LECTURE 8: PERFORMANCE OPTIMIZATION

Computer Systems and Networks

Dr. Pallipuram
(vpallipuramkrishnamani@pacific.edu)

Today’s Agenda

Performance Optimization: Compiler Techniques

Performance Optimization: Programmer Techniques

- Code motion
- Reduce procedure calls
- Eliminate unneeded memory accesses
- Loop Unrolling

Compiler Goals

What are the compiler's goals with optimization switch on?

Reduce program **code size**

Reduce program **execution time**

Compiler Optimization Levels

O1: Moderately optimize the code, but do not increase the compilation time

```
gcc –O1 –o myexec main.c
```

O2: Optimize more, take time, but do not increase the code size

```
gcc –O2 –o myexec main.c
```

O3: Optimize aggressively, take time, even if code size increases!

```
gcc –O3 –o myexec main.c
```

Problem 1: O3 increases code size due to inlining

Inline Functions

Write down pros and cons.

```
int max(int a, int b)
{
        if(a>b)
                return a;
        else
                return b;
}
```

```
max1 = max(w,x);
max2 = max(y,z);
printf("%i %i\n",
        max1, max2);
```

```
if(w>x) max1 = w;
else max1 = x;

if(y>z)max2 = y;
else max2 = z;

printf("%i %i\n",
        max1, max2);
```

Function Call Overhead

What specific overhead exists here?

```
int max(int a, int b)
{
        if(a>b)
                return a;
        else
                return
b;
}
```

Calling a function

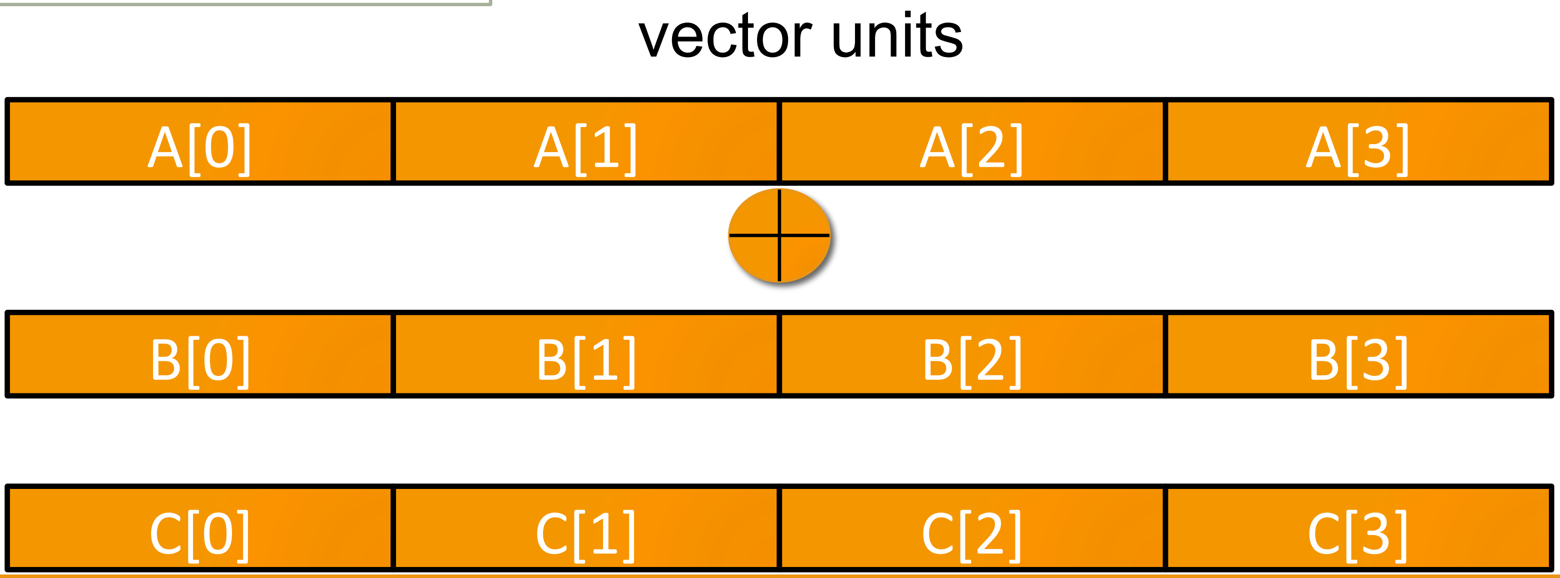
- Save variables in the processor ("registers") to memory (in the stack)
- Jump to the function
- Create new stack space for function and its local variables

Returning from function

- Load old values from stack
- Jump to prior location

O3 performs Loop vectorization

```
for(i=0;i<16;i++) {
C[i]=A[i]+B[i];
}
```



Vote

Who will do a better job improving program performance?

The compiler -vs- *The programmer*

Problem 2: Programmer Optimization: Code Motion

Move a code section from a loop to outside because that section does not need to be called over and over again!

```
for (int x=0; x<strlen(userinput); x++)
{

        if(tolower(game.grid[i][j+x])==tolower(userinput[x]))
        {
                flag=1;
        }
        else {
                flag=0;
                break;
        }
}
```

Problem 3: Program Optimization: Reduce Procedure Calls

Reduce function calls as much as you can. Can you find out why this code is inefficient and fix it?

```
struct list {
struct list *next;
int num;
};
```

```
for(i=0;i<listsize;i++)
{

ele = get_num(head,i);
printf("%d",ele);
}
```

```
int get_num(struct list
*head, int position) {

struct list *temp=head;
for(int i=0;i<position;i++)
{
temp=temp->next;
}
return temp->num;
}
```

Problem 4: Program Optimization: Reduce Unwanted memory accesses. Assume level2v and level1v as float arrays

Where is the inefficiency? Fix it!

```
for(i=0;i<1e6;i++) {
level2v[i]+ =  0.5*(1+atan2(divide((level1v[i]+1.2),
18)));
level2v[i]+= 0.5*(1+atan2(divide((level1v[i]-2),30)));
level2v[i]+= divide(1,cos(divide((level1v[i]-2),60)))
}
```

Problem 5: Program Optimization: Loop Unrolling (do it 2 times for now). 4 times for your practice

In Problem 4 solution, Where is the inefficiency? Fix it!

In-Class Participation Problem: 5 minutes

Google search why excessive use of global variables is discouraged.

Google search: switch vs. if-else ladder. Which one is better for performance?

Programmer Optimizations

Third part of lab will step you through six code optimizations

1. Code motion
2. Reducing procedure calls
3. Eliminating memory accesses
4. Unrolling loops x2
5. Unrolling loops x3
6. Adding parallelism

Let’s Keep Coding!

LAB 5 AND LAB 6