(i = 0; i<8; i++)
   {
     a[i] = 120;
     b[i] = 115+2*i;
   }
for(i=0; i<30000000; i++)
   {
     test_loop_c(a, b, c);
   }

for(i=0; i<30000000; i++)
   {
 test_loop_simd(a, b, c);
   }
```

GCC 3.5 - not optimized	93.453
GCC 4.0 - not optimized	70.008
GCC 4.1 - not optimized	71.41
ICC 8.1 - not optimized	74.793

GCC 4.0	35.57
GCC 4.1	34.927
ICC 8.1	28.671

GCC SIMD	2.015
ICC SIMD	3.084

GCC 4.0 behavior	-O2 optim, no vectorization
GCC 4.1 behavior	-O2 optim, no vectorization
ICC behavior	Unrolling

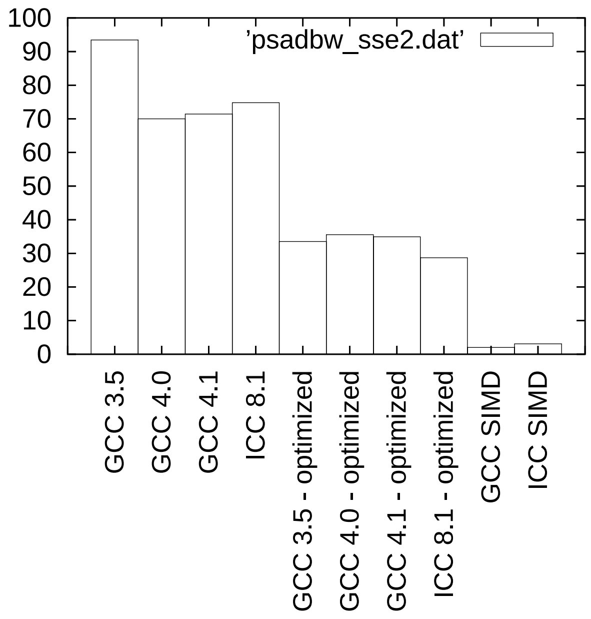

Figure 66: Benchmarks for psadwb - SSE2 version

References

[1] Intel. Ia-32 intel architecture software developer's manual. Technical report, Intel, 2004.

[2] Jason Merril. Generic and gimple, a new tree representation for entire functions. In *Proceedings of the 2003 GCC Developers Summit*, pages 171–193, May 2003.