Optimizing C code

March 7, 1996

Topics

- principles
- sparse matrix example

Concepts

Focus on the loops

- inner loops first
- then outer loops

Use temps to promote array elements to registers

- C compilers have trouble with array elements because of the potential for aliasing.
- e.g. a[i] and b[i] may be the same element.

Use pointer arithmetic

- does not mean you simply convert x[r] to *(x + r)

Aliasing

Question: Can the compiler safely accumulate the lhs in a register,instead of loading and storing each time?

```
myproc(int *a, int *b, int r) {
int i;

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

```
myproc(int *a, int *b, int r) {
int i, temp;

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


Original C code

```
typedef struct {
  matrix_t type;          /* ( 0)  */
  int nrow;               /* ( 4) */
  int nentries;           /* ( 8)  */
  ftype_t *val;           /* (12)  */
  int *cindex;            /* (16)  */
  int *rstart;            /* (20)  */
} csr_rec, *csr_ptr;
```

```
int r, ci;

for (r = 0; r < M->nrow; r++) {
  z[r] = 0.0;
  for (ci = M->rstart[r]; ci < M->rstart[r+1]; ci++) {
    z[r] += M->val[ci] * x[M->cindex[ci]];
}
```

```
--begin inner loop
0x4c:   lw      v1,16(t0)
0x50:   sll     v0,a3,2
0x54:   addu    v0,v0,v1
0x58:   lw      v1,12(t0)
0x5c:   lw      v0,0(v0)
0x60:   addu    a0,a0,v1
0x64:   sll     v0,v0,3
0x68:   addu    v0,v0,a1
0x6c:   lwc1    $f2,0(a0)
0x70:   lwc1    $f3,4(a0)
0x74:   lwc1    $f0,0(v0)
0x78:   lwc1    $f1,4(v0)
0x7c:   nop
0x80:   mul.d   $f2,$f2,$f0
0x84:   lwc1    $f0,0(t1)
0x88:   lwc1    $f1,4(t1)
0x8c:   nop
0x90:   add.d   $f0,$f0,$f2
0x94:   swc1    $f0,0(t1)
0x98:   swc1    $f1,4(t1)
0x9c:   lw      v0,20(t0)
0xa0:   nop
0xa4:   addu    v0,t2,v0
0xa8:   lw      v0,4(v0)
0xac:   addiu   a3,a3,1
0xb0:   slt     v0,a3,v0
0xb4:   bne     v0,zero,0x4c
0xb8:   sll     a0,a3,3
--end inner loop
```

```
--begin inner loop
 0x4c:    lw      v1,16(t0)
 0x50:    sll     v0,a3,2
 0x54:    addu    v0,v0,v1
 0x58:    lw      v1,12(t0)
 0x5c:    lw      v0,0(v0)
 0x60:    addu    a0,a0,v1
 0x64:    sll     v0,v0,3
 0x68:    addu    v0,v0,a1
 0x6c:    lwc1    $f2,0(a0)
 0x70:    lwc1    $f3,4(a0)
 0x74:    lwc1    $f0,0(v0)
 0x78:    lwc1    $f1,4(v0)
 0x7c:    nop
 0x80:    mul.d   $f2,$f2,$f0
 0x84:    lwc1    $f0,0(t1)
 0x88:    lwc1    $f1,4(t1)
 0x8c:    nop
 0x90:    add.d   $f0,$f0,$f2
 0x94:    swc1    $f0,0(t1)
 0x98:    swc1    $f1,4(t1)
 0x9c:    lw      v0,20(t0)
 0xa0:    nop
 0xa4:    addu    v0,t2,v0
 0xa8:    lw      v0,4(v0)
 0xac:    addiu   a3,a3,1
 0xb0:    slt     v0,a3,v0
 0xb4:    bne     v0,zero,0x4c
 0xb8:    sll     a0,a3,3
--end inner loop
```

```
int r, ci;

for (r = 0; r < M->nrow; r++) {
  ftype_t temp = 0.0;
  for (ci = M->rstart[r]; ci < M->rstart[r+1]; ci++)
    temp += M->val[ci] * x[M->cindex[ci]];
  z[r] = temp;
}
```

```
--begin inner loop (promoting z[r] to a register)
 0x5c:    lw      v0,0(v1)     # v0 = cindex[ci]
 0x60:    lwc1    $f2,0(t0)    # load val[ci]
 0x64:    lwc1    $f3,4(t0)
 0x68:    sll     v0,v0,3      # v0 = byte offset
 0x6c:    addu    v0,v0,a1     # v0 = &x[cindex[ci]
 0x70:    lwc1    $f0,0(v0)    # load x
 0x74:    lwc1    $f1,4(v0)
 0x78:    nop
 0x7c:    mul.d   $f2,$f2,$f0   # mult *val and *x
 0x80:    addiu   t0,t0,8       # t0 = &val[ci+1]
 0x84:    addiu   v1,v1,4       # v1 = &cindex[ci+1]
 0x88:    addiu   a3,a3,1       # a3 = ci++ (*)
 0x8c:    slt     v0,a3,t2      # check loop bound
 0x90:    bne     v0,zero,0x5c  # and repeat
 0x94:    add.d   $f4,$f4,$f2   # after summing
--end inner loop
```



```
int r, ci;

for (r = 0; r < M->nrow; r++) {
  ftype_t temp = 0.0;
  for (ci = M->rstart[r]; ci < M->rstart[r+1]; ci++)
    temp += M->val[ci] * x[M->cindex[ci]];
  z[r] = temp;
}
```

```
--begin inner loop
 0x5c:    lw      v0,0(v1)
 0x60:    lwc1    $f2,0(t0)
 0x64:    lwc1    $f3,4(t0)
 0x68:    sll     v0,v0,3
 0x6c:    addu    v0,v0,a1
 0x70:    lwc1    $f0,0(v0)
 0x74:    lwc1    $f1,4(v0)
 0x78:    nop
 0x7c:    mul.d   $f2,$f2,$f0
 0x80:    addiu   t0,t0,8
 0x84:    addiu   v1,v1,4
 0x88:    addiu   a3,a3,1
 0x8c:    slt     v0,a3,t2
 0x90:    bne     v0,zero,0x5c
 0x94:    add.d   $f4,$f4,$f2
--end inner loop
```

```
int r;
ftype_t *val = M->val;
int *cindex_start = M->cindex;
int *cindex = M->cindex;
int *rnstart = M->rstart+1;
for (r = 0; r < M->nrow; r++) {
  ftype_t temp = 0.0;
  int *cindex_end = cindex_start + *(rnstart++);
  while (cindex < cindex_end)
    temp += *(val++) * x[*cindex++];
  z[r] = temp;
}
```

```
-- begin inner loop (pointer arith on the inner loop)
 0x48:    lw      v0,0(v1)     # v0 = *cindex
 0x4c:    lwc1    $f2,0(t0)     # load *val
 0x50:    lwc1    $f3,4(t0)
 0x54:    sll     v0,v0,3      # v0 = byte offset
 0x58:    addu    v0,v0,a1     # v0 = &x[*cindex]
 0x5c:    lwc1    $f0,0(v0)    # load x[*cindex]
 0x60:    lwc1    $f1,4(v0)
 0x64:    nop
 0x68:    mul.d   $f2,$f2,$f0  # val * x
 0x6c:    addiu   v1,v1,4      # v1 = cindex++
 0x70:    addiu   t0,t0,8       # t0 = val++
 0x74:    sltu    v0,v1,a3     # check bound
 0x78:    bne     v0,zero,0x48# and repeat
 0x7c:    add.d   $f4,$f4,$f2  # after summing
-- end inner loop
```

```
 int r; ftype_t *val = M->val;
int *cindex_start = M->cindex;
int *cindex = M->cindex;
int *rnstart = M->rstart+1;
for (r = 0; r < M->nrow; r++) {
  ftype_t temp = 0.0;
  int *cindex_end = cindex_start + *(rnstart++);
  while (cindex < cindex_end)
    temp += *(val++) * x[*cindex++];
  z[r] = temp;
}
```

```
-- begin outer loop
 0x28:    lw        v0,0(t2)     # v0 = *rnstart
 0x2c:    mtc1      zero,$f4     # temp = 0.0
 0x30:    mtc1      zero,$f5
 0x34:    sll       v0,v0,2      # v0 = bye offset
 0x38:    addu      a3,t3,v0     # cindex_end
 0x3c:    sltu      v0,v1,a3     # check bound
 0x40:    beq       v0,zero,0x80
 0x44:    addiu     t2,t2,4       # t2 = rnstart++
INNER LOOP
 0x80:    swc1      $f4,0(a2)     # store *z
 0x84:    swc1      $f5,4(a2)
 0x88:    lw        v0,4(a0)     # get nrow
 0x8c:    addiu     t1,t1,1      # increment r
 0x90:    slt       v0,t1,v0     # check bound
 0x94:    bne       v0,zero,0x28# and repeat
 0x98:    addiu     a2,a2,8      # bump  z
--end outer loop
```

```
 int r;  ftype_t *val = M->val;
 int *cindex_start = M->cindex;
 int *cindex = M->cindex;
 int *rnstart = M->rstart+1;
 ftype_t *z_end = z+M->nrow;
 while (z < z_end) {
   ftype_t temp = 0.0;
   int *cindex_end = cindex_start + *(rnstart++);
   while (cindex < cindex_end)
     temp += *(val++) * x[*cindex++];
   *z++ = temp;
 }
```

```
-- begin outer loop (pointer arith on outer loop)
 0x2c:    lw        v0,0(t0)     # v0 = *rnstart
 0x30:    mtc1      zero,$f4     # clear temp
 0x34:    mtc1      zero,$f5
 0x38:    sll       v0,v0,2      # v0 = offset
 0x3c:    addu      a0,t2,v0     # a0=cindex_end
 0x40:    sltu      v0,v1,a0     #
 0x44:    beq       v0,zero,0x84# and quit
 0x48:    addiu     t0,t0,4       # after updating
INNER LOOP
 0x84:    swc1      $f4,0(a2)    # store *z
 0x88:    swc1      $f5,4(a2)
 0x8c:    addiu     a2,a2,8      # a2 = z++
 0x90:    sltu      v0,a2,t1     # branch back
 0x94:    bne       v0,zero,0x2c
 0x98:    nop
-- end outer loop
```