The Case for Tasking

CS315B
Lecture 16

What is Tasking?

- Dependence graphs

- Operations
  - Nodes

- Ordering dependencies between operations
  - Edges
  - May or may not imply data movement
Dependence Graphs

The Case for Tasking

- Tasking is not good or bad, except in comparison to alternatives
- The competition is message passing
Tasking vs. Message Passing

- What are the advantages and disadvantages of tasking and message passing, relative to each other?

- Four areas
  - Performance
  - Portability
  - Programmability
  - System Complexity
How Much Does the System Know?

- One way to frame the differences is in how much the programming system knows about the programs being run

- Affects ability to
  - Do optimizations (performance)
  - Automatically retarget programs (portability)
  - Let the programmer say less (programmability)

What Does Tasking Know?
What Does Tasking Know?

What Does Message Passing Know?
What Does Message Passing Know?

What Do They Know?

• Tasking
  - Knowledge of future operations
    • Of multiple kinds
  - And their dependences

• Message Passing
  - A message needs to be sent now
  - A collective communication needs to be done
    • Usually now
What Does Legion Know?

- What tasking knows
  - What data a task needs
  - Dependences between tasks

- Refined by the structure of program data
  - What subset of the data does a task need?
  - More accurate computation of dependences
    - Because of knowledge of partitioning & field use

Performance Advantages

<table>
<thead>
<tr>
<th>Tasking</th>
<th>Message Passing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability to map and schedule</td>
<td></td>
</tr>
<tr>
<td>Do work ahead of time</td>
<td></td>
</tr>
</tbody>
</table>
  - Hide latency |
**Mapping**

- \( t_1 \)
- \( t_2 \)
- \( t_3 \)
- \( t_4 \)
- \( t_5 \)

**S3D: Combustion Simulation**

- Simulates chemical reactions

- Two parts
  - Physics
    - Nearest neighbor communication
    - Data parallel
  - Chemistry
    - Local
    - Complex task parallelism
    - Large working sets/task

Recent 3D DNS of auto-ignition with 30-species DME chemistry (Bansal et al. 2011)
Mapping for Heptane 48^3

- Dynamic Analysis for (rhsf+2)
- Clean-up/meta tasks

4 AMD Interlagos
Integer cores
Legion Runtime

8 AMD Interlagos FP
cores for application

NVIDIA Kepler K20

Mapping for Heptane 96^3

- Handle larger problem sizes per node
  - Higher computation-to-communication ratios
  - More power efficient
- Different mapping
  - Limited by size of GPU framebuffer
- Legion analysis is independent of problem size
  - Larger tasks → fewer runtime cores
Weak Scaling: PRF on Titan

Fast Graph Analytics

- Conventional wisdom:
  - Graph processing has trouble taking advantage of distributed memory

- High performance graph processing systems are dominated by shared-memory CPU-based systems

- Observation
  - GPUs provide higher memory bandwidth than CPUs
  - Can avoid communication by careful placement of data in the memory hierarchy
Fast Graph Processing

Performance comparison on a single GPU (lower is better).

Competitive with state-of-the-art single-GPU graph processing engines.

Orders of magnitude speedup compared to state-of-the-art distributed/shared memory CPU systems.

Performance comparison among different graph processing frameworks (lower is better).

Convolutional Neural Networks (CNNs)

- In CNNs, data is commonly organized as 4D tensors.
  - tensor = [image, height, width, channel]

- Existing tools parallelize the image dimension.

- Motivation
  - Explore other parallelizable dimensions
  - Allow each layer to be parallelized differently
Results

Hiding Latency
Hiding Latency Through Deferred Execution

Performance Advantages

Tasking

• Ability to map and schedule
• Do work ahead of time

Message Passing

• No inherent advantage
  - Tasking can have fast messages, too
• And additional constraints
  - In order message delivery
But

- Two assumptions underlie tasking’s advantages
- There are extra compute resources
  - Otherwise can’t precompute decisions
- Tasking overheads are no worse than what message passing programmers write by hand
  - Programmers give up some control in tasking
  - What is that cost?

Summary: Performance

- No inherent performance advantage for message passing
  - If anything, tasking has been shown to have advantages
- The question is really whether tasking is performant enough compared to the non-message passing part of message passing codes
**Portability Advantages**

<table>
<thead>
<tr>
<th>Tasking</th>
<th>Message Passing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Program is machine independent</td>
<td>• Message passing layer is highly portable</td>
</tr>
<tr>
<td>• Remapping has been shown to give portability across a wide range of machines</td>
<td>• But across a narrower class of machines</td>
</tr>
</tbody>
</table>

**But**

- Does anyone care about portability?
- Yes, organizations care
  - They bear the risks/costs of long-lived software
- But programmers starting new projects have different concerns
  - Portability is usually low on the list
Summary: Portability

- Tasking has a clear advantage
- But the benefits are realized only in the long term

Programmability

Tasking | Message Passing

- Big data models have focused on programmability
  - Map-Reduce, Spark, Tensorflow
**Programmability**

<table>
<thead>
<tr>
<th>Tasking</th>
<th>Message Passing</th>
</tr>
</thead>
</table>
| • Big data models have focused on programmability  
  - Map-Reduce, Spark, Tensorflow | • C++ w/message passing is a complex programming model |

**But**

• Programmability is not a well-defined concept

• In the real world
  - C++ and MPI have huge numbers of trained programmers
  - Programmability often equated with familiarity

• Also HPC programming systems have many new features
  - Not found in popular big data systems, for example
Summary: Programmability

- Programmer productivity is real
  - But hard to measure

- C++/MPI has the advantage of a huge base
  - And that means continued investment in tools

- Tasking has two clear advantages
  - Porting, remapping
  - Composition of large systems
  - These are again long term advantages

System Complexity

<table>
<thead>
<tr>
<th>Tasking</th>
<th>Message Passing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Own much more of a program</td>
<td>Smaller footprint</td>
</tr>
<tr>
<td>System is inherently more complex</td>
<td>Simpler system</td>
</tr>
</tbody>
</table>
But

- System complexity is a fixed cost
  - In compiler, runtimes, other tools

- With enough programmers extra complexity is justifiable

But, But

- To get to a lot of programmers, you have to start with a small number of programmers

- And system complexity is super-linear in the number of features

- And remember Amdahl’s law!
  - Everything matters
Summary

• In the end performance, portability and programmability are not separable
  - It’s the whole package

• Tasking has clear advantages in all three areas
  - Performance, portability, programmability

• Focus now is on making it worth the trouble
  - In start-up cost, initial benefit, training

• We await your feedback!