The Case for Tasking

CS315B
Lecture 14

What is Tasking?

- Dependence graphs

- Operations
  - Nodes

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

The Case for Tasking

- Tasking is not good or bad, except in comparison to alternatives

- The competition is message passing
What 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?

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

What Does Message Passing Know?

- A message needs to be sent now
- A collective communication needs to be done
  - Usually now

What Do They Know?

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

- Message Passing

**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**

**Tasking**
- Ability to map and schedule
- Do work ahead of time
  - Hide latency

**Message Passing**
- More accurate computation of dependences

**Mapping**

**S3D: Combustion Simulation**

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

**Figure 5:** Computational domain and grid to be used for simulations of the CRF HCCI engine.

**Figure 6:** Reaction and diffusion structures for OH radical during the third stage thermal explosion of a high-pressure DME fueled autoignition process.

Recent 3D DNS of auto-ignition with 30-species DME chemistry (Bansal et al. 2011)
Mapping for Heptane 48³

- 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 96³

- 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 $\rightarrow$ fewer runtime cores

Weak Scaling: PRF on Titan

- 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

<table>
<thead>
<tr>
<th>Tasking</th>
<th>Message Passing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Ability to map and schedule</td>
<td>• No inherent advantage</td>
</tr>
<tr>
<td>• Do work ahead of time</td>
<td>• Tasking can have fast messages, too</td>
</tr>
<tr>
<td></td>
<td>• And additional constraints</td>
</tr>
<tr>
<td></td>
<td>• In order message delivery</td>
</tr>
</tbody>
</table>

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

Tasking
- Program is machine independent
- Remapping has been shown to give portability across a wide range of machines

Message Passing
- Message passing layer is highly portable
- But across a narrower class of machines

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

Message Passing
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></td>
</tr>
<tr>
<td>• Simpler system</td>
<td></td>
</tr>
</tbody>
</table>

• Smaller footprint

• System is inherently more complex
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
  - This is an issue for tasking implementations
  - Many things have to be well implemented for a range of applications

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!