Software Optimizations

Ned Nedialkov

McMaster University
Canada

SE 3F03
March 2013

Outline

Data locality

Assume

- n particles with coordinates (x, y, z)
- We can store them (in C) as
  double `a[3][n];`
  or
  double `a[n][3];`
- In C arrays are stored row-wise
- Which is one should we choose?

- **double** `a[3][n];`
- Coordinates are stored as

$$\begin{array}{llll} x_1 & x_2 & x_3 & \cdots \\ y_1 & y_2 & y_3 & \cdots \\ z_1 & z_2 & z_3 & \cdots \end{array}$$

- The layout in memory is

$$x_1 \quad x_2 \quad x_3 \quad \cdots \quad y_1 \quad y_2 \quad y_3 \quad \cdots \quad z_1 \quad z_2 \quad z_3 \quad \cdots$$

- Accessing (x_i, y_i, z_i) is likely to result in cache misses

- **double** `a[n][3];`
- Coordinates are stored as

$$\begin{matrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \vdots & \vdots & \vdots \end{matrix}$$

- The layout in memory is

$$x_1 \quad y_1 \quad z_1 \quad x_2 \quad y_2 \quad z_2 \quad x_3 \quad y_3 \quad z_3 \cdots$$

When accessing (x_i, y_i, z_i), in most cases it will be in cache

Common subexpressions

- Assume
  double `a,b,c,s1,s2;`
- Consider

```
s1 = a+b+c;
s2 = a+b-c;
```

- A compiler may re-arrange to

```
t = a+b; //temporary
s1 = t+c;
s2 = t-c;
```

- Try to eliminate common subexpressions

- Consider

```
s1 = a+c+b;
s2 = a+b-c;
```

- In FP arithmetic `(a+c)+b` is not always `(a+b)+c`
- A compiler would evaluate from left to right
  `a+b+c` is evaluated as `(a+b)+c`
- Put brackets if needed, to help the compiler

- Consider

```
r = x[i]*x[i];
s = x[i]-1;
```

- As written, 3 memory accesses to `x[i]`
- An optimizing compiler would do

```
t = x[i];
r = t*t;
s = t-1;
```

Strength reduction

- Replacement of an arithmetic expression by another one that is faster
- Assume **int** `i`. Replace `2*i` by `i+i` or `i<<1`
- Replace `y = pow(x,2)` by

```
y = x*x;
```

- Replace `y = pow(x,4)` by

```
y = x*x;
y *= y;
```

Loop invariant code motion

- Consider

```
for (i=0; i<n; i++)
  a[i] = r*s*a[i];
```

- Move `r*s` outside the loop

```
t = r*s
for (i=0; i<n; i++)
  a[i] = t*a[i];
```

Loop unrolling

- Consider computing a dot product

```
double dotproduct(int n, double *a, double *b)
{
  int i;
  double c = 0;
  for (i=0;i<n;i++)
    c += a[i]*b[i];
}
```

- We can unroll the **for** loop 4 times as

```
double dotproduct(int n, double *a, double *b)
{
  int i, rem = n&0x3;
  double c = 0;
  for (i=0;i<n-rem;i+=4)
    {
      c += a[i]*b[i];
      c += a[i+1]*b[i+1];
      c += a[i+2]*b[i+2];
      c += a[i+3]*b[i+3];
    }
  for (i=n-rem;i<n;i++)
    c += a[i]*b[i];
}
```

- What are the advantages and disadvantages of loop unrolling?