Compsci 590.3:
Introduction to Parallel Computing

Alvin R. Lebeck

Some slides based on those from the University of Oregon
Admin

Logistics
- Remote Access
  - Ignore xterm stuff
- Unix Tutorial
- C / C++ tutorial
  - Use icc or icpc
- SSH Keys
- Gitlab
  - Must use shibboleth authentication
- See me at end of class if troubles
- Homework 0
  - Serial Hello World
  - Parallel Hello World (after today)

- Homework Groups
  - Will send a link for spread sheet
  - Use piazza if you need to find partners
- HW groups assigned to specific machines and Phis…to reduce conflict
- Lectures generally alternate between
  1. Theory: Patterns / concepts
  2. Practice: Models / paradigms / systems

Outline
- Intro to Shared Memory
- OpenMP
- Cilk
- Threading Building Blocks (TBB)
What is Parallel Programming?

• Writing programs that exploit multiple hardware resources simultaneously to solve a given problem.
• This is not just concurrency!
  ▪ There is concurrency on computers that can do only one operation at a time---Operating System support for switching.
• It is not just having parallel hardware!
  ▪ You can write programs that are not parallel and run many of them on a computer with many processors.
• Concurrency + parallel hardware = parallel processing
• Writing programs for parallel processing = parallel programming
5. What are some issues?

- Specify the amount of parallelism?
- Name a datum across processors?
- Communicate values?
- Coordinate and synchronize (wait for a value to be produced)?
- We want language and runtime support to make this easy!
Simple Problem

for i = 1 to N
    A[i] = (A[i] + B[i]) * C[i]
    sum = sum + A[i]

• Split the loops
  » Independent iterations

    for i = 1 to N
        A[i] = (A[i] + B[i]) * C[i]
    for i = 1 to N
        sum = sum + A[i]
Shared-Memory Programming Model

- Provide a shared-memory abstraction
  - Familiar and efficient for programmers
  - Any processor can assign value to a variable: e.g., SSN = 123456789;
Shared Memory System

- Communication, sharing, and synchronization with loads/stores (reads/writes) on shared variables
- Private regions
- Shared regions

![Diagram of Shared Memory System]

© Alvin R. Lebeck

Compsci 590.3 Parallel Computing
Return of the Simple Problem (Shared Memory)

private int i, my_start, my_end, mynode;
shared float A[N], B[N], C[N], sum;
for i = my_start to my_end
    A[i] = (A[i] + B[i]) * C[i]
GLOBAL_SYNCH;
if (mynode == 0)
    for i = 1 to N
        sum = sum + A[i]

- Can run this pseudo code on any machine that supports shared memory
- Next up programming environments to do this!
  - language and systems
Three Parallel Programming Environments

• OpenMP
  ▪ [http://openmp.org](http://openmp.org)
  ▪ A good tutorial [https://computing.llnl.gov/tutorials/openMP/](https://computing.llnl.gov/tutorials/openMP/)

• Cilk Plus
  ▪ [https://www.cilkplus.org/](https://www.cilkplus.org/)

• Threading Building Blocks (TBB)
  ▪ [https://www.threadingbuildingblocks.org/](https://www.threadingbuildingblocks.org/)

• Enable expressing parallelism in C/C++
  ▪ Language extensions and/or pragmas
  ▪ Goal is to make it easier to write parallel programs
  ▪ Generic Parallel tasks
  ▪ Parallel for loops
  ▪ Many other things too…
Task Parallelism

- General execution model for each paradigm
- Language extensions / pragmas to specify tasks that may execute concurrently
- Tasks are executed by SW threads (!= HW Threads)
  - Generally 1:1 Tasks to Threads, but not always
- Support for coordination of the tasks
- Example pseudo code (i.e., not real code):

1. `printf("I'm the one\n");`
2. `do in parallel`
3. `{`
4. `printf("No, I'm the one\n");`
5. `}`
6. `printf("But it is really me\n");`

- Master task
- Can have nested parallelism

© Alvin R. Lebeck
Task Parallelism Continued

What is happening in the runtime?

1. printf(“I’m the one\n”);
2. for 0 to 3 {
3.   create task {
4.     printf(“No, I’m the one\n”);
5.   }
6. }
7. Wait for tasks
8. printf(“But it is really me\n”);
1. `printf("I'm the one\n");`
2. `int y = -1;`
3. `do in parallel`
4. `{`
5. `int x = 0;`
6. `if (my_thread_ID == 0)`
7. `y = 0;`
8. `else`
9. `x = my_thread_ID;`
10. `while (y == -1);`
11. `}`
12. `printf("%d says %d is the one…\n", x, y);`
13. `}`
14. `printf("But it is really me\n");`

- Might be ok to signal an event
- Lots of potential issues arise when you do this
- More later…
The OpenMP simple program

#include <omp.h>
void main()
{
    double A[N], B[N], C[N];
    // initialize A B and C

    #pragma omp parallel for
    for (i=0; i<N; i++) {
        A[i] = (A[i] + B[i]) * C[i];
    }
    for (i=0; i<N; i++){
        sum += A[i];
    }
    // print some result
}

• Include the OpenMP definitions
• #pragma compiler directive
  ▪ Tells the compiler to handle this specially
• omp indicates this is an OpenMP directive
• parallel indicates this is a parallel construct
  ▪ Applies only to following statement or {} enclosed statements
• for indicates this is a parallel for loop
• System automatically divides work among
• Global sync is implicit in parallel for
• To compile this program use:
  ▪ icpc simple_openmp.c -o simple_openmp
    –fopenmp
The OpenMP API

• Three main components
1. Directives
   ▪ Express parallelism
   ▪ E.g., #pragma omp parallel { code block }
2. Runtime system
   ▪ Can set/retrieve state of system
   ▪ E.g., omp_get_thread_num();
3. Environment Variables
   ▪ Control runtime
   ▪ setenv OMP_NUM_THREADS 32
   ▪ export OMP_NUM_THREADS=32

Directives (examples)
• #pragma omp parallel
   { .... }
• #pragma omp parallel for
   for (int i= 0; i< N; i++) { ... }
• #pragma omp cancel
   ▪ Cancels the parallel construct (generally for error handling, but may have other uses)
   ▪ #pragma omp parallel
     { ....
      #pragma omp cancel
     ....} 
• Others…will cover as needed, refer to docs
The OpenMP API

Runtime System (Examples)
• omp_get_thread_num();
• omp_set_num_threads();
• omp_get_wtime();
• Etc.
printf("my ID is %d\n", omp_get_thread_num());

Environment Variables (Examples)
• OMP_NUM_THREADS
• OMP_NESTED
• OMP_MAX_ACTIVE_LEVELS

• BASH
export OMP_NUM_THREADS=32
• CSH/TCSH
setenv OMP_NUM_THREADS 32
The Cilk (Plus) simple program

```c
#include <cilk/cilk.h>
void main()
{
    double A[N], B[N], C[N];
    // initialize A B and C

cilk_for (i=0; i<N; i++) {
    A[i] = (A[i] + B[i]) * C[i];
}
for (i=0; i<N; i++){
    sum += A[i];
}
// print some result
}
```

- Include the Cilk definitions
- `cilk_for` indicates this is a cilk parallel for loop
- System automatically divides work among
- Global sync is implicit in parallel for
- To compile this program use:
  - `icpc simple_cilk.c -o simple_cilk`
The Cilk (Plus) API

- Three main components

1. Keywords
   - Express parallelism
   - E.g., cilk_for

2. Runtime system
   - Can set/retrieve state of system
   - E.g., __cilkrts_get_worker_number();

1. Environment Variables
   - Control runtime
   - setenv CILK_NWORKERS 32
   - export CILK_NWORKERS =32

Keywords (examples)

- cilk_spawn
  for (int i= 0; i< N; i++) {
    cilk_spawn printf …
    … }

- cilk_sync
  - Wait for spawned tasks to finish (explicit sync)

- cilk_for
  cilk_for (int i= 0; i< N; i++) { printf … }

- Others…will cover as needed, refer to docs
The Cilk API

Runtime System (Examples)
#include <cilk/cilk_api.h>
__cilkrts_get_worker_number();
__cilkrts_get_get_nworkers();
__cilkrts_get_get_total_workers();
• Others…
printf("my ID is %d\n",
    __cilkrts_get_worker_number());

• Environment Variables (Examples)
• CILK_NWORKERS

• BASH
export CILK_NWORKERS =32
• CSH/TCSH
setenv CILK_NWORKERS 32
The TBB simple program (version 1)

```c
#include <tbb/tbb.h>

double A[N], B[N], C[N];

struct Summer {
  void operator() (const blocked_range<int> &r) const {
    for (int i=r.begin(); i<r.end(); i++){
      A[i] = (A[i] + B[i]) * C[i];
    }
  }
};

void main() {
  // initialize A B and C
  Summer sumit;
  parallel_for(blocked_range<int>(0,N), sumit);
  // print some result
}
```

- **Include the TBB definitions**
- **struct Summer** defines a structure with no data, but an operator () for that type
  - (const blocked_range<int> &r) is the argument list for the function
  - const {…} is body of the function
- **parallel_for** (blocked_range<int>(0,N), sumit)
  - Indicates this is a parallel for loop
  - Applies only to called function
  - System automatically divides work (N) among threads
  - Each worker gets a unique begin / end
- **To compile this program use:**
  - icpc simple_tbb.c -o simple_tbb -ltbb
Lambda Expression

- Create an unnamed function inline
- Syntax: `[] ( ) { };`
- `[]`
  - Capture Clause
  - What variables from current scope
  - `[]` Capture nothing
  - `[=]` Capture by value
  - `[&]` Capture by reference
  - `[=, &x]` Capture by reference, x by reference
  - `[name]` Capture name by value, but nothing else
- `( )`
  - Argument list
  - Optional if no arguments
- `{ };`
  - Function body;
  - Return type is deduced by compiler

Example
```
int x=3, y=7, z =0;
auto sum = [ ] (int a, int b) { return a+b;};
z = sum(x,y);
printf(“sum is %d\n”,z);
```
The TBB simple program (version 2)

#include <tbb/tbb.h>
void main()
{
    double A[N], B[N], C[N];
    // initialize A B and C
    parallel_for(blocked_range<int>(0,N),
        [&](blocked_range<int> r) {
            for (int i=r.begin(); i<r.end(); i++){
                A[i] = (A[i] + B[i]) * C[i];
            }
        })
    for (i=0; i<N; i++){
        sum += A[i];
    }
    // print some result
}

• Include the TBB definitions
• parallel_for(blocked_range<int>(0,N),
    - indicates this is a parallel for loop
    - Applies to a function…
• lambda expression
• Unnamed function object
• [&] capture all variables by reference
  - = : capture by value, &A A by reference, etc.
• (blocked_range<int> r) input parameter
• { … } body of function
• System automatically divides work
  - Each worker gets a unique begin / end
• To compile this program use:
  - icpc simple_tbb.c -o simple_tbb –ltbb

TBB API

- `task_scheduler_init`  
  - Initialize scheduler, set params  
  - Useful if you want control, we will usually ignore
- `tick_count`  
  - timing

Environment Variables

- `TBB_USE_DEBUG`  
  - 1 = enable TBB debugging features
Running on the Xeon Phi

• Two methods
• Native and Offload

Native
1. export MIC_ENV_PREFIX=MIC
2. export SINK_LD_LIBRARY_PATH=/opt/intel/composer_xe_2015.3.187/compiler/lib/mic/
3. Compile program with –mmic compiler option
   1. e.g., icpc -std=c++11 -mmic foo.c -o foo –fopenmp
4. Use mcnativeloadex program to launch executable on a Phi
   • e.g., mcnativeloadex foo
   • e.g., mcnativeloadex foo -d 1 // says to run on device 1
   • Try this with hello world

• Offload later.