

Performance Optimization

Note Title

11/5/2007

- there is much that compiler cannot do
- = there is much that the programmer should not do, most of the time

invoke optimization supported by compiler

1) drop -g switch

2) add -O2
=>

things the compiler can't do

```
void twiddle1(int *xp, int *yp) {  
    *xp += *yp;  
    *xp += *yp;  
}
```

alternate code?

```
void twiddle2(int *xp, int *yp) {  
    *xp += 2 * *yp;  
}
```

what if $xp == yp$

pointers can point anywhere

e.g. 2:

```
int f(int);
```

```
int func1 (int x) {  
    return ( f(x) + f(x) + f(x) + f(x) );  
}
```

```
int func2 (int x) {  
    return ( 4 * f(x) );  
}
```

```
int counter = 0;  
int f(int x) {  
    return (counter++);  
}
```

side effects

problem size n

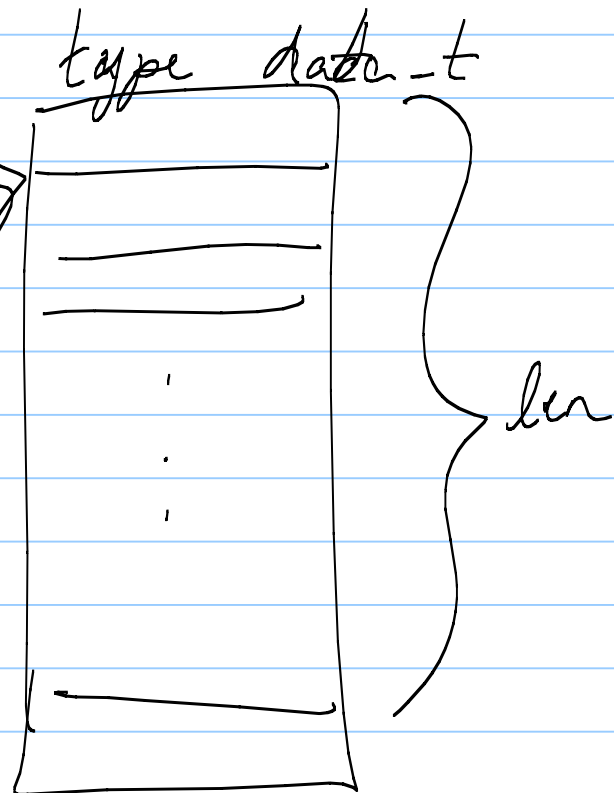
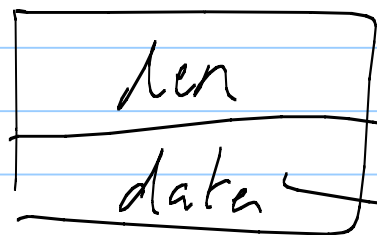
modul run time

total cycles = CPE * n + starting overhead

perform assoc. operation on array of elements

x_1 OP x_2 OP x_3 OP ... OP x_n

vec_rec



```
/* abstract data type for vector */
typedef struct {
    int len;
    data_t *data;
} vec_rec, *vec_ptr;

/* create vector */
vec_ptr new_vec(int len) {
    /* allocate header structure */
    vec_ptr result = (vec_ptr)malloc(sizeof(vec_rec));
    if (!result) return NULL; /* no memory available */
    result->len = len;
    /* allocate array */
    if (len > 0) {
        data_t *data = (data_t *)calloc(len, sizeof(data_t));
        if (!data) {
            free((void *)result);
            return NULL;
        }
        result->data = data;
    }
    else {
        result->data = NULL;
    }
    return result;
}
```

```
/* retrieve vector element -> dest */
/* return 0 (out of bounds) or 1 (successful) */
int get_vec_element(vec_ptr v, int index, data_t *dest) {

    if (index < 0 || index >= v->len) {
        return 0;
    }
    *dest = v->data[index];
    return 1;
}

/* retrieve vector length */
int vec_length(vec_ptr v) {
    return v->len;
}

/* set vector element */
/* return 0 (out of bounds) or 1 (successful) */
int set_vec_element(vec_ptr v, int index, data_t val) {

    if (index < 0 || index >= v->len) {
        return 0;
    }
    v->data[index] = val;
    return 1;
}
```

operation parameterized by #define

```
#define OP + or *  
#define IDENT 0 1
```

data type parameterized by typedef

```
typedef int data_t;  
float  
double
```

```

void combine1(vec_ptr v, data_t *dest) {
    int i;
    data_t val;

    *dest = IDENT;
    for (i=0; i<vec_length(v); i++) {
        get_vec_element(v, i, &val);
        *dest = *dest OP val;
    }
    return;
}

```

unoptimized	compiler	CPE	42.06	int sum
-O2		CPE	31.25	

- Code Migration

more call $\#cc_length()$ out of loop

```
void combine2(vec_ptr v, data_t *data) {
    int i;
    int length;
    data_t val;

    length = vec_length(v);
    *dest = IDENT;
    for (i=0; i<length; i++) {
        get_vec_element(v, i, &val);
        *dest = *dest OP val;
    }
    return;
}
```

CPE

22.61

vs.

31.25

before code movement

- Diminish Generality

drop proc call w/ bounds check
and use knowledge that array
is contiguous

```
data_t *get_vec_start(vec_ptr v) {  
    return v->data;  
}  
  
void combine3(vec_ptr v, data_t *dest) {  
    int i;  
    int length;  
    data_t *data;  
  
    length = vec_length(v);  
    data = get_vec_start(v);  
    *dest = IDENT;  
    for (i=0; i<length; i++) {  
        *dest = *dest OP data[i];  
    }  
    return;  
}
```

resulting CPE 6.0

- Dominant Memory Refs

* test

=> use local variable

```

void combine4(vec_ptr v, data_t *dest) {
    int i;
    int length;
    data_t *data;
    data_t x;

    length = vec_length(v);
    data = get_vec_start(v);
    x = IDENT;
    for (i=0; i<length; i++) {
        x = x OP data[i];
    }
    *dest = x;
    return;
}

```

resulting CPE 2.0
 w. original CPE 42.06
 speedup of 21X

exploit knowledge of micro architecture

unroll loop

```

void combine5(vec_ptr v, data_t *dest) {
    int i;
    int length;
    data_t *data;
    data_t x;
    int limit;

    length = vec_length(v);
    data = get_vec_start(v);
    x = IDENT;
    limit = length-2;
    /* combine 3 elements at a time */
    for (i=0; i<limit; i+=3) {
        x = x OP data[i] OP data[i+1] OP data[i+2];
    }

    /* finish any remaining elements */
    for (; i<length; i++) {
        x = x OP data[i];
    }
    *dest = x;
    return;
}

```

CPE = 1.33
 startup overheads have
 increased