Автоматическая векторизация в GCC для архитектуры PowerPC

исходный код программы на языке C:

```
#define N 16

void fbar (float *);
void ibar (int *);
void sbar (short *);

/* multiple loops */

foo (int n)
{
  float a[N+1];
  float b[N];
  float c[N];
  float d[N];
  int i;

  /* Strided access. Vectorizable on platforms that support load of strided
     accesses (extract of even/odd vector elements).  */
  for (i = 0; i < N/2; i++){
    a[i] = b[2*i+1] * c[2*i+1] - b[2*i] * c[2*i];
    d[i] = b[2*i] * c[2*i+1] + b[2*i+1] * c[2*i];
  }
  fbar (a);

}
```

Строка компиляции: **ppc-linux-gnu-gcc -O3 -maltivec -ftree-vectorizer-verbose=1 -ftree-vectorize -S vect-1.c**

> Каждая команда PowerPC имеет длину 32 бита. Первые 6 бит определяют команду, а остальные имеют различное значение, зависящее от команды. Тот факт, что команды имеют фиксированную длину, позволяет процессору выполнять их более эффективно. Поскольку команды PowerPC имеют длину только 32 бита, внутри команд, загружающих постоянные величины, в наличии имеется только 16 бит. Поэтому, так как адрес может быть до 64 бит в длину, мы должны загружать его небольшими порциями. Значок @ в ассемблере указывает ассемблеру использовать специальную форму.

Результат:

```
        .file   "vect-1.c"
```

```
        .section        ".text"
        .align 2
        .globl foo
        .type   foo, @function
foo:
    # Переместить значение из Link Register(похоже что это адрес текущей инструкции) в регистр 0
        mflr 0
        # Store Word with Update (сохранить значение регистра 1(биты 32...63), в адрес памяти(EA) = <значение регистра 1>+<число -224>
        # EA <- (1) - 224
        # MEM(EA, 4) <- (1)32:63
        # (1) <- EA
        stwu 1,-224(1)
        # Load Immediate Shifted(непосредственная загрузка)
    # Она загружает величину (биты 16-31 адреса LC1)
    # сдвигает число на 16 бит налево и затем сохраняет результат в регистре 11
    # Биты 16-31 регистра 11 содержат биты 16-31 адреса.
        lis 11,.LC1@ha
        lis 9,.LC0@ha
        # Load Address
        # la RT,SI(RS) (equivalent to: addi RT,RA,SI)
        # if RA = 0 then RT <- EXTS(SI)
        # else           RT <- (RA) + EXTS(SI)
        # The sum (RA|0) + SI is placed into register RT.
    # Поместить в 11 регистр сумму 11 регистра и битов 0:15 LC1
    # Хы: в 11 регистре окажутся 0:31 биты из ячейки по адресу LC1
        la 11,.LC1@l(11)
    # в 10 регистре сумма 1 и 16
        addi 10,1,16
        # Store Word
        # stw RS,D(RA)
        # if RA = 0 then b <- 0
        # else           b <- (RA)
        # EA <- b + EXTS(D)
        # MEM(EA, 4) <- (RS)32:63
        # Let the effective address (EA) be the sum (RA|0)+ D. (RS)32:63 are stored into the word in storage addressed by EA.
    # биты 32:63 регистра 0 будут помещены по адресу значение регистра 1 + 228 (в биты 0-31?)
        stw 0,228(1)
        addi 8,10,16
        lvx 11,0,11
        addi 11,1,16
        lvx 13,0,11
        addi 11,1,80
        lvx 10,0,11
        addi 11,1,96
        lvx 1,0,11
        addi 11,1,112
```

```
        lvx 8,0,11
        la 9,.LC0@l(9)
        addi 11,1,128
        lvx 7,0,9
        lvx 6,0,8
        addi 9,8,32
        lvx 0,0,11
        addi 8,8,16
        lvx 4,0,9
        addi 9,1,144
        lvx 12,0,8
        mr 3,9
        vperm 3,8,0,11
        vperm 5,10,1,11
        vperm 9,13,6,11
        vperm 8,8,0,7
        vperm 10,10,1,7
        vperm 13,13,6,7
        vspltisw 0,-1
        vslw 0,0,0
        vmaddfp 13,13,10,0
        vmaddfp 9,9,5,0
        vsubfp 13,13,9
        stvx 13,0,9
        addi 9,9,16
        vperm 11,12,4,11
        vperm 12,12,4,7
        vmaddfp 11,11,3,0
        vmaddfp 12,12,8,0
        vsubfp 12,12,11
        stvx 12,0,9
        vor 1,0,0
        bl fbar
        lwz 0,228(1)
        addi 1,1,224
        mtlr 0
        blr
        .size   foo, .-foo
        .section        .rodata.cst16,"aM",@progbits,16
        .align 4
.LC0:
        .byte   4
        .byte   5
        .byte   6
        .byte   7
        .byte   12
        .byte   13
        .byte   14
        .byte   15
        .byte   20
        .byte   21
```

```
        .byte   22
        .byte   23
        .byte   28
        .byte   29
        .byte   30
        .byte   31
.LC1:
        .byte   0
        .byte   1
        .byte   2
        .byte   3
        .byte   8
        .byte   9
        .byte   10
        .byte   11
        .byte   16
        .byte   17
        .byte   18
        .byte   19
        .byte   24
        .byte   25
        .byte   26
        .byte   27
        .ident  "GCC: (GNU) 4.3.0 20080202 (experimental)"
        .section        .note.GNU-stack,"",@progbits
```