Lecture 4: Compilers and Vector Operations + Creating User Accounts

0.1

Login to your team server

```
ssh root@129.24.245.{TeamID}
```

Install GNU system compiler

```
yum install g++
```

Richard Stallman
GNU (GNU is Not Linux)

GNU Tools are developed by the Free Software Foundation
Writing free tools based on UNIX utilities since 1983

Install GNU system compiler

```
yum install gfortran
```

Richard Stallman
GNU (GNU is Not Linux)

GNU Tools are developed by the Free Software Foundation
Writing free tools based on UNIX utilities since 1983

Install GNU system compiler

```
yum install emacs
```

Richard Stallman
GNU (GNU is Not Linux)

GNU Tools are developed by the Free Software Foundation
Writing free tools based on UNIX utilities since 1983

Install GNU system compiler

```
man emacs
```

Richard Stallman
GNU (GNU is Not Linux)

GNU Tools are developed by the Free Software Foundation
Writing free tools based on UNIX utilities since 1983

Install GNU system compiler

```
man emacs
```

You can use whatever text editor you want of course.

Vi is preinstalled on all Linux systems

If you don’t already know a linux editor, you can install nano with

```
yum install nano
```

Richard Stallman
GNU (GNU is Not Linux)

GNU Tools are developed by the Free Software Foundation
Writing free tools based on UNIX utilities since 1983

POSIX (Portable OS Interface) vs GNU

UNIX
UNIPLEXED
INFORMATION
COMPUTING SYSTEM

Vi is preinstalled because it was part of the 1970s POSIX Standard that helped define UNIX.

“Do one thing and do it well” philosophy

GNU tools were written to be a free clone of POSIX UNIX

“Everyone should be free to change their software”

Preinstalling emacs would stifle changes

Richard Stallman
GNU (GNU is Not Linux)

GNU Tools are developed by the Free Software Foundation
Writing free tools based on UNIX utilities since 1983

Install GNU Emacs

GNU software is copyleft – meaning it can be freely distributed and modified, as long as the modified code is also copyleft.

This means software derived from GNU free software must remain free.

That doesn't mean programmers can't make money from derived software but they can't restrict access.

Richard Stallman
GNU (GNU is Not Linux)

GNU Tools are developed by the Free Software Foundation
Writing free tools based on UNIX utilities since 1983

Install GNU Emacs

Richard Stallman
GNU (GNU is Not Linux)

Etc.

IBM owns RedHat Linux. But since it is based on GNU copyleft code they can't keep others from copying it.

Copyleft

```
[root@moonshine ~]# gfortran --version
GNU Fortran (GCC) 11.4.1 20230605 (Red Hat 11.4.1-2)
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying
conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
```

Create User Account

```
[root@moonshine ~]# adduser matthew
```

Create User Account

```
[root@moonshine ~]# adduser matthew
[root@moonshine ~]# passwd matthew
Changing password for user matthew.
New password:
Retype new password:
```

Create User Account

```
[root@moonshine ~]# adduser matthew
[root@moonshine ~]# passwd matthew
Changing password for user matthew.
New password:
Retype new password:
[root@moonshine ~]# exit
logout
```

Login to your team server

```
ssh matthew@129.24.245.{TeamID}
```

Login to your team server

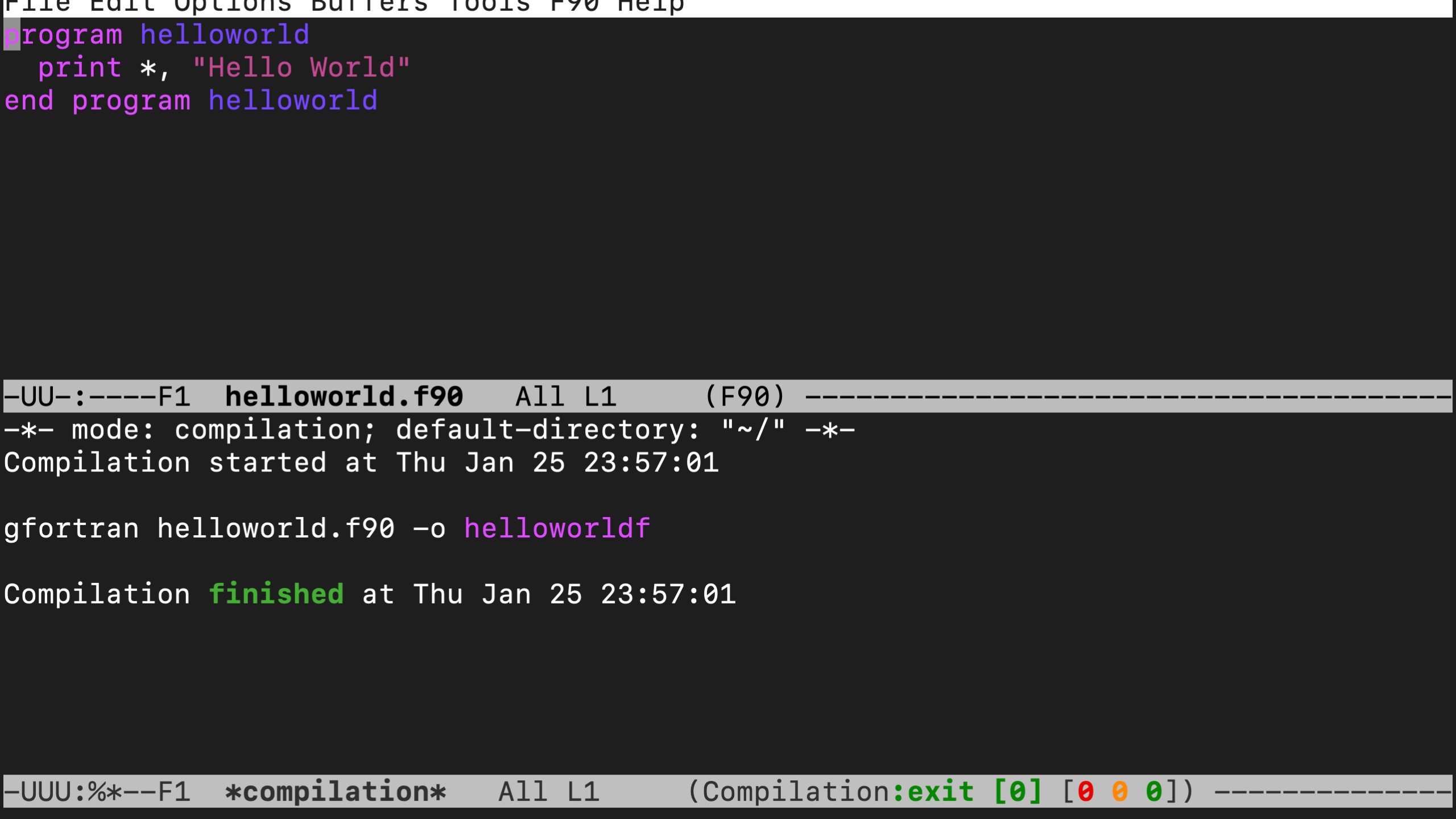
```
[matthew@moonshine ~]$ emacs helloworld.cpp
```

Write a simple C++ program

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello World" << endl;

  return 0;
}

-UU-:----F1  helloworld.cpp    All L1      (C++//l Abbrev) ----------------------
-*- mode: compilation; default-directory: "~/" -*-
Compilation started at Thu Jan 25 23:15:48

g++ helloworld.cpp -o helloworld

Compilation finished at Thu Jan 25 23:15:48

-UUU:%*--F1  *compilation*    All L1      (Compilation:exit [0] [0 0 0]) ------------
```

Run our C++ program

```
[matthew@moonshine ~]$ ./helloworld
Hello World
```


File Edit Options Buffers Tools F90 Help
program helloworld
print *, "Hello World"
end program helloworld
-UU-:----F1 helloworld.f90 All L1 (F90)
-*- mode: compilation; default-directory: "~/" -*-
Compilation started at Thu Jan 25 23:57:01
gfortran helloworld.f90 -o helloworldf
Compilation finished at Thu Jan 25 23:57:01
-UUU:%*--F1 *compilation* All L1 (Compilation:exit [0] [0 0 0])

Run our Fortran program

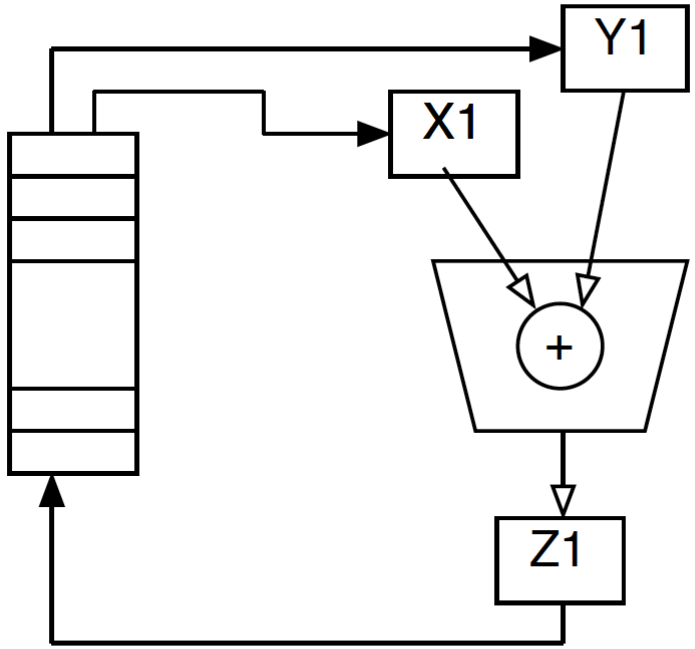
```
[matthew@moonshine ~]$ ./helloworldf
 Hello World
```

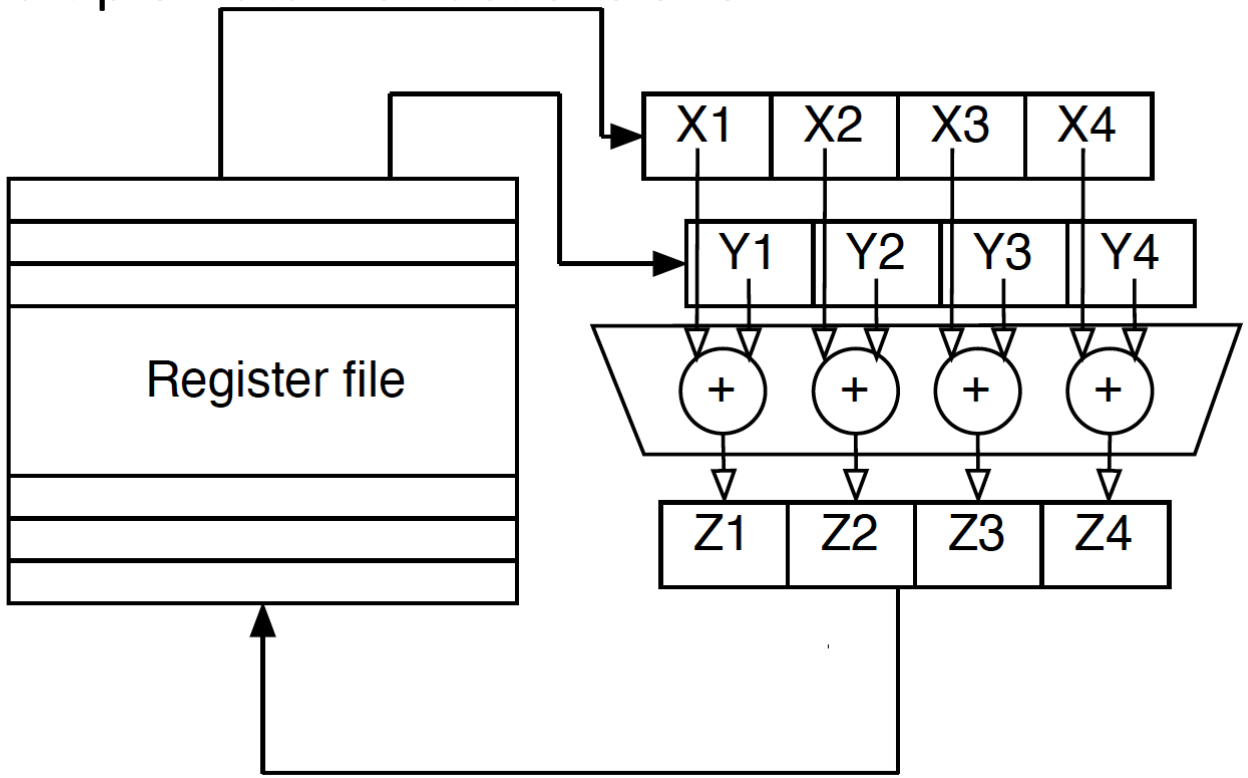
Normal non-vector operation array add (one at a time)

```
#define MAX_SIZE (1024 * 1024)

static unsigned short A[MAX_SIZE];
static unsigned short B[MAX_SIZE];

/* Usual vector add - iterate over the arrays one at at time and add */
void add(long size, unsigned short * a, const unsigned short *b) {
    for (int i = 0; i < size; ++i) {
        a[i] += b[i];
    }
}

int main()
{
  /* do it a lot so we can measure */
  for (int i = 0; i < 10e3; i++ )
    add(MAX_SIZE, A, B);
  return 0;
}
```

SSE Vector Operations take 128 bits of data (4 ints or floats/2 doubles), AVX 256 bit ops for floating point but only 128 for ints, AVX2 256 bits (8 ints or floats/4 doubles)

These are examples of Single Operation Multiple Data instructions or SIMD.





GNU C and FORTRAN provide many “intrinsics” so you can access those vector operations. https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html provides a complete list.

Normal non-vector operation array add (one at a time)

```
/* header file for Intel intrinsics */
#include <immintrin.h>
#include <smmintrin.h>

#define MAX_SIZE (1024 * 1024)

static unsigned short A[MAX_SIZE];
static unsigned short B[MAX_SIZE];

/* vectorized version --- 128-bit registers */
void add_SSE(long size, unsigned short * a, const unsigned short *b) {
    for (int i = 0; i < size; i += 8)
         {

        /* load 128 bits from a */
        /* a_part = {a[i], a[i+1], a[i+2], ..., a[i+7]} */
        __m128i a_part = _mm_loadu_si128((__m128i*) &a[i]);
        /* load 128 bits from b */
        /* b_part = {b[i], b[i+1], b[i+2], ..., b[i+7]} */
        __m128i b_part = _mm_loadu_si128((__m128i*) &b[i]);

         /* Use the SSE 128 bit addition operation */
        a_part = _mm_add_epi16(a_part, b_part);
        _mm_storeu_si128((__m128i*) &a[i], a_part);
    }
}
```

Compilation command is a bit complex to make sure the vector operation differences are visible

```
[matthew@moonshine ~]# gcc -g -O1 -std=c99 -mavx2 -mtune=skylake -Wall -fwrapv -fno-strict-aliasing regular_add.c -o regular_add
```

```
[matthew@moonshine ~]# gcc -g -O1 -std=c99 -mavx2 -mtune=skylake -Wall -fwrapv -fno-strict-aliasing sse_add.c -o sse_add
```

Hypothesis for speed increase:

`static unsigned short A[MAX_SIZE];`
Each unsigned short is **16 bits = 2 bytes**.

`_mm_add_epi16(a_part, b_part);` uses **128-bit registers to add 16 bit integers (shorts not ints)**

128 bits/ 16 bits = 8

Each `_mm_add_epi16` instruction adds **8 pairs of 16-bit integer values** (short) in one go.

What if we had `static int A[MAX_SIZE]` instead?

Always pay attention to the type definitions for your environment

Data Type	Bits per Element	Elements per 128-bit Register
`char` (8-bit)	8	16
`short` (16-bit)	16	8
`int` (32-bit)	32	4
`float` (32-bit)	32	4
`double` (64-bit)	64	2

```
rapv -fno-strict-aliasing
```

```
rapv -fno-strict-aliasing
```

ISO C Specification only says the following, but modern compilers have standardized on the sizes above

char is **always 1 byte (8 bits)**.
short is **at least 16 bits**
int is **at least 16 bits**
long is **at least 32 bits**
long long is **at least 64 bits**

What if we had `static int A[MAX_SIZE]` ins

Model	int	long	long long	void*	Used By
LP64	32	**64**	64	64	Linux, macOS, UNIX
LLP64	32	**32**	64	64	Windows x86_64
ILP64	64	64	64	64	Some HPC/Fortran libs
LP32	32	32	64	32	Old 32-bit Linux/Windows

Void* is a generic pointer, so it is how large memory addresses can be. This is what we mean by 32-bit or 64-bit operating systems.

Compilation command is a bit complex to make sure the vector operation differences are visible

```
[matthew@moonshine ~]# gcc -g -O1 -std=c99 -mavx2 -mtune=skylake -Wall -fwrapv -fno-strict-aliasing
regular_add.c -o regular_add
```

```
[matthew@moonshine ~]# gcc -g -O1 -std=c99 -mavx2 -mtune=skylake -Wall -fwrapv -fno-strict-aliasing
sse_add.c -o sse_add
```

Flag	Purpose
–O1	Basic optimisation
–mavx2	Enable AVX2 instructions*
–mtune=skylake	Optimise for specific CPU architecture
–fno–strict–aliasing	Prevents dangerous optimisations †

* We are using SSE instructions but compiling with AVX2. AVX2 includes SSE.

† Don’t assume pointers of incompatible types can’t refer to the same memory location. Belt and suspenders.

Does it matter?

```
[matthew@moonshine ~]# time ./regular_add

[matthew@moonshine ~]# time ./sse_add
```

Does it matter?

```
[matthew@moonshine ~]# time ./regular_add


real 0m9.587s
user 0m9.565s
sys 0m0.001s


[matthew@moonshine ~]# time ./sse_add


real 0m1.447s
user 0m1.441s
sys 0m0.002s
```

Does it matter?

```
[matthew@moonshine ~]# time ./regular_add


real 0m9.587s
user 0m9.565s
sys 0m0.001s


[matthew@moonshine ~]# time ./sse_add


real 0m1.447s
user 0m1.441s
sys 0m0.002s
```

With SSE the code is 6.6 times faster.*

* Why not 8 times faster?

Here are just some of the reasons we don’t get 8x speedup

- Memory bandwidth limitations
- Memory latency
- Unaligned memory access
- Loop overhead
- Load and store instructions
- Instruction latency
- Pipeline stalls
- Wall-clock benchmarking artifacts
- Instruction-level parallelism limits
- Scalar remainder handling (N % 8 ≠ 0)
- Vector setup and teardown cost
- Cache effects
- Compiler auto-vectorisation of scalar baseline
- Suboptimal compiler optimisation level
- Branch prediction overhead
- System noise and context switching

```
sys 0m0.001s

[matthew@moonshine ~]# time ./sse_add

real 0m1.447s
user 0m1.441s
sys 0m0.002s
```

* Why not 8 times faster?

Sudoers

The $ means you are logged into a user account.
If it were a # you would know it is a root account.

```
[matthew@moonshine ~]$ yum install tar
Error: This command has to be run with superuser privileges
(under the root user on most systems).
[matthew@moonshine ~]$
```

The user accounts you created can't do administrative tasks.

We will allow the user accounts to run administrative commands. Because:

1. Accountability – everyone logged into root looks the same
2. The root account is often disabled because it is such a juicy target
   your root accounts already have thousands of attacks peer week.

Sudoers

```
[root@moonshine ~]# visudo
```

We are going to modify the file that controls account permissions.
/etc/sudoers

Modifying the sudoers file is dangerous. If it is formatted incorrectly you could lock yourself out of the system entirely. You can use a regular text editor but the visudo command is intended for this purpose.

Visudo does some basic checks to make sure the sudoers file is still valid when you exit.

*more than 4 times due to loop unrolling

Sudoers

```
## Sudoers allows particular users to run various commands as
## the root user, without needing the root password.
##
## Examples are provided at the bottom of the file for collections
## of related commands, which can then be delegated out to particular
## users or groups.
##
## This file must be edited with the 'visudo' command.


## Host Aliases
## Groups of machines. You may prefer to use hostnames (perhaps using
## wildcards for entire domains) or IP addresses instead.
# Host_Alias     FILESERVERS = fs1, fs2
# Host_Alias     MAILSERVERS = smtp, smtp2


## User Aliases
## These aren't often necessary, as you can use regular groups
## (ie, from files, LDAP, NIS, etc) in this file - just use %groupname
## rather than USERALIAS
# User_Alias ADMINS = jsmith, mikem



## Command Aliases
"/etc/sudoers.tmp" 120L, 4349B                                1,1           Top
```

We are going to scroll down to the section where root gets its privileges and copy them.

(a better way would be to add the users to the wheel group but I want you to see the sudoers file)

Sudoers

```
## The COMMANDS section may have other options added to it.
##
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
matthew ALL=(ALL)       ALL
## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOCATE, DRIVERS


## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)       ALL


## Same thing without a password
# %wheel        ALL=(ALL)       NOPASSWD: ALL


## Allows members of the users group to mount and unmount the
## cdrom as root
# %users  ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom


## Allows members of the users group to shutdown this system
# %users  localhost=/sbin/shutdown -h now
```

We are going to scroll down to the section where root gets its privileges and copy them.

(a better way would be to add the users to the wheel group but I want you to see the sudoers file)

That was a yucky way to get super powers…

The better way is to add users to the wheel* group:

As root: usermod -aG wheel username

We are going to scroll down to the section where root gets its privileges and copy them.

(a better way would be to add the users to the wheel group but I want you to see the sudoers file)

Wheel comes from UNIX slang. People with superpowers were “big wheels”.

Next Time

- Linux and Devices