ADAPTIVE TASK SCHEDULING USING LOW-LEVEL RUNTIME APIs AND MACHINE LEARNING

Keynote, ADVCOMP 2017

November, 2017, Barcelona, Spain

Prepared by: Ahmad Qawasmeh
Assistant Professor
The Hashemite University, Jordan
Outline

1. Motivation and Background

2. Related Work

3. APARF Framework Implementation (OpenUH)
   - OpenMP tasking profiling APIs
   - OpenMP profiling tool and performance analysis
   - A hybrid machine learning model for adaptive prediction

4. Analysis and Evaluation

5. Summary and Future work
• Predicting the optimum task scheduling scheme for a given OpenMP program by developing Adaptive and portable framework
Main Contributions

A. I proposed a new open-source API for OpenMP task profiling in OpenUH RTL.

B. I developed a reliable OpenMP profiling tool for capturing useful low-level runtime performance measurements.

C. I exploited my performance framework to perform a comprehensive scheduling analysis study.

D. I built and evaluated a portable framework (APARF) for predicting the optimal task scheduling scheme that should be applied to new, unseen applications.
Background
Shared Memory: Logical View

Shared memory space

proc1  proc2  proc3  procN

SMP Vs cc-NUMA
A standard **API** to write parallel shared memory applications in C, C++, and Fortran

Consists of **compiler directives, runtime routines, environment variables**

Master thread

Worker thread

Barrier

Parallel Regions

A Nested Parallel region

http://www.openmp.org
A task is an asynchronous work unit

- C/C++: `#pragma omp task`
- Fortran: `!$omp task`

Contains a task region and its data environment

```c
int fib(int n) {
    int x, y;
    if (n < 2) return n;
    else {
        #pragma omp task shared(x)
        x = fib(n-1);
        #pragma omp task shared(y)
        y = fib(n-2);
        #pragma omp taskwait
        return x + y;
    }
}
```
OpenMP Task Scheduling

- Global Shared Pool
- Shared/Distributed Pool
- Hierarchical Pool
- Distributed Queues with Work Stealing
Performance Observation

Profiling vs. Tracing
OpenMP performance APIs before OMPT

- **POMP (Profiler for OpenMP)**
  - Instrumentation calls inserted by a source-source tool (TAU, KOJAK, Scalasca)
  - Can notably affect compiler optimizations

- **ORA (Collector API)**
  - Sampling of call stack
  - Originally has 11 mutually exclusive states, 9 requests, and 22 defined callback event
  - Was accepted as a white paper by ARB
  - Introduced before tasks and implemented in OpenUH RTL

- **OMPT (OpenMP Tool Interface)**
Related Work
An OpenMP scheduler was proposed to adapt the granularity of work within loops depending on data placement info.

Some previous works have focused on disabling threads in parallel loops in the presence of contention.

A thread scheduling policy embedded in a GOMP-based framework was proposed for OpenMP programs featuring irregular parallelism.

Another area of research aims to reduce scheduling overhead by increasing task granularity by chunking a parallel loop or by using a cut-off technique.
Characterization using Machine Learning

- Machine learning was used to characterize programs in representative groups
Automatic Portable and Adaptive Runtime Feedback-Driven (APARF) Framework
Task Execution Model in OpenUH

- **READY**
  - __ompc_task_switch: execute task removed from pool
  - __ompc_task_wait

- **RUNNING**
  - __ompc_task_create: adds child tasks to the task pool
  - __ompc_task_wait

- **WAITING**
  - __ompc_remove_task_from_pool: removes a task from the pool, and switches to it

- **EXITING**
  - __ompc_remove_task_from_pool: removes a task from the pool, and switches to it

- num_children == 0?
  - (other_task) __ompc_task_exit
  - num Blocking Children == 0?
  - task destroyed

http://web.cs.uh.edu/~openuh/
proposed a tasking profiling interface in the OpenUH RTL as an extension to the ORA before OMPT

- Task creation
- Task execution
- Task completion
- Task switching
- Task suspension

OMPT is a super-set of ORA

- Support sampling of call stack with optional trace event generation.
- State support, task creation and completion are mandatory, while the others are optional

Adapting my tasking APIs to be compatible with OMPT was straightforward
Overhead Analysis in OpenUH RTL

FFT

Health

UTS

Alignment

SparseLU

Fibonacci

Floorplan

NQueens

Strassen
Adaptive Scheduling Through APARF

**Queue Organization**: Global, Distributed, Hybrid, Hierarchical

**Task placement and removal**: LIFO, FIFO, Deque, INV-Deque

**Chunk size**, NUM-SLOTS, STORAGE

**Compute/memory bound**

**Granularity (fine-coarse grain)**

#tasks directives

**Nested tasks and structure**

---

**Scheduling scheme**

**OpenMP Program**

**OMPT Tasking API in OpenUH RTL**

**Profiling Tool**

**Hybrid Machine Learning Model**

- **APARF FRAMEWORK**
  - Data-Preprocessing
  - Clustering
  - Classification

**Performance system**

- **Adaptive Feedback**

- **Class 1**
- **Class 2**
- **Class 3**
Interaction Example in APARF

- Ahmad Qawasmeh, Abid Malik, Barbara Chapman, Kevin Huck, Allen Malony, "Open Source Task Profiling by Extending the OpenMP Runtime API", IWOMP2013, pp. 186-199, September 2013, Canberra, Australia.
APARF OpenMP Profiling Tool

- Implements a single handler to handle all events.
- Initializes the API to establish a connection with the runtime.
- Captures useful low-level runtime performance measurements.
- Timing, HWCs, and Energy/power sensors were integrated.

```c
int fib(int n) {
    int x, y;
    if (n < 2)  return n;
    else {
        #pragma omp task shared(x)
        x = fib(n-1);
        #pragma omp task shared(y)
        y = fib(n-2);
        #pragma omp taskwait
        return x + y;
    }
}
```
An OpenMP task scheduler can be distinguished based on:

- Queue organization
- Work-stealing capability.
- Order in which a task graph is traversed

Two crucial issues should be managed by a task scheduler:

- Data locality
- Load balancing

Conflicting Goals:

- Queue contention, work stealing, synchronization overheads
- Task granularity (coarse vs. fine)
We performed a detailed analysis study

- 200 scheduling schemes were applied to eight BOTS benchmarks
- Three different sets of threads were used with two input sizes
- Initial observation: categorized into three representative groups

<table>
<thead>
<tr>
<th>Platform</th>
<th>Facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD cluster</td>
<td>an x86-64 cc-NUMA Linux system with a four 2.2 GHz 12-core AMD Opteron processors (48 cores total) and 512 KB L2 cache per core, and 10 MB L3 cache shared by all cores</td>
</tr>
<tr>
<td>Intel cluster</td>
<td>an x86-64 cc-NUMA Linux system with two 2.5 GHz 12-core Intel Xeon processors (24 cores total) and 512 KB L2 cache per core, and 15 MB L3 cache shared by all cores</td>
</tr>
</tbody>
</table>
We have used our performance framework

- The captured runtime events are: task suspension, task execution, task completion, task creation, explicit/implicit barrier, parallel-region, and single/master/loop region

- Exploiting data locality can best be expressed by demonstrating the cache behavior (cache misses, CPI, TLB)

- Maintaining load balancing was evaluated by obtaining the timing distribution among threads for each captured event.

Similarity Among Benchmarks
Similarity Among Benchmarks

Alignment (Time)

Alignment (L2 Miss Rate)

SparseLU (Time)

SparseLU (L2 Miss Rate)
Hybrid Machine Learning Modeling

Why machine learning?
- Measurements obtained from the runtime by external tool regardless of the used runtime or compiler
- 384 data instances with 14 selected features (Overwhelming for human processing)

Meaning of hybrid in our context?
- Unsupervised learning (K-Means clustering)
- Supervised learning

Major challenges?
- Complex search space
- Limited # task-based programs for training
- Features selection

Java tool based on the weka API
Classification Process for Prediction

<table>
<thead>
<tr>
<th>Rank</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.519</td>
<td>c_suspend</td>
</tr>
<tr>
<td>1.18</td>
<td>c_execution</td>
</tr>
<tr>
<td>1.169</td>
<td>c_parallel</td>
</tr>
<tr>
<td>0.987</td>
<td>t_finish</td>
</tr>
<tr>
<td>0.919</td>
<td>t_creation</td>
</tr>
<tr>
<td>0.895</td>
<td>t_suspend</td>
</tr>
<tr>
<td>0.841</td>
<td>c_single</td>
</tr>
<tr>
<td>0.83</td>
<td>c_creation</td>
</tr>
<tr>
<td>0.793</td>
<td>t_execution</td>
</tr>
<tr>
<td>0.678</td>
<td>c_finish</td>
</tr>
<tr>
<td>0.66</td>
<td>c_barrier</td>
</tr>
<tr>
<td>0.5</td>
<td>t_parallel</td>
</tr>
<tr>
<td>0.489</td>
<td>t_barrier</td>
</tr>
<tr>
<td>0.488</td>
<td>t_single</td>
</tr>
</tbody>
</table>
# Training Data Improvement

<table>
<thead>
<tr>
<th>Actual class</th>
<th>Predicted class</th>
<th>Improvement (AMD)</th>
<th>Improvement (Intel)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>simple</td>
<td>public</td>
<td>default</td>
</tr>
<tr>
<td>Fib</td>
<td>26%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Health</td>
<td>30%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sort</td>
<td>21%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FFT</td>
<td>10%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Nqueens</td>
<td>9%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Strassen</td>
<td>8%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Alignment</td>
<td>3%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sparse</td>
<td>4%</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Training Data Behavior

Normalized data

Runtime Event
Portable Prediction Behavior

93% prediction accuracy

<table>
<thead>
<tr>
<th>Program</th>
<th>Predicted Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTS</td>
<td>24 public(1)</td>
</tr>
<tr>
<td>Floorplan</td>
<td>16 simple(0), 8 default(2)</td>
</tr>
<tr>
<td>EPCC</td>
<td>24 public(1)</td>
</tr>
<tr>
<td>Whetstone</td>
<td>24 simple(0)</td>
</tr>
<tr>
<td>MD</td>
<td>24 simple(0)</td>
</tr>
</tbody>
</table>

MD Application

UTS Benchmark

31
Summary/Future Work
Summary and Future Work

A. I proposed a new open-source API for OpenMP task profiling in OpenUH.

B. I developed a reliable OpenMP profiling tool for capturing useful low-level runtime performance measurements.

C. I used my performance framework to perform a comprehensive scheduling analysis study.

D. I built and evaluated a portable framework (APARF) for predicting the optimal task scheduling scheme that should be applied to new, unseen applications.

>>> Predict energy consumption behavior at the fine-grain level
2017


2015

- [c5] Ahmad Qawasmeh, Barbara M. Chapman, Maxime R. Hugues, Henri Calandra: *GPU technology applied to reverse time migration and seismic modeling via OpenACC*. PMAM@PPoPP 2015: 75-85

2014

- [c3] Anilkumar Nandamuri, Abid Muslim Malik, Ahmad Qawasmeh, Barbara M. Chapman: *Power and energy footprint of openMP programs using OpenMP runtime API*. E2SC@SC 2014: 79-88

2013


2012

- [c1] Ahmad Qawasmeh, Barbara M. Chapman, Amrita Banerjee: *A Compiler-Based Tool for Array Analysis in HPC Applications*. ICPP Workshops 2012: 454-463
Acknowledgement

- HPCTools Group at the University of Houston, Texas, USA
Thank You!