# Lecture 3

Advanced Programming Topics



# Topics

- Pre-processor directives
- Defining macros through compilation flags
- External Libraries & Functions
- Makefile
- Cache blocking



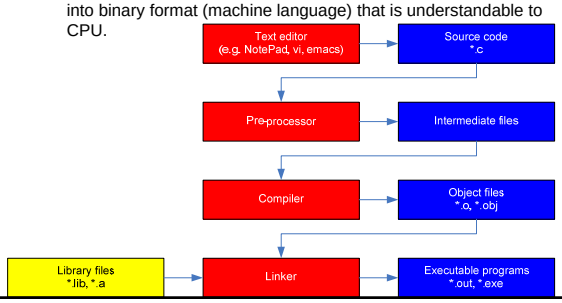
# Pre-processor Directives

#define / #undef macro
#include
conditional compilation



# How executable files are generated

- Compiled
  - Programs written in C language are translated through compiler into binary format (machine language) that is understandable to CPU.




# #define / #undef

- Define a macro with certain “value”, and when source code is pre-processed, the macro/label will be replaced by its value.

/home/course/Lecture03/01.cpp

```
#define N 1000

int main() {
    int a[N][N];

    for(int i=0;i<N;i++) {
        for(int j=0;j<N;j++) {
            a[i][j]=0;
        }
    }
}

#undef N
```

```
int main() {
    int a[1000][1000];

    for(int i=0;i<1000;i++) {
        for(int j=0;j<1000;j++){
            a[i][j]=0;
        }
    }
}
```

# #define / #undef

- Macro can also have arguments, and it looks like functions.

```
#define SQUARE(x) ((x) * (x))
#define MAX(x,y) ((x)>(y)?(x):(y))
#include <iostream>

int main() {
    std::cout << "\n5^2=" << SQUARE(5);
    std::cout << "\nMAX(3,5)=" << MAX(3,5);

    return 0;
}
```

/home/course/Lecture03/02.cpp

```
int main() {
    std::cout << "\n5^2=" << ((5) * (5)) ;
    std::cout << "\nMAX(3,5)=" << ((3)>(5)?(3):(5));

    return 0;
}
```

# #include

- To add the content of other files into the current file at the place of the #include directive before compilation.

**abc.inc**

```
int n=3;
int j=4;
```

```
int main() {
    int n=3;
    int j=4;

    std::cout << "\nn=" << n << ", j=" << j;
    return 0;
}
```

**myProg.c**

```
int main() {
#include "abc.inc"

    std::cout << "\nn=" << n << ", j=" << j;
    return 0;
}
```



# Conditional Compilation

- #ifdef … #else … #endif
- #ifndef … #else … #endif

/home/course/Lecture03/03.cpp

```
#define BETA
#include <iostream>
int main() {
#ifdef BETA
    std::cout << "\nThis is the beta version";
#else
    std::cout << "\nThis is the full version";
#endif
    return 0;
}
```

```
int main() {
    std::cout << "\nThis is the beta version";
    return 0;
}
```



# Defining macros through compilation flags



## 1. Define DATA type outside the source code

/home/course/Lecture03/04.cpp

```
#include <iostream>
using namespace std;
int main() {
  DATA a[100];
  cout << "a occupies " << sizeof(a)
            << " bytes." << endl;
  return 0;
}
```

g++ 01.cpp -D DATA=int
g++ 01.cpp -D DATA=double
g++ 01.cpp -D DATA="unsigned char"



## 2. Use with conditional compilation

/home/course/Lecture03/05.cpp

```
#include <iostream>
using namespace std;

int main() {
  #ifdef LIMIT
    cout << "The trial version supports 10 integers.";
    int data[10];
  #else
    cout << "The full version supports 200000 integers.";
    int data[200000];
  #endif
  data[0] = 5;
  return 0;
}
```

g++ 02.cpp
g++ 02.cpp –DLIMIT



# External Functions & Libraries

## /home/course/Lecture03/06.cpp

```
#include "stopWatch.h"
int count=0;
double t=0;
double y, ang=24.0;
stopWatch timer;
do {
  timer.start();
  y=log10(ang);
  timer.stop();
  t += timer.elapsedTime();
  count++;
} while(t<0.1);

std::cout << "\nThis machine can do " << ((double)
 count) / t << " log10 operations per second. ";
```

13

## External Functions & Libraries

- Having a 06.cpp, stopWatch.h, and stopWatch.o
- **Method 1: (N/A)**
  - g++ –Wall 06.cpp stopWatch.cpp –o 06.exe
  - SOURCE CODE NOT PROVIDED!
- **Method 2:**
  - g++ -c stopWatch.cpp
  - g++ -Wall 06.cpp stopWatch.o –o 06.exe
  - Note that the order IS important

14

## External Functions & Libraries

- **Method 3:**
  - g++ –c stopWatch.cpp
  - ar –q libMylib.a stopWatch.o
    *Create a statically linked library "libMyLib.a" with stopWatch.o in the library*
  - g++ –Wall 06.cpp libMylib.a –o 06.exe
    g++ –Wall 06.cpp -lMylib –o 06.exe
    g++ –Wall 06.cpp –lMylib –LmyPath –o 06.exe

15

## Makefile

16

## Makefile

- Makefile is a kind of script to describe how to compile a program or a series of related programs
- Nowadays make is mostly GNU Make
  - http://www.gnu.org/software/make/make.html
  - Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files.
- Command 'make' in UNIX/Linux first searches the current directory for makefile or Makefile to generate your code
  - You may specify alternative makefiles by using 'make -f *mymakefile*'

17

## Scenario

Assignment #2
Question 1
Question 2
Question 3
example.c

→ stopWatch.cpp
watchStart()
watchStop()
elapsedTime()

```
g++ q2.cpp stopWatch.c –o q2.exe
g++ q3.cpp stopWatch.c –o q3.exe
g++ example.cpp stopWatch.c –o example.exe
```

```
g++ stopWatch.c
g++ q2.cpp stopWatch.o –o q2.exe
g++ q3.cpp stopWatch.o –o q3.exe
g++ example.cpp stopWatch.o –o example.exe
```

18

## Example

```
g++ stopWatch.c
g++ q2.cpp stopWatch.o –o q2.exe
g++ q3.cpp stopWatch.o –o q3.exe
g++ example.cpp stopWatch.o –o example.exe
```

```
Target: source file(s)
        command (must be preceded by a tab)
```

```
all: q2.exe q3.exe example.exe

q2.exe: q2.cpp stopWatch.o
[TAB]   g++ q2.cpp stopWatch.o –o q2.exe

q3.exe: q3.cpp stopWatch.o
[TAB]   g++ q3.cpp stopWatch.o –o q3.exe

example.exe: example.cpp stopWatch.c
[TAB]   g++ example.cpp stopWatch.o –o example.exe

stopWatch.o: stopWatch.c
[TAB]   g++ –c stopWatch.c
```

19

## Example

```
# Define compiler & flags
CC=g++
CFLAGS=-Wall -O2

# Define projects to be built
all: q2.exe q3.exe example.exe

# Defines rules for compiling the project
q2.exe: q2.cpp stopWatch.o
      $(CC) $(CFLAGS) q2.cpp stopWatch.o –o q2.exe

q3.exe: q3.cpp stopWatch.o
      $(CC) $(CFLAGS) q3.cpp stopWatch.o –o q3.exe

example.exe: example.cpp stopWatch.o
      $(CC) $(CFLAGS) example.cpp stopWatch.o –o example.exe

stopWatch.o: stopWatch.c
      $(CC) $(CFLAGS) –c stopWatch.c
```

20

## Example - Macros

```
# Define compiler & flags
CC=g++
CFLAGS=-Wall -O2

OBJS=stopWatch.o
all: q2.exe q3.exe example.exe

# Defines rules for compiling the project
q2.exe: q2.cpp $(OBJS)
      $(CC) $(CFLAGS) q2.cpp $(OBJS) –o q2.exe

q3.exe: q3.cpp $(OBJS)
      $(CC) $(CFLAGS) q3.cpp $(OBJS) –o q3.exe

example.exe: example.c $(OBJS)
      $(CC) $(CFLAGS) example.cpp $(OBJS) –o example.exe

stopWatch.o: stopWatch.c
      $(CC) $(CFLAGS) –c stopWatch.c
```

21

## Special macros

CC: Contains the current C compiler. Defaults to cc.

CFLAGS: Special options which are added to the built-in C rule.

$@: Full name of the current target.

$?: A list of files for current dependency which are out-of-date.

$<: The source file of the current (single) dependency.

22

## Example – Special macros

```
# Define compiler & flags
CC=g++
CFLAGS=-Wall –O2

OBJS=stopWatch.o
all: q2.exe q3.exe example.exe

# Defines rules for compiling the project
q2.exe: q2.cpp $(OBJS)
      $(CC) $(CFLAGS) $< $(OBJS) –o $@

q3.exe: q3.cpp $(OBJS)
      $(CC) $(CFLAGS) $< $(OBJS) –o $@

example.exe: example.cpp $(OBJS)
      $(CC) $(CFLAGS) $< $(OBJS) –o $@

stopWatch.o: stopWatch.c
      $(CC) $(CFLAGS) –c $<
```

23

## Example – Implicit Rules

```
# Define compiler & flags
CC=g++
CFLAGS=-Wall –O2

OBJS=stopWatch.o
all: q2.exe q3.exe example.exe

%.exe: %.cpp $(OBJS)
      $(CC) $(CFLAGS) $< $(OBJS) –o $@
%.o: %.c
      $(CC) $(CFLAGS) –c $<
```

24

## Cache Blocking

25

## Efficient code

- We have introduced several loop transformation techniques and data-type considerations last week.
- Essentially, efficient code =
  - Minimal effort (e.g. move invariants out of loop!)
  - Use cache memory → **locality**
  - Good compiler!
- There are two kinds of locality
  - Spatial locality → continuous memory access
  - Temporal locality → blocking

26

## Blocking

- Making efficient use of cache memory
  - Once a data is read, use it as many times as possible (cache-reuse)
- Classical example: matrix-matrix multiplication

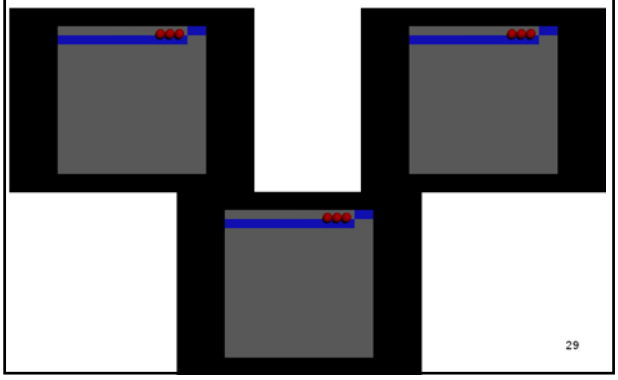
27

## Matrix-matrix multiplication naïve version

```
typedef unsigned int uint;
void naive(const uint N, stopWatch &t) {
  double a[N][N];
  double b[N][N];
  double c[N][N];
  init(N, &a[0][0], &b[0][0], &c[0][0]);

  for(uint i=0;i<N;i++)
    for(uint j=0;j<N;j++)
      for(uint k=0;k<N;k++)
        c[i][j] += a[i][k] * b[k][j];
}
```

/home/courses/Lecture03/07.cpp

## Naïve version



29

## Matrix-matrix multiplication with Blocking

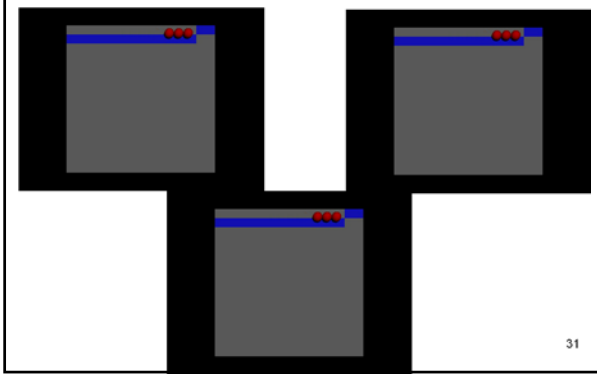
```
const uint blk=16; ← need to be tuned empirically

for(uint i0=0;i0<N;i0+=blk)
    for(uint j0=0;j0<N;j0+=blk)
      for(uint k0=0;k0<N;k0+=blk)
        for(uint i=i0;i<min(i0+blk,N);i++)
          for(uint j=j0;j<min(j0+blk,N);j++)
            for(uint k=k0;k<min(k0+blk,N);k++)
              c(i, j) += a(i, k) * b(k, j);
```

/home/courses/Lecture03/07.cpp

30

Reference: http://www.netlib.org/utk/papers/autoblock/node2.html

## Blocking version



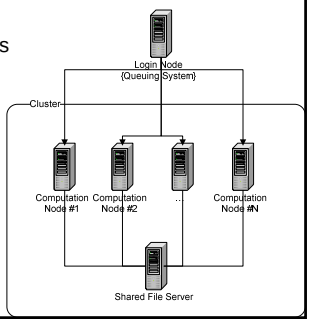
31

## IT Group Cluster2

32

## What is a cluster?

- A cluster is a dedicated resource for running computational tasks.
  - A collection of computers




## IT Group Cluster2 (1/2)

- Base system: Gentoo Linux (http://www.gentoo.org)
  - 8 Pentium D930 (Dual core, 3.0GHz) nodes
  - 4 Core 2 dual E6320 (dual core, 1.86GHz) nodes
  - All equipped with 4GB RAM (available memory varies)
- Queuing system: SLURM (http://www.llnl.gov/linux/slurm/)
  - 2 public queues (Public, Core) with 30-minute limits
  - 1 private queue without time limit
  - First come, first serve.
- Monitoring system: Ganglia (http://ganglia.sourceforge.net/)
  - http://140.118.5.6:8000

34

## IT Group Cluster2 (2/2)

- **Programming environment**
  - Compilers
    - GNU Compiler Collection suite (gcc) / gcc, g++
    - Intel C/C++/Fortran Compilers
    - *PathScale, PGI*
  - Auxiliary utilities
    - make, gdb, valgrind, gprof, …
  - Supporting programming libraries
    - MPI: OpenMPI, MPICH2
    - Numerical libraries: Intel MKL, AMD ACML, …

35

## Important Commands / Queuing

- **sinfo** → get information about the queue

```
ymhsieh@n00 ~ $ sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
Public*      up      30:00     8   idle n[00-07]
Core         up      30:00     4   idle c[00-03]
```

```
ymhsieh@n00 ~ $ sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
Public*      up      30:00     4  alloc n[01-04]
Public*      up      30:00     4   idle n[00,05-07]
Core         up      30:00     4   idle c[00-03]
```

alloc: allocated / occupied

36

## Important Commands / Queuing

- **squeue** → information about the queue

```
ymhsieh@n00 ~/Work/04_Test/MPI $ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES
NODELIST(REASON)
   1008    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1009    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1010    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1006    Public  startup  ymhsieh   R       0:04      3 n[01-03]
   1007    Public  startup  ymhsieh   R       0:03      3 n[04-06]
```

```
PD: Pending
R: Runing
```



## Important Commands / Queuing

- **scancel** → cancel a job in the queue

```
ymhsieh@n00 ~/Work/04_Test/MPI $ scancel 1011
ymhsieh@n00 ~/Work/04_Test/MPI $ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON
   1010    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1012    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1013    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1014    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1008    Public  startup  ymhsieh   R       0:38      3 n[01-03]
   1009    Public  startup  ymhsieh   R       0:37      3 n[04-06]
```



## Important Commands / Queuing

- **sbatch** (batch processing) → submit a *job script* for later execution. The script will typically contain one or more srun commands to launch parallel tasks.
  - sbatch -n8 /opt/mpich2/startup ./myProg.exe
- **srun** (interactive processing) → submit a job for execution or initiate job steps in interactive mode.
  - srun ./myProg.exe
  - srun -n8 ./myProg.exe
  - srun -n8 ./myProg.exe 200
- Complete Reference: https://computing.llnl.gov/linux/slurm/quickstart.html



## Summary

- sinfo
- squeue
- scancel
- sbatch
- srun