Computer Systems and Networks

ECPE 170 – Jeff Shafer – University of the Pacific

Performance Optimization

Lab Schedule

Activities

- **Today**
  - Background discussion
  - **Lab 6 – Performance Optimization**
- **Next Tuesday**
  - **Lab 6 – Performance Optimization**
- **Next Thursday**
  - **Lab 7 – Memory Hierarchy**

Assignments Due

- **Lab 5**
  - **Due by Feb 23rd 5:00am**
- **Lab 6**
  - **Due by Mar 2nd 5:00am**
- **** Midterm Exam ****
  - **Thur Mar 5th**

Co-Person of the Day: Fran Allen

- IBM Research: 1957-2002
- Expert in **optimizing compilers** (i.e. compilers that optimize the program they produce)
- Expert in **parallelization**
- Winner of ACM *Turing Award*, 2006
  - *First female winner!*

Co-Person of the Day: Donald Knuth

- Author, *The Art of Computer Programming*
  - Algorithms, algorithms, and more algorithms!
- Creator of TeX typesetting system
- Winner, ACM *Turing Award*, 1974

LaTeX – Input

```
\documentclass[12pt]{article}
\usepackage{amsmath}
\title{\LaTeX}
\date{}
\begin{document}
  \maketitle
  \LaTeX{} is a document preparation system for the \TeX{}
  typesetting program. It offers programmable desktop publishing
  features and extensive facilities for automating most aspects of
  typesetting and desktop publishing, including numbering and
  cross-referencing, tables and figures, page layout, bibliographies,
  and much more. \LaTeX{} was originally written in 1984 by Leslie
  Lamport and has become the dominant method for using \TeX; few
  people write in plain \TeX{} anymore. The current version  is
  \LaTeXe.

  % This is a comment; it will not be shown in the final output.
  % The following shows a little of the typesetting power of LaTeX:
  \begin{align}
    E &= mc^2                              \\
    m &= \frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}}
  \end{align}
\end{document}
```

LaTeX – Output

LaTeX

Side Note:

*LATEX works **great** in version control systems!*

LaTeX is a document preparation system for the TeX typesetting program. It offers programmable desktop publishing features and extensive facilities for automating most aspects of typesetting and desktop publishing, including numbering and cross-referencing, tables and figures, page layout, bibliographies, and much more. LaTeX was originally written in 1984 by Leslie Lamport and has become the dominant method for using TeX; few people write in plain TeX anymore. The current version is LaTeX 2_ε.

$$E = mc^2 \tag{1}$$

$$m = \frac{m_0}{\sqrt{1 - \frac{v^2}{c^2}}} \tag{2}$$

Quotes – Donald Knuth

"**Computer programming is an art**, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better." – Donald Knuth

"Random numbers should not be generated with a method chosen at random." – Donald Knuth

Quotes – Donald Knuth

"People who are more than casually interested in computers should have at least **some idea of what the underlying hardware is like**. Otherwise the programs they write will be pretty weird."
– Donald Knuth

Remember this when we're learning MIPS assembly later this semester!

Performance Optimization

Vote

- **Who will do a better job improving program performance?**

- *The compiler* -vs- *The programmer*

Lab 6 Goals

1. What can the *compiler* do for programmers to improve performance?
2. What can *programmers* do to improve performance?

The Compiler

Compiler Goals

- **What are the compiler's goals with optimization off?**
- Obvious
  - Generate binary (executable) that produces correct output when run
  - Compile fast
- Less Obvious:
  - Make debugging produce expected results!
  - Statements are independent
    - If you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you expect from the source code

Compiler Goals

- **What are the compiler's goals with optimization on?**
- Reduce program **code size**
- Reduce program **execution time**
- **These may be mutually exclusive!**

Optimization Tradeoffs

- **What might we lose when we turn on optimization?**
- Compilation will take a lot longer
- Debugging is harder

Compiler Optimizations

- **Inline Functions**

```
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);
```

- **Pros?** Lower overhead
- **Cons?** Bigger binary
  (except for tiny functions – like this?)

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

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

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

Compiler Optimizations

- **What specific overhead exists here?**

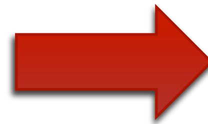
```
int max(int a, int b)
{
    if(a>b)
        return a;
    else
        return b;
}
```

- 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

Compiler Optimizations

- **Unroll Loops**

```
int x;
for (x = 0; x < 100; x++)
{
    delete(x);
}
```



```
int x;
for (x = 0; x < 100; x+=5)
{
    delete(x);
    delete(x+1);
    delete(x+2);
    delete(x+3);
    delete(x+4);
}
```

- **Pros?** Lower overhead
  Parallelism (potentially)
- **Cons?** Bigger binary

Compiler Optimizations

- **What specific loop <u>overhead</u> exists here?**

```
int x;
for (x = 0; x < 100; x++)
{
    delete(x);
}
```

- Top of loop
  - Compare x against 100
  - If less than, jump to …
  - Otherwise, jump to…
- Bottom of loop
  - Increment x by 1
  - Jump to top of loop
- *Impact on Branch Predictor (CPU microarchitecture)*

Compiler Optimizations

- A large number of common compiler optimizations won't make sense until we learn assembly code later this semester
  - The compiler is optimizing the *assembly code*, not the high-level source code

The Programmer

The Compiler –vs– The Programmer

- Humans can do a better job at optimizing code than the compiler
  - Tradeoff: many developer-hours of time

- Big picture idea: The compiler must be **safe** and only make optimizations that function for **all possible data sets**.
  - Even if the programmer knows that a particular corner case cannot happen, the compiler doesn't know that

The Compiler –vs– The Programmer

- **Is this optimization safe for a compiler to do?**

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

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

- Twiddle1() needs 6 memory accesses
  - 2x read xp
  - 2x read yp
  - 2x write xp
- Twiddle2() needs 3 memory accesses
  - Read xp
  - Read yp
  - Write xp

The Compiler –vs– The Programmer

- **What if *xp and *yp pointed to the same memory address?**
- Twiddle1()
  - *xp += *xp;
  - *xp += *xp; // ***xp increased 4x**
- Twiddle2()
  - *xp += 2 * *xp; // ***xp increased 3x**
- This is **memory aliasing** (two pointers to the same address), and is hard for compilers to detect
  - But the programmer can know whether aliasing is a concern!

The Compiler –vs– The Programmer

- **Is this optimization safe for a compiler to do?**

```
int f();

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

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

The Compiler –vs– The Programmer

- **Depends on what f() does!**

```
int counter = 0;

int f()
{
    return counter++;
}
```

- With func1(): 0+1+2+3 = **6**
- With func2(): 4*0 = **0**
- Hard for compiler to detect **side effects**

The Compiler –vs– The Programmer

- Compare two functions that convert a string to lowercase

```
void lower1(char *s)
{
  int i;

  for (i = 0; i < strlen(s); i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
        s[i] -= ('A' - 'a');
}
```

- **What does strlen() do again?**

```
void lower2(char *s)
{
  int i;
  int len = strlen(s);

  for (i = 0; i < len; i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
        s[i] -= ('A' - 'a');
}
```

- **Could the compiler make this optimization for us?**

The Compiler –vs– The Programmer

- **Could the compiler make this optimization for us?**
- Very hard!
  - strlen() checks the elements of each string…
  - … and the string is being changed as each letter is set to lowercase
  - Would need to determine that the null character is not being set earlier or later in string!

The Compiler –vs– The Programmer

- **An awesome compiler won't make up for a poor programmer**
  - *No compiler will ever replace a lousy bubble sort algorithm with a good merge sort algorithm*

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

Programmer Optimizations

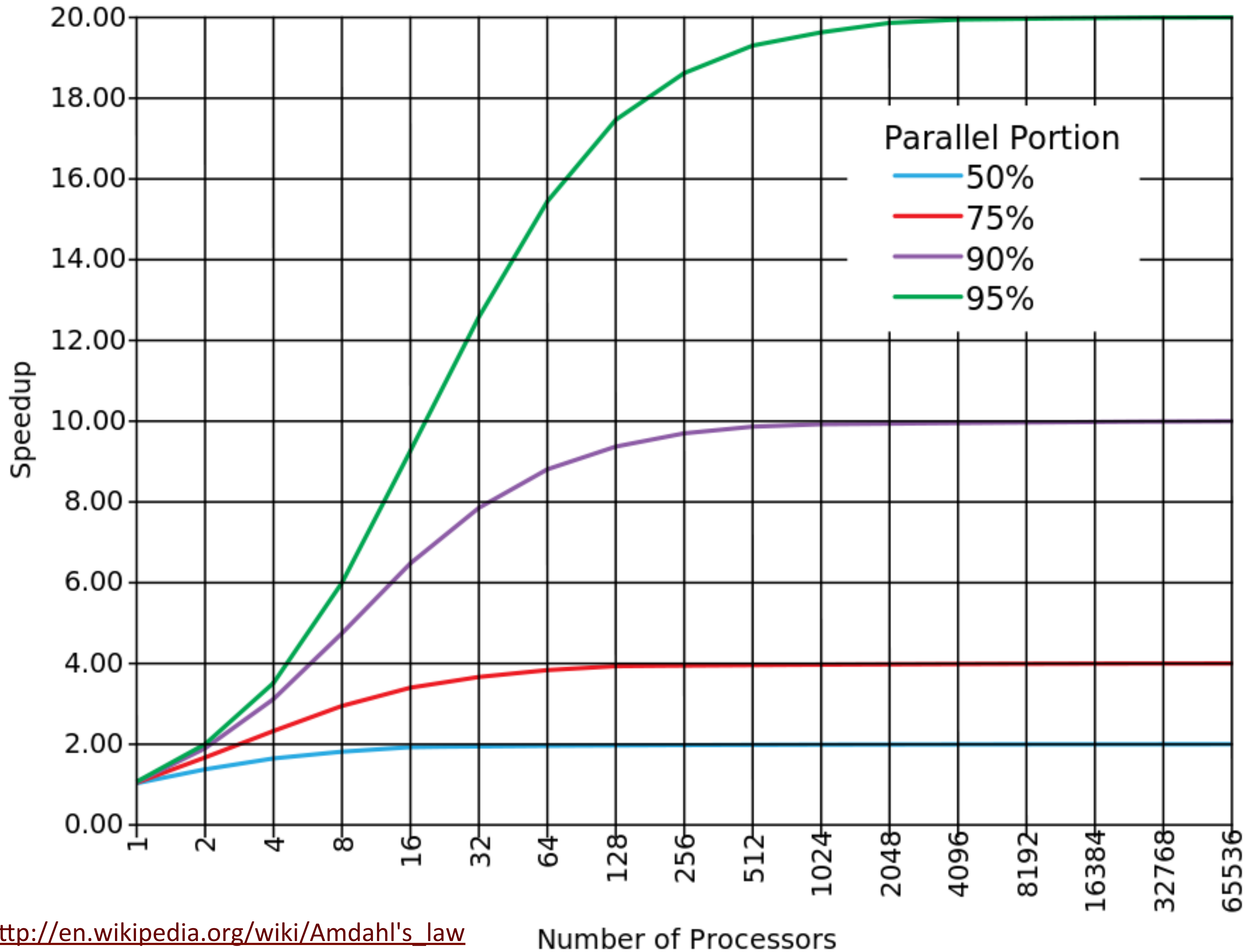
- **Should we use these optimizations everywhere?**
- **Beware of premature optimization!** Only spend effort optimizing if the performance monitoring tools point out that a particular algorithm/function is a bottleneck.
- "Premature optimization is the root of all evil (or at least most of it) in programming." - Donald Knuth
- **Amdahl's law**

Amdahl's Law

- The overall performance of a system is a result of the interaction of all of its components
- System performance is most effectively improved when the performance of the most heavily used components is improved - **Amdahl's Law**

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where *S* is the overall speedup;
f is the fraction of work performed by a faster component; and
k is the speedup of the faster component



http://en.wikipedia.org/wiki/Amdahl's_law

Lab Part 1

- **What does the combine1() function do?**
- Performs a mathematical operation across all elements in a vector
  - Possible operations: Multiply or Add
  - Possible vector types: Float, Double, Int, Long, Char
  - For example, **add** all elements of an array of **floats** together