Optimization: More Optimization

CPSC 501: Advanced Programming Techniques
Fall 2020

Jonathan Hudson, Ph.D
Instructor
Department of Computer Science
University of Calgary

Wednesday, August 5, 2020

UNIVERSITY OF CALGARY

Strength Reduction

Code Tuning – Strength Reduction

- **Strength Reduction**
  - Is where you replace an expensive operation with a cheaper operation
    - E.g. Replace multiplication with addition
      - Remember: multiplication is repeated addition
    - E.g.

```
for (i = 0; i < saleCount; i++) {
    commission[i] = (i + 1) * revenue
            * baseCommission * discount;
}
```


UNIVERSITY OF CALGARY

Code Tuning – Strength Reduction (cont'd)

- After strength reduction:

```
increment = revenue * baseCommission * discount;
cumulativeCommission = increment;

for (i = 0; i < saleCount; i++) {
    commission[i] = cumulativeCommission;
    cumulativeCommission += increment;
}
```

Routines

Code Tuning – Routines - Inline

- Routines
  - Rewrite routines inline
  - **(looks like function but is code replacement)**
    - C++ has the inline keyword
    - With other languages, use macros
      - E.g. in C

```
#define SQUARE(x)  ((x) * (x))
. . .
int a = 5, b;
b = SQUARE(a);
```


UNIVERSITY OF CALGARY

Code Tuning – Routines – Re-Code

- Recode in a low-level language
  - E.g. If in Java, use a native method written in C
  - E.g. If in C or C++, use assembly
  - **Portability is lost**
  - Best applied to small routines or sections of code
  - E.g. SPARC assembly

```
        .global cube
cube:   smul  %o0, %o0, %o1
        smul  %o0, %o1, %o0
        retl
        nop
```

Code Tuning – Routine - Rewrite

- Rewrite expensive system routines
  - E.g. **double log2(double x)** may give more precision than you need
    - Rounding integer version:

```
unsigned int log2(unsigned int x) {
    if (x < 2)            return 0;
    if (x < 4)            return 1;
    if (x < 8)            return 2;
     . . .
    if (x < 2147483648) return 30;
    return 31;
}
```

Data Format

Code Tuning – Data Transformation – Float/Int

- Data Transformation techniques
  - Replace f.p. numbers with integers
  - *(in OO maybe also be able to replace object type with primitive type)*
  - E.g. Visual Basic

```
Dim x As Single
For x = 0 to 99
    a(x) = 0
Next
```

  - Is faster as:

```
Dim x As Integer
. . .
```

Code Tuning – Data Transformation – Array Dims

- Reduce array dimensions where possible
  - E.g. C or C++ array

```
for (row = 0; row < numRows; row++) {
    for (column = 0; column < numColumns; column++) {
        matrix[row][column] = 0;
    }
}
```

  - Is faster as a 1D array:

```
for (entry = 0; entry < numRows * numColumns; entry++) {
    matrix[entry] = 0;
}
```

Code Tuning – Data Transformation – Array Refs

- Minimize array references
  - E.g.

```
for (i = 0; i < size; i++) {
    for (j = 0; j < n; j++) {
        rate[j] *= discount[i];
    }
}
```

  - Is better as:

```
for (i = 0; i < size; i++) {
    temp = discount[i];
    for (j = 0; j < n; j++) {
        rate[j] *= temp;
    }
}
```

Code Tuning – Data Transformation – Supp

- Use supplementary indices

  - Length index for arrays
    - E.g. Add a string-length field to C strings

  - Faster than using strlen(), which loops until null found Parallel index structure
    - E.g. Often easier to sort an array of references to a data array, than the data array itself
      - Avoids swapping data that's expensive to move (i.e. is large or on disk)

Code Tuning – Data Transformation – Caching

- Use caching
  - Save commonly used values, instead of recomputing or rereading them
  - Java example:

```java
private double cachedH = 0, cachedA = 0, cachedB = 0;

public double Hypotenuse(double A, double B) {
    if ((A == cachedA) && (B == cachedB)) {
        return cachedH;
    }
    cachedH = Math.sqrt((A * A) + (B * B));
    cachedA = A;
    cachedB = B;
    return cachedH;
}
```

Expressions

Code Tuning - Expressions

- Expressions
  - Exploit algebraic identities
    - i.e. replace expensive expressions with cheaper ones

    - E.g. not a and not b
    - Better as: not (a or b)

    - E.g. if (sqrt(x) < sqrt(y))
    - Better as: if (x < y)

Code Tuning – Expressions – Strength Reduction

- Use strength reduction
  - Replace expensive operations with cheaper ones
  - Some possibilities:

Original	Replacement
Multiplication	Addition
Exponentiation	Multiplication
Trig routines	Tri. Identities
Long ints	Ints
f.p. numbers	Fixed point numbers/ints
Doubles	Floats
Mult/div by power 2	Left/right shift

Code Tuning – Expressions – Compile Time

- Initialize at compile time
  - i.e. use constants where possible
  - E.g.

```
unsigned int Log2(unsigned int x) {
    return (unsigned int)(log(x) / log(2));
}
```

  - Is better as:

```
const double LOG2 = 0.69314718;
unsigned int Log2(unsigned int x) {
    return (unsigned int)(log(x) / LOG2);
}
```

Code Tuning – Expressions – Data Type

- Use the proper data type for constants
  - i.e. avoid runtime type conversions
  - E.g.

```
double x;
. . .
x = 5;
```

  - Is better as:

```
x = 5.0;
```

Code Tuning – Expressions – Common Sub-Exp

- Eliminate common subexpressions
  - Assign to a variable, and use it instead of recomputing
  - E.g.

```
p = (1.0 - (r / 12.0)) / (r / 12.0);
```

  - Is better as:

```
y = r / 12.0;
p = (1.0 - y) / y;
```

Code Tuning – Expressions – Precompute

- Precompute results
  - Often better to look up values than to recompute them
  - Values could be stored in constants, arrays, or files

I/O

Code Tuning – I/O

- I/O techniques
  - Minimize disk and network accesses
    - Use buffered I/O, instead of single reads/writes
  - Use RAM instead of disk whenever possible
    - Cache commonly used data
  - Localize memory accesses
    - Reading/writing registers is faster than cache memory, which is faster than DRAM
    - C and C++ provide the register keyword
      - Is a hint to the compiler to use a register instead of RAM
      - E.g. `register int x;`

Onward to … assembly optimization.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~hudsonj/

UNIVERSITY OF CALGARY