

Performance Optimization

CSE 361S

Compiler Optimization

- There is much the compiler can do
- Invoke via:
 - Drop the -g switch
 - Add the -O2 switch
- There is much the compiler cannot do
- There is much that the programmer should not do, most of the time

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) );
}
```

```
int func2(int x) {
    return( 3*f(x) );
}
```

Side Effects

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

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

int func2(int x) {
    return( 3*f(x) );
}
```

func1(1) =

func2(1) =

Simple Performance Measure

- Problem size: n
 - E.g., matrix-matrix multiply problem with $n \times n$ matrices
- Model execution time in terms of
 - n
 - Processor cycles
 - Startup overhead (assumed to be small)

total cycles = $(CPE)(n) + \text{startup overhead}$

- Stick with problems that are linear in n

Running Example Problem

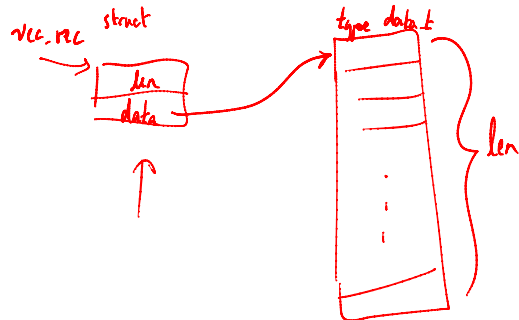
- Perform assoc. operation on array of elements

$$\text{Type OP } x_1 \text{ OP } x_2 \text{ OP } x_3 \text{ OP } \dots \text{ OP } x_n$$
- Operation parameterized by #define


```
#define OP    +    or    *
#define IDENT 0    or    1
```
- Data type parameterized by typedef


```
typedef int  data_t;
or
typedef float data_t;
```

Memory Image of Array



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

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

```
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;
}
```

data type: int operation: +

unoptimized compile CPE 42.06 cycles/element
 -O2 compile CPE 31.25 cycles/element

Code Migration

- Move call to `vec_length()` out of loop

```
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;
}
```

```
void combine2(vec_ptr v, data_t *dest) {
    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;
}
```

-O2 compile	CPE	31.25 cycles/element
w/ code migration	CPE	22.61

Diminish Generality

- Drop procedure call with bounds check
- Use knowledge that array is contiguous
 - Note that interface of `get_vec_element()` makes no such assumption

```
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;
}
```

CPE goes from 22.61 to 6.0

Diminish Memory Reference Count

- Replace references to `*dest` with local variable

```
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;
}
```

```
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
vs. original CPE 42.06 speedup of 21X

Exploit Microarchitecture Knowledge

- Superscalar instruction execution
 - Can amortize more compute relative to looping overheads
 - Unroll loop

```
void combine5(vec_ptr v, data_t *dest) {
    int i;
    int length = vec_length(v);
    data_t *data = get_vec_start(v);
    data_t x = IDENT;
    int limit = length-2;

    for (i=0; i < limit; i += 3) { /* combine 3 elements at a time */
        x = x OP data[i] OP data[i+1] OP data[i+2];
    }
    for(; i < length; i++) {
        x = x OP data[i];
    }
    *dest = x;
    return;
}
```

resulting CPE 1.33 startup overheads larger