OpenUH

Sabarinath Mahadevan
sabarinath.mahadevan@rwth-aachen.de

Seminar: Automation, Compilers, and Code-Generation – 11.07.2016

High Performance and Automatic Computing
RWTH Aachen University, Germany

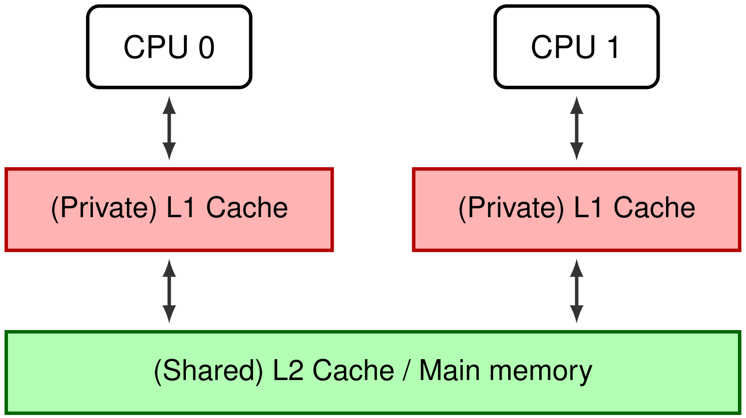
Outline

Introduction

- Modern computing devices are equipped with multiple processors.
- The importance of parallel programming has increased.
- OpenMP is an API that helps programmers to develop a shared memory application.
- Portable and robust compiler for OpenMP required.
  - Increasing popularity of OpenMP.
  - Ever increasing set of target architectures.
  - Academic purposes.
- OpenUH is such a portable, and robust compiler that is based on the Open64 architecture.

Shared Memory Parallel Programming

- Same global, shared memory.
- Parallelism achieved through threads.
- Each thread has its own set of private and shared variables.
- Communication between threads mainly through shared variables.
- Focus on synchronizing access.



Shared memory architecture. Image Credit: Diego Fabregat-Traver

APIs for Shared Memory Parallel Programming

- **POSIX threads (pthreads).**
  Focus on task parallelism.
  Low level.
  Explicit.
  Mainly available on UNIX systems.

- **OpenMP**
  Relatively high level.
  Managed by OpenMP Architecture Review Board (or OpenMP ARB).
  Focus on data parallelism.
  Support for C/C++ and Fortran.

POSIX Threads (pthreads)

- **Threads are usually used to implement parallelism.**
- **In the past, each hardware vendor used to have their own proprietary threads.**
- **Light weight.**
- **Focus on performance.**

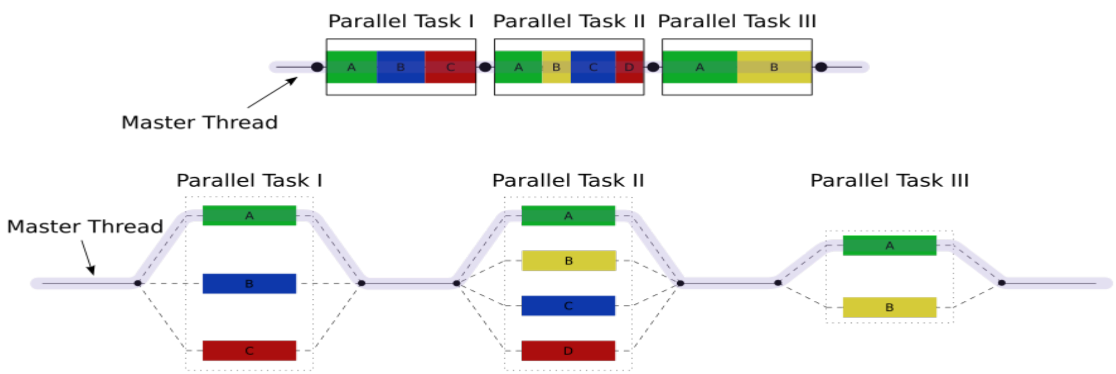
```
void *print_hello_world(void *arg)
{
   printf("Hello World. Greetings from thread %d\n", (int)arg);
   pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
  ...
for(i = 0; i < NUMBER_OF_THREADS; i++) {
  status = pthread_create(&threads[i], NULL,
                          &print_hello_world, (void*)i);
}
}
```

- **Header**
  - #include<pthread.h>
- **Thread Management**
  - pthread_create(…)
  - pthread_exit (…)
  - pthread_join(…)
  - …
- **Mutex Variables**
  - pthread_mutex_init (…)
  - pthread_mutex_lock(…)
  - pthread_mutex_destroy(…)

  …

OpenMP

- Set of compiler directives, library routines, and environment variables.
- Uses fork-join model



Fork-Join paradigm. This illustration is taken from Wikipedia.

- OpenMP directives start with the *#pragma* keyword.
- Code to be executed in parallel is wrapped within *#pragma omp parallel*.
- User may provide additional information on how to run in parallel.
  - #pragma omp parallel num_threads(4)
  - omp_set_schedule(static | dynamic | ...);

Open64: An Overview

- Open64 is an open source, optimizing compiler.
- Uses a common intermediate representation called WHIRL.
- Components.
  Inter-procedural analyzer (IPA), loop-nest optimizer (LNO), global scalar optimizer (WOPT) and code generator (CG).
- WHIRL serves as a common interface.
- Optimisations can be done at a single point.
- Can be easily adapted to any target architecture.

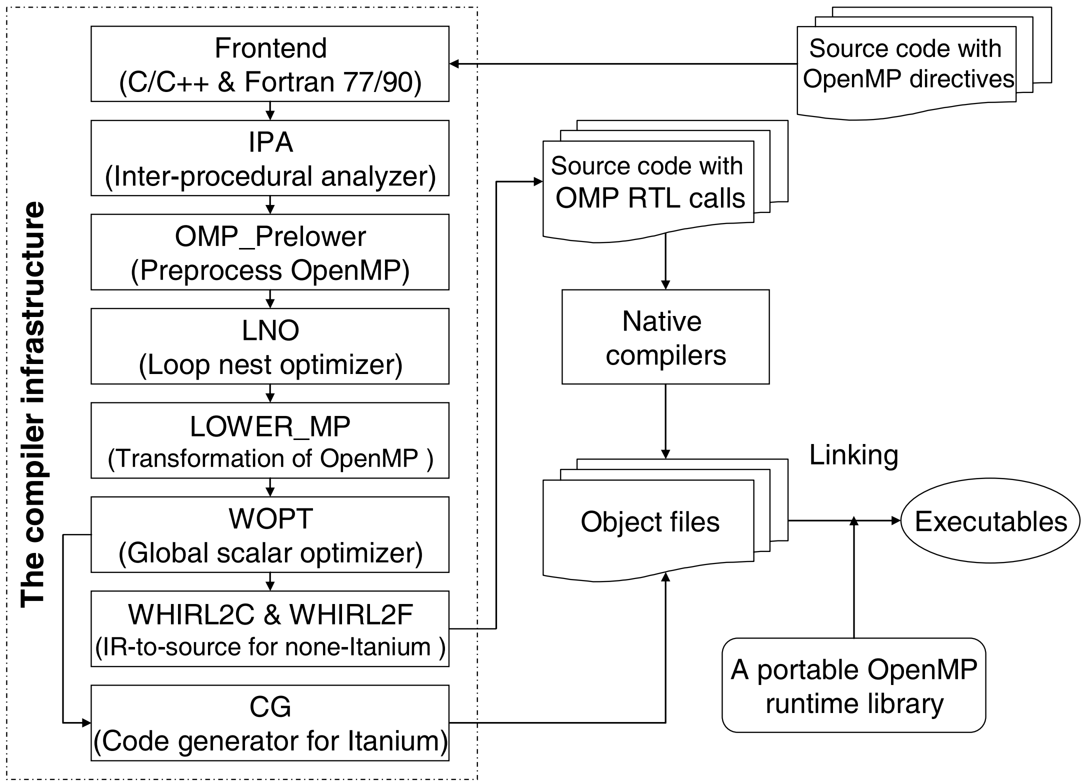
OpenUH: Evolution and Motivation

- Increase in importance for shared memory parallel programming.
- Expanding set of target Architectures.
- Proprietary compilers do not share source.
  For example, Intel, Sun studio ...
- Portable, open source implementation of OpenMP compiler desired.
- Designing such a compiler from scratch was expensive.
- Open64 met the requirements of such a compiler.

OpenUH: Introduction

- Portable, and robust OpenMP compiler.
- Started as a research compiler.
- Hybrid Approach.
  Source to source translator.
  Object code generator.
- Portability is achieved using source to source translator, but at the cost of performance.
- The end to end compiler focusses on optimisation.
- Based on Open64 compiler.

OpenUH: Architecture



Architecture of OpenUH. This illustration is taken from [Chunhua Liao 07]

Translation of Parallel Regions

- Identified by the OpenMP construct *#pragma omp parallel*.
- Other regions are allowed to go through the normal course.
- Required number of threads are spawned by the master thread.
- OpenUH uses pthread api to create threads.
- Code is translated in two stages
  - Pre lowering.
  - Lowering.

Translation of Parallel Regions

- **Pre lowering.**
  - **OpenMP constructs are pre processed.**
  - **Semantic check.**
  - **Translates OpenMP constructs that are not easy to handle.**

Source Code	Code Translated by OpenUH (*whirl2c* representation)

<pre>#pragma omp sections
{
 #pragma omp section
 {
  printf("Section 1");
 }
 #pragma omp section
 {
  printf("Section 2");
 }
}</pre>
	<pre>#pragma omp for private(_w2c_omp_section)
	schedule(interleave, 1U)
	    for(_w2c_omp_section = 0;
	        _w2c_omp_section <= 1;
	        _w2c_omp_section =
	         _w2c_omp_section + 1)
	    {
	      switch((long long)
	             (_w2c_omp_section))
	      {
	      case 0LL :
	        goto _258;
	      case 1LL :
	        goto _514;
	      }
	      _258 :;
	      printf("Section 1");
	      goto _770;
	      _514 :;
	      printf("Section 2");
	      _770 :;
	    }</pre>

OpenUH translation of the *sections* construct.

Translation of Parallel Regions

- **Lowering.**
  - **Translation of parallel regions into pthreads.**
  - **Usually, a process called outlining is used.**
  - **OpenUH uses a different process called inlining.**
- **Parallel regions are extracted into methods**
- **These methods are called micro tasks.**
- **The extracted methods are *inlined* in OpenUH, and hence global variables are shared.**

<table>
<tr><th>Original OpenMP Code</th><th>Outlined Translation</th></tr>
<tr><td>int main(void)
{
int a,b,c;

#pragma omp parallel private(c)
do_sth(a,b,c);

return 0;
}</td>
<td rowspan="3">/*Outlined function with an extra argument for passing addresses*/
static void __ompc_func_0(void **__ompc_args){
int *_pp_b, *_pp_a, _p_c;

/*dereference addresses to get shared variables */
_pp_b=(int *)(*__ompc_args);
_pp_a=(int *)(*(__ompc_args+1));

/*substitute accesses for all variables*/
do_sth(*_pp_a,*_pp_b,_p_c);
}

int _ompc_main(void){
int a,b,c;
void *__ompc_argv[2];

/*wrap addresses of shared variables*/
*(__ompc_argv)=(void *)(&b);
*(__ompc_argv+1)=(void *)(&a);
…
/*OpenMP runtime call has to pass the addresses of shared variables*/
_ompc_do_parallel(__ompc_func_0,
__ompc_argv);
…
}</td></tr>
<tr><th>Inlined (Nested) Translation</th></tr>
<tr><td>_INT32 main()
{
int a,b,c;

/*inlined (nested) microtask */
void __ompregion_main1()
{
_INT32 __mplocal_c;

/*shared variables are keep intact, only substitute the access to private variable*/
do_sth(a, b, __mplocal_c);
}
…
/*OpenMP runtime call */
__ompc_fork(&__ompregion_main1);
…
}</td></tr>
</table>

Inlining v/s Outlining. This illustration is taken from [Chunhua Liao 07]

Translation of Parallel Regions

- The *__ompc_fork* library routine is responsible for creation of pthreads.
- It creates the required number of slaves to execute the micro task.

```
/* The main fork API. at the first fork, initialize the RTL*/
void
__ompc_fork(const int _num_threads, omp_micro micro_task,
            frame_pointer_t frame_pointer)
{
 ....
 for (i=0; i<__omp_level_1_team_size; i++) {
      __omp_level_1_team[i].frame_pointer = frame_pointer;
      __omp_level_1_team[i].team_size = __omp_level_1_team_size;
      __omp_level_1_team[i].entry_func = micro_task;
   }
 ...

 for (i=__omp_level_1_team_alloc_size; i<new_num_threads; i++) {
     //Some initialisations
     ....
     return_value = pthread_create( &(__omp_level_1_pthread[i].uthread_id),
                                    &__omp_pthread_attr, (pthread_entry) __ompc_level_1_slave,
                                    (void *)((unsigned long int)i));

     __omp_level_1_pthread[i].stack_pointer = (char *)0;
     ...


   }
    ...
    //Master thread executes the slaves.
    __omp_level_1_pthread[0].task = &(__omp_level_1_team[0]);

    __omp_current_task = __omp_level_1_team[0].implicit_task;

    __ompc_set_state(THR_WORK_STATE);
    micro_task(0, frame_pointer);

    __ompc_level_1_barrier(0);
   ...
}
```

Open MP fork method [Pseudo code taken from the compiler].

Translation of Data Constructs

- **OpenMP data constructs.**
  private, firstprivate, lastprivate, shared, threadprivate ...
- **Shared variables are passed as reference in the outlining process, but are shared by default in OpenUH inlining.**
- **Private variables are declared within the function.**
- **For firstprivate, the local copy is initialised first.**
- **For lastprivate, code is added at the end of parallel region to calculate the final value of the variable.**

C Code	OpenMP translation

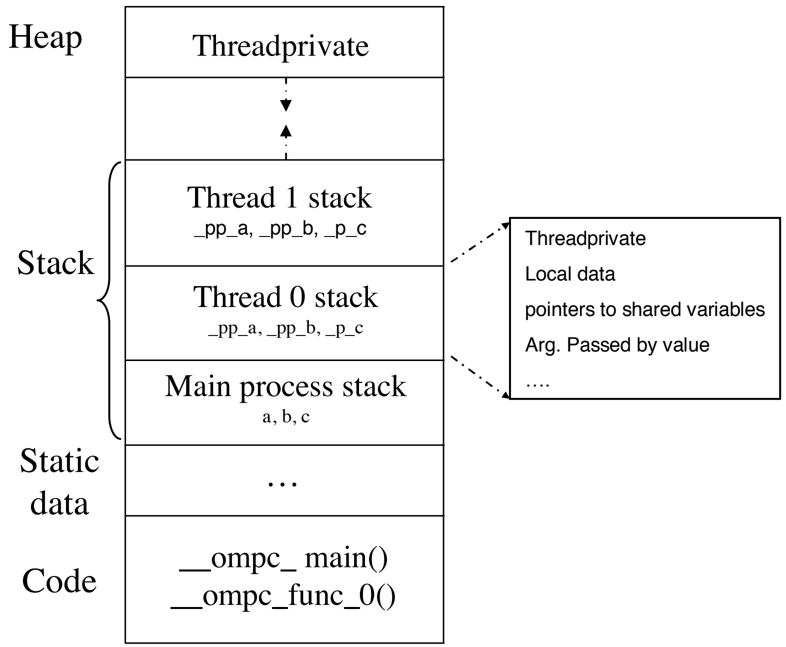
int main(void)
{
  int A = 1, B = 1, C = 1;
 	
#pragma omp parallel num_threads(2) \\
    default(none) shared(A) private(B) firstprivate(C)
  {
    int i;
    #pragma omp for
    for(i=0; i<20; i++)
    {
      C = C + i;
    }
  }
  return 0;
}
	int main(void)
	{
	  /* inlined microtask generated from parallel region */
	  void __ompregion_main1(…)
	  {
	  /* get current thread id */
	  __ompv_gtid_s = __ompc_get_thread_num();
	  _mplocal_B;
	  _mplocal_C = C
	  ………..
	  /* execute loop body using assigned iteration space */
	  for(__mplocal_i = __ompv _do_lower; (__mplocal_i <= __ompv _do_upper); __mplocal_i = (__mplocal_i + 1))
	  { …
	  }
	  /* Implicit BARRIER after work sharing constructs */
	  __ompc_barrier();
	  return;
	  }
	
	  /* Implement multithreaded model */
	  __ompv_in_parallel = __ompc_in_parallel();
	  __ompv_ok_to_fork = __ompc_can_fork();
	  if(((__ompv_in_parallel== 0) && (__ompv_ok_to_fork == 1)))
	  {
	  /* Parallel version: a runtime library call for creating multiple threads and executing the microtask in parallel */
	  __ompc_fork(&__ompregion_main1,…);
	  } else
	  { /* Sequential version */
	  ….
	  return;
	  }
	}

An example that illustrates the OpenMP translation of the *omp for* work sharing construct.

Runtime Library

- Implements OpenMP routines.
- Manipulates the underlying threads.
- Uses Pthreads to create parallel threads.
- Master thread spawns the required number of threads.
- Threads are put to sleep at the end of parallel regions.
- Each thread maintains its private stack.
- Variables that are declared as *threadprivate* are stored in heap with an array of references.

Runtime Library



Memory Allocation. This illustration is taken from [Barbara Chapman 08]

Summary

- OpenMP has become the de facto standard for shared memory parallel programming.
- OpenUH is a portable, and robust OpenMP compiler.
- It uses a hybrid model.
  Source to source translator for portability.
  Object code generator for performance.
- Based on Open64 architecture.
- Front end for C/C++ and Fortran.

Thank you for your attention

Sabarinath Mahadevan

`sabarinath.mahadevan@rwth-aachen.de`

`http://hpac.rwth-aachen.de/teaching/sem-accg-16/`

References

[Barbara Chapman 08] R.v.d.P. Barbara Chapman, Gabriele Jost: *Using OpenMP:Portable Shared Memory Parallel Programming*. The MIT Press, Sept. 2008. 20

[Chunhua Liao 07] B.C.W.C.W.Z. Chunhua Liao, Oscar Hernandez: OpenUH: an optimizing, portable OpenMP compiler. *Wiley InterScience*, Vol., 2007. 11, 15