OpenMP

An Easy Way for Parallel Programming

Chuizheng Meng

May 17, 2016

What is OpenMP

- OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify high-level parallelism in Fortran and C/C++ programs.
- OpenMP is designed for Fortran, C and C++.
- Supported compilers
  - GCC($\geqslant$ 4.2.0)
  - Visual Studio 2008-2010 C++
  - Clang($\geqslant$ 3.8)
  - IBM, Intel, Texas Instrument...
- Only a supported compiler is needed!

Hello OpenMP

hello.cpp

```cpp
#include <stdio.h>

int main() {
    #pragma omp parallel
    printf("Hello OpenMP!\n");
    return 0;
}
```

Compile & Run(with flag "-fopenmp")

```
g++ -Wall -Werror -O3 hello.cpp -o hello -fopenmp
./hello
```

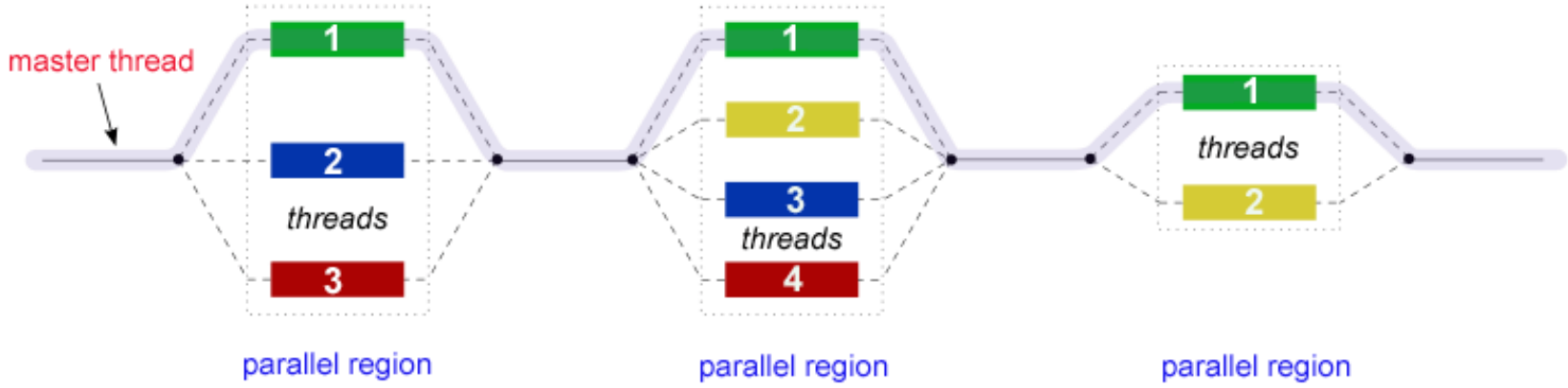
Hello OpenMP

Result

```
neozero ~ oop2016 week13pre ./hello
Hello OpenMP!
Hello OpenMP!
Hello OpenMP!
Hello OpenMP!
neozero ~ oop2016 week13pre
```

How OpenMP Works

Fork-Join Model



- All OpenMP programs begin as a single process: the master thread.
- Fork
- Join

How OpenMP Works

- Shared Memory Model
- Thread Based Parallelism
- Explicit Parallelism
- Compiler Directive Based
- Dynamic Threads
- I/O(It is your duty!)

OpenMP API Overview

- Compiler Directives
- Runtime Library Routines
- Environment Variables

Compiler Directives

Format

```
#pragma omp directive-name [clause,...]
```

Directive-name

- parallel
  - A parallel region is a block of code that will be executed by multiple threads.
  - It is the fundamental OpenMP parallel construct.
- for
  - Shares iterations of a loop across the team.
  - Data parallelism
- sections
  - Each section is executed by a thread.
  - Functional parallelism
- single
  - The enclosed code is to be executed by only one thread in the team. Other threads will never execute it.
  - Can be used to deal with IO.

Directive-name

- critical
  - Only ONE thread can execute codes in the block at same time.
  - Often used when a shared variable is modified.
- atomic
  - A unit of storage can only be modified by ONE thread at same time.
  - Often used when some value of a shared array is modified.

Clause

Clause	Directive					
	PARALLEL	DO/for	SECTIONS	SINGLE	PARALLEL DO/for	PARALLEL SECTIONS
IF	●				●	●
PRIVATE	●	●	●	●	●	●
SHARED	●	●			●	●
DEFAULT	●				●	●
FIRSTPRIVATE	●	●	●	●	●	●
LASTPRIVATE		●	●		●	●
REDUCTION	●	●	●		●	●
COPYIN	●				●	●
COPYPRIVATE				●		
SCHEDULE		●			●	
ORDERED		●			●	
NOWAIT		●	●	●		

Clause

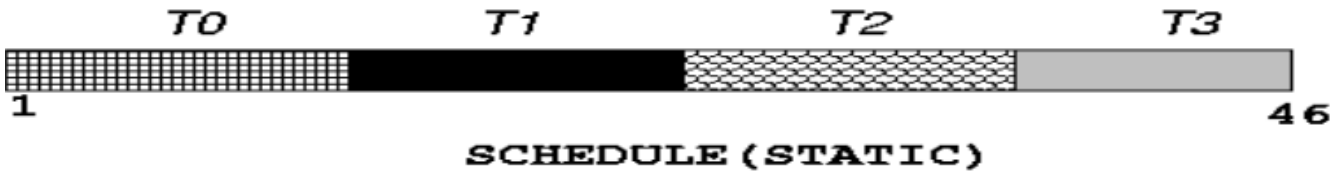
- if (expression)
  - True: A team of threads is created.
  - False: The region is executed serially by the master thread.
- private(list)
  - Variables in the list is private to each thread.
  - They should be assumed to be uninitialized for each thread.(Use firstprivate to solve that.)
- shared(list)
  - Variables in the list is shared among all threads.
  - It is the programmer's responsibility to ensure that multiple threads properly access shared variables!

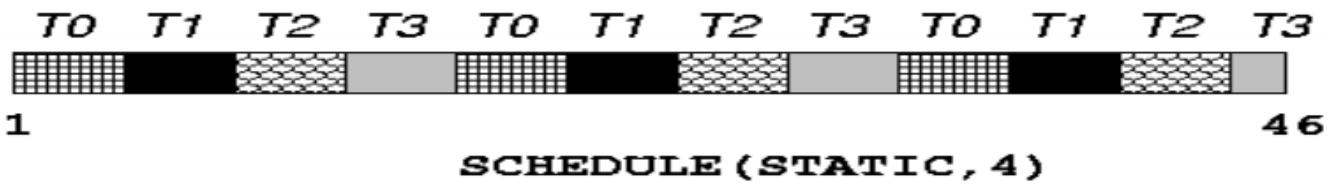
Clause

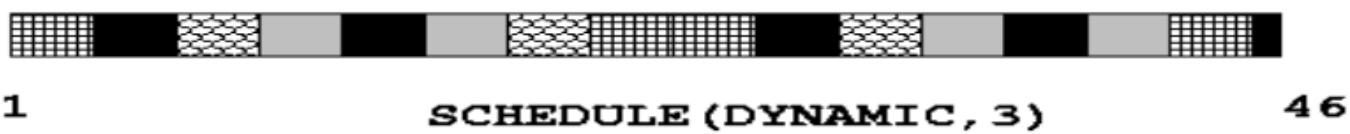
- reduction(op:list)
  - Performs a reduction on the variables that appear in its list.
  - Operator must observe commutative law and associative law.
  - For some variable x in list, multiple private copies of x is created in each thread, and finally x is set as (x op x op x op ... op x).
- schedule(kind[, chunksize])
  - Automatically arrange "working schedule" for different threads of for loop.
  - Chunksize: size of each piece of work.
  - Kind: method to distribute work, including static, dynamic, guided and runtime.

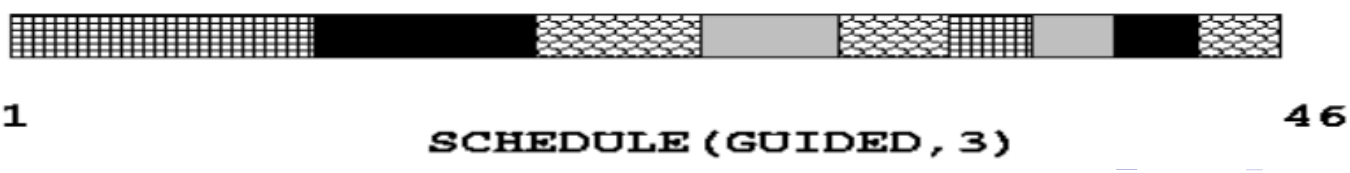
Clause

- schedule(kind[, chunksize])
  - Kind
    - static: loop
    - dynamic: first come, first serve
    - guided: like dynamic, but with descending chunksize
    - runtime: follow the env value OMP_SCHEDULE







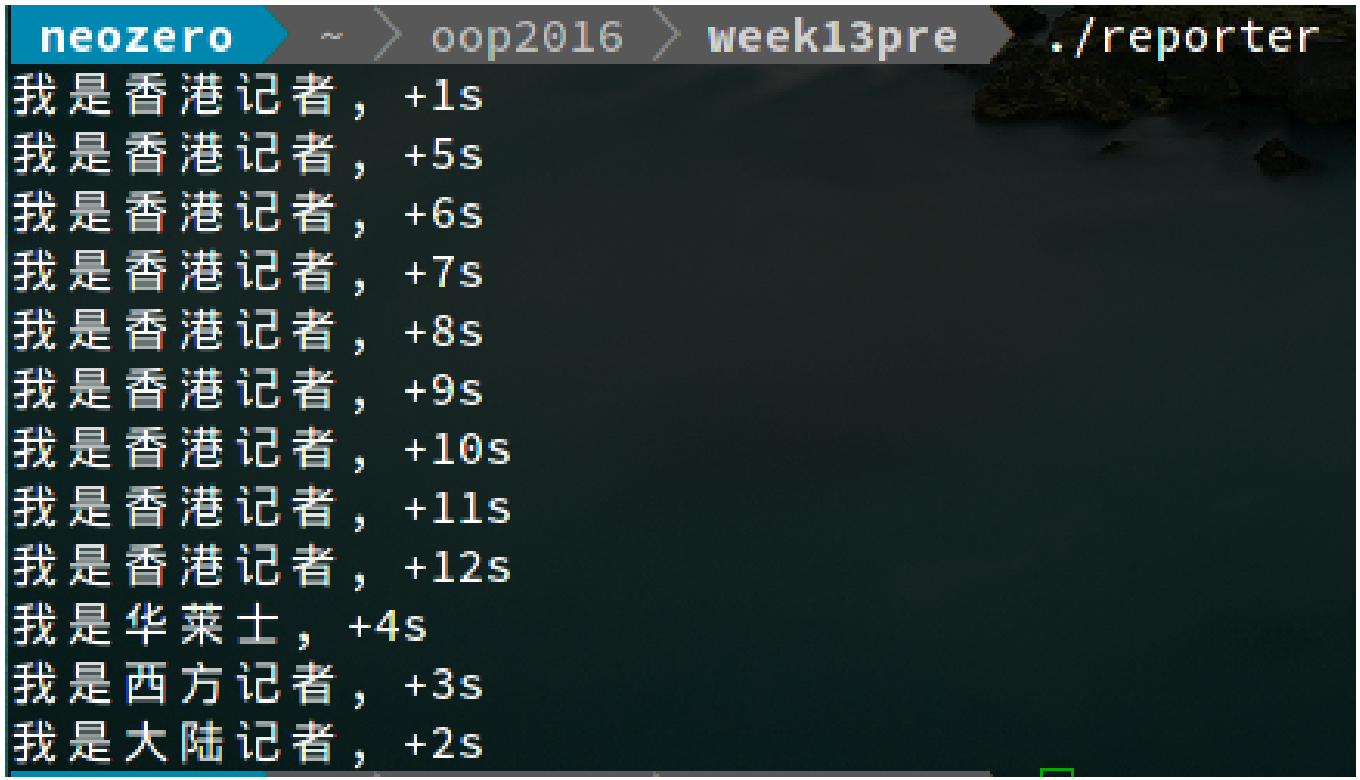


Clause

Static

Clause

Dynamic



Clause

Guided

Runtime Library Routines

- For C/C++, include the <omp.h> header file.
- void omp_set_num_threads(int num_threads)
- int omp_get_num_threads(void)
- int omp_get_thread_num(void)
- Manipulations about lock(using critical may be easier)
- ...

Environment Variables

- OpenMP provides some environment variables for controlling the execution of parallel code.
- OMP_SCHEDULE
- OMP_NUM_THREADS
- OMP_THREAD_LIMIT

Examples

- Merge Sort with OpenMP
- Numeric calculation of π

Merge Sort

```
void mergesort(int* array, int start, int end, int
    thread, int* buffer) {
    if (end - start <= 1)
        return;
    int p = ((start + end) >> 1);
    if (thread <= 1) {
        mergesort(array, start, p, 1, buffer);
        mergesort(array, p, end, 1, buffer);
    }
```

Merge Sort

```
    else {
        #pragma omp parallel sections
        {
        #pragma omp section
        {
        mergesort(array, start, p, thread / 2, buffer);
        }
        #pragma omp section
        {
        mergesort(array, p, end, thread – thread / 2,
            buffer);
        }
        }
    }
    merge(array, start, p, end, buffer);
}
```

Merge Sort


neozero ~ oop2016 week13pre gcc -O3 mergesort_demo.c -o mergesort_demo
-fopenmp
neozero ~ oop2016 week13pre gcc -O3 mergesort_demo.c -o mergesort_demo_
nomp
neozero ~ oop2016 week13pre ./mergesort_demo 100000000 master
Time cost: 8.189 s
neozero ~ oop2016 week13pre ./mergesort_demo_nomp 100000000
Time cost: 14.674 s
neozero ~ oop2016 week13pre master

Numeric calculation of π

$$\int_0^1 \frac{4}{1+x^2} = 4 \arctan 1 = \pi \tag{1}$$

$$\int_0^1 \frac{4}{1+x^2} = \lim_{N \to +\infty} \frac{1}{N} \sum_{i=0}^{N-1} \frac{4}{1+\left(\frac{i}{N}\right)^2} \tag{2}$$

Numeric calculation of π

```
#pragma omp parallel for schedule(static) reduction(+:
    sum)
for (i = 0; i < MAXSTEP; ++i) {
    sum = sum + FUNC((double)i/MAXSTEP);
}
```

```
neozero ~ oop2016 week13pre gcc -O3 pi_calc.c -o pi_calc -fopenmp
neozero ~ oop2016 week13pre gcc -O3 pi_calc.c -o pi_calc_withoutmp
neozero ~ oop2016 week13pre ./pi_calc
Calculated PI is: 3.1415926546
Time: 7.164 s
neozero ~ oop2016 week13pre ./pi_calc_withoutmp
Calculated PI is: 3.1415926546
Time: 13.500 s
neozero ~ oop2016 week13pre
```

Pros & Cons

- Pros
  - Make full use of CPU with little work.
  - Portable multithreading code, not platform-specific.
  - Work can be easily scheduled.
  - Unified code for both serial and parallel applications.
- Cons
  - High chance of accidentally writing false sharing code.
  - Difficult to debug.
  - Not very suitable for some situation, i.e. network programming.
  - Mainly for parallel computing, not for distributed computing.

Reference

- "OpenMP" - Blaise Barney, Lawrence Livermore National Laboratory
  https://computing.llnl.gov/tutorials/openMP/
- OpenMP 并行编程 - 中国科学技术大学超级计算中心
  http://scc.ustc.edu.cn/zlsc/cxyy/200910/W020121113517997951933.pdf
- OpenMP - Wikipedia
  https://en.wikipedia.org/wiki/OpenMP