PolyTasking: Exploration of Code Structure Similarities in Program Parallelization and Optimization

Guang Gao
Daniel Orozco

University of Delaware
http://www.udel.edu

Computer Architecture and Parallel Systems Laboratory
http://www.capsl.udel.edu
Overview

Introduction to the PetaApps NSF project.

Motivation

Tasks and HPC programs.

Similarity of Tasks

The problem with execution.

Polytasks: A way to optimize execution of tasks.

Is it relevant?

Results

Conclusions
Introduction: The NSF Project

Collaborative Research: PetaApps: Enabling Multiscale Modeling of Turbulent Clouds on Petascale Computers

University of Delaware
Guang R. Gao, Distinguished Professor of Electrical and Computer Engineering
Chandra Kambhamettu, Professor of Computer and Information Sciences
Xiaoming Li, Assistant Professor of Electrical and Computer Engineering
Louis Rossi, Associate Professor of Mathematical Sciences
Lian-Ping Wang (PI), Professor of Mechanical Engineering

National Center for Atmospheric Research
Wojciech W. Grabowski (PI), Senior Scientist,
Mesoscale and Microscale Meteorology Division
Andrzej A. Wyszogrodzki, Scientist, Research Application Laboratory

Project website
http://cloud-physics.udel.edu/

2011/8/15  Gao and Orozco -- Polytasking  3
A multiscale problem

….. that requires petascale computing.
(\(10^{15}\) floating point operations per second)

NSF Cyberinfrastructure Framework for 21\textsuperscript{st} Century Science and Engineering (CF21): \textbf{Computation is accepted as the third pillar supporting innovation and discovery in science and engineering.}
Two Basic Functions of Modern OS

- **Function 1 (service API):** Extending the Machine (or virtual machine)
  
  Purpose: Make the machine *easier* to program (e.g. through *system calls*)

- **Function 2 (host):** Managing the Resources
  
  Purpose: Provide an *orderly and controlled* allocation of resources to various programs/jobs competing for them.

---

Functions Do Not Belong To A Classical OS?

- In sequential processors/cores, the OS does not do (or interfere with):
  - Instruction scheduling
  - Register allocation
  - Branch prediction
  - Control speculation
  - Etc ...

- But Why?
How About OS in Many-Core Era?
Questions?

- Should OS directly manage user threads?
- Should OS directly manage inter-thread synchronization/communication?
- Should OS dictates shared memory semantics of a multi-thread programs? (consistency model, etc.)
- Should OS ...
The Key to Fill the Gap Is Parallel Program Execution Model!

2011/8/15 Gao and Orozco -- Polytasking
Overview

Introduction to the PetaApps NSF project.

Motivation

Tasks and HPC programs.

Similarity of Tasks

The problem with execution.

Polytasks: A way to optimize execution of tasks.

Is it relevant?

Results

Conclusions
A Summary of Techniques

Programming Model
- Graph Representation
- Parallel Loop Support
- Dependence-Based Model
- Composability

Task Scheduling
- Priorities
- Parallel Loop Support
- High Throughput
- Distributed Scheduling
- Non Preemptive Model

Synchronization
- Distributed Algorithms
- Fine Grained Synchronization
- Pipeline Synchronization

Processor Architecture
- Many Core Architectures
- Atomic Operations
- Synchronization Busses
- Resource Limitations
- Energy

Program Execution Model
- DFM2011
- LCPC2011
- ICPP2009
- POHLL2007
- ETC2011
- EuroPar2011
- MultiProg2011
New challenges for Execution Models
HPC Programs in Many-Core Architectures

HPC programs are expressed as tasks
– OpenMP, Cilk, Habanero, TIDeFlow

Example:

```
#pragma omp parallel for
for ( i = 0; i < N; i++ )
{
    f( i );
}
```
Tasks are similar...

During execution, tasks that come from the same parallel loop are very similar:

– They execute the same function
– They use the same data
– Their parameters only differ in the loop iteration

```c
#pragma omp parallel for
for ( i = 0; i < N; i++ )
{
    f( i );
}
...

#pragma omp parallel for
for ( i = 0; i < M; i++ )
{
    g( i );
}
...

#pragma omp parallel for
for ( i = 0; i < L; i++ )
{
    h( i );
}
```
Execution of Tasks

Processors create (spawn) and consume (execute) tasks from the work pool.

Usually, the work pool is implemented as a (distributed) queue.
The Problem

Does task management (creation, assignment, others) require a significant amount of time?

Is task management the bottleneck of the application?

Answer:

– It depends on the granularity of the program
– It depends on the number of processors

<table>
<thead>
<tr>
<th>Process</th>
<th>Cycles</th>
<th>Process</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enqueue one task</td>
<td>6200</td>
<td>Enqueue a task for each processor</td>
<td>$9.9 \times 10^5$</td>
</tr>
<tr>
<td>Execute a tile of size 1</td>
<td>180</td>
<td>Execute a tile of size 256</td>
<td>16000</td>
</tr>
<tr>
<td>Execute a tile of size 1024</td>
<td>56000</td>
<td>Execute a tile of size 16384</td>
<td>$9.0 \times 10^5$</td>
</tr>
<tr>
<td>Execute a tile of size 65536</td>
<td>$3.6 \times 10^6$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Opportunity

Depending on the number of processors and granularity, task management may be a significant source of overhead.

The problem comes from the necessity to put and get the tasks in the work pool.

But, tasks are almost identical, is it possible to have a compressed representation?
The Technique

Writing N tasks requires N units of time: A slow approach.

Instead, they can be represented as a single polytask.
How to do it

<table>
<thead>
<tr>
<th>Task Info</th>
<th>PolyTask Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Function</td>
<td></td>
</tr>
<tr>
<td>- Parameters</td>
<td></td>
</tr>
<tr>
<td>- Environment info</td>
<td></td>
</tr>
<tr>
<td>(Data, etc)</td>
<td></td>
</tr>
<tr>
<td>- Consumers/Return Info</td>
<td></td>
</tr>
</tbody>
</table>

Two counters are all that is required to support polytasks.

Increment/Decrement operations are used to read individual tasks.
Write N, Read 1

Write N
Creates 1 polytask

Work Pool

High Performance Queue

Read 1

Work Pool

High Performance Queue

2011/8/15
Is this relevant to me?

This figure shows the effect of the granularity of the program and the runtime used.

Conclusions:
1. The use of polytasks is very important for fine granularity.
2. The choice of runtime does not play a significant role.
Results

Effect of Polytasks on Programs

<table>
<thead>
<tr>
<th></th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FDTD1D</td>
</tr>
<tr>
<td>MS Queue</td>
<td>1.92</td>
</tr>
<tr>
<td>MC Queue</td>
<td>8.12</td>
</tr>
<tr>
<td>SpinQueue</td>
<td>510.9</td>
</tr>
</tbody>
</table>

2011/8/15
Conclusions

HPC programs have low information entropy at the runtime system level.

The low information entropy can be exploited to compress redundant information.

The user gains in execution efficiency, but only when:

– The program is a fine-grained program
– The program runs in a system with many processors.