Outline

Introduction
Shared Memory Parallel Programming
APIs for Shared Memory Parallel Programming
   POSIX Threads (pthreads)
   OpenMP
Open64: An Overview
OpenUH: Evolution and Motivation
OpenUH: Architecture
Translation of Parallel Regions
Translation of Data Constructs
Runtime Library
Summary
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.

```c
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(...)
  - ...

```c
• 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

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]

IPA
(Inter-procedural analyzer)

OMP_Prelower
(Preprocess OpenMP)

LNO
(Loop nest optimizer)

LOWER_MP
(Transformation of OpenMP)

WOPT
(Global scalar optimizer)

WHIRL2C & WHIRL2F
(IR-to-source for none-Itanium)

CG
(Code generator for Itanium)

Frontend
(C/C++ & Fortran 77/90)

Source code with
OpenMP directives

Source code with
OMP RTL calls

Native compilers

Object files

Linking

Executables

A portable OpenMP runtime library
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.

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Code Translated by OpenUH (whirl2c representation)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#pragma omp sections</code></td>
<td><code>#pragma omp for private(_w2c_omp_section)</code></td>
</tr>
<tr>
<td>`{</td>
<td>schedule(interleave, 1U)</td>
</tr>
<tr>
<td>#pragma omp section</td>
<td>for(_w2c_omp_section = 0;</td>
</tr>
<tr>
<td>{</td>
<td>_w2c_omp_section &lt;= 1;</td>
</tr>
<tr>
<td>printf(&quot;Section 1&quot;);</td>
<td>_w2c_omp_section =</td>
</tr>
<tr>
<td>}</td>
<td>_w2c_omp_section + 1)</td>
</tr>
<tr>
<td>#pragma omp section</td>
<td>{</td>
</tr>
<tr>
<td>{</td>
<td>switch((long long)</td>
</tr>
<tr>
<td>printf(&quot;Section 2&quot;);</td>
<td>(_w2c_omp_section))</td>
</tr>
<tr>
<td>}</td>
<td>{</td>
</tr>
<tr>
<td>case 0LL :</td>
<td>goto _258;</td>
</tr>
<tr>
<td>case 1LL :</td>
<td>goto _514;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>_258 ::;</td>
<td></td>
</tr>
<tr>
<td>printf(&quot;Section 1&quot;);</td>
<td>goto _770;</td>
</tr>
<tr>
<td>_514 ::;</td>
<td></td>
</tr>
<tr>
<td>printf(&quot;Section 2&quot;);</td>
<td>_770 ::;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

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.
### Original OpenMP Code

```c
int main(void)
{
    int a,b,c;
    #pragma omp parallel private(c)
    do_sth(a,b,c);
    return 0;
}
```

### Outlined Translation

```c
/*Outlined function with an extra argument for passing addresses*/
static void __ompc_func_0(void **__ompc_args){
    int *__pp_b, *__pp_a, __p_c;

    /*derefence addresses to get shared variables */
    __pp_b=(int *)(*__ompc_args);
    __pp_a=(int *)(*(__ompc_args+1));
}

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);
    ...
}
```

### Inlined (Nested) Translation

```c
_INTC32 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);
    ...
}
```

### Inlining v/s Outlining

In this illustration, it 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.

```c
/* 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_level_1_team[0].implicit_task = __omp_level_1_team[0].implicit_task;
    __omp_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.
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.
Figure 8.22: Storing an OpenMP program’s objects in memory

– Each thread has its own stack for storing its private data, including local data in procedures invoked from within parallel regions. We have indicated two places in which threadprivate data might be saved. The actual location is implementation-dependent. We indicate where objects from the program shown in Figure 8.11 will be stored.

parallel region. Inside the parallel region, memory will be dynamically allocated on the heap to store the integers pointed to by this array. Each thread will retrieve its own threadprivate variable via the corresponding pointer and its threadid. The address it computes is used thereafter in the code. In Figure 8.24, we give an excerpt from the translated code fragment. Recall that this construct requires that the number of threads used to execute parallel regions does not change for the duration of the program.

8.3.9 Do Idle Threads Sleep?

When threads are not working, perhaps because a parallel region has terminated or because there is an imbalance in the workload between different threads, they may need to wait. The waiting may be realized in two different ways: idle threads may busy-wait (or spin-wait) or they may be put to sleep (or be suspended). In the former case, the threads actively continue to check for new work. They will begin this work without delay when it is available. Unfortunately, however, busy-waiting threads also consume resources and may interfere with the work of other threads.

The alternative strategy is to put threads to sleep. This removes the problem of conflicting with the work of other threads. The alternative strategy is to put threads to sleep. This removes the problem of interfering with the work of other threads.
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

