Habanero Overview

Mainstream Parallel-Oblivious Programmers

Parallel-Aware Programmers

Concurrency Experts

- Support non-expert programmers on multicore systems
- Joe needs high level programming models designed for domain experts
- Stephanie needs simple parallel programming models with safety nets
- Today's parallel programming models emphasize Dreg
- Habanero goal: portable parallel software for Joe and Stephanie!

Parallel Programming with F# and TPL

- Functional
  - Higher order functions
  - Encourages use of immutable values
  - Type inference for floating point computations
  - Asynchronous computation expressions
- Imperative
  - Iterative updates (ref cells and arrays)
  - For loops
- .NET Compatibility
  - Full access to .NET libraries (e.g. TPL)

Comparing Phaser Accumulators With TPL

Key Lesson
Microsoft TPL needs efficient accumulation and synchronization

Microsoft

next / signal / wait

next = Notify “reached next” = signal (or ph.signal())
wait for others to notify = wait

- Semantics of next depends on registration mode
  - signal/wait: next = signal + wait
  - signal-only: next = signal (Don’t wait for any activity)
  - wait-only: next = wait (Don’t disturb any activity)

- A master activity is selected in activities w/ wait capability
- It receives all signals and broadcasts a barrier completion notice
- Master can be freed or unfree'd

Related Project at Rice: HPC Toolkit

http://hpctoolkit.org/

Habanero Multicore Software Project
http://habanero.rice.edu

Rice University
David Peixotto, Keith Cooper, John Mellor-Crummey, and Vivek Sarkar

Phasers (Help for Stephanie)

- Suitable synchronization for multiple communication patterns
  - Collective Barrier
    - All-to-all
  - Point-to-point synchronization
  - Specified producer-consumer
  - Deadlock freedom
  - Phase ordering property
  - Dynamic parallelism
    - If synchronized threads is varying dynamically
  - Lightweight synchronization
  - Local-spin blocking implementation

Phaser Accumulators

phasers ph = new phaser();
accumulator a = new accumulator(ph, SUM, Int32.class);
accumulator b = new accumulator(ph, MDL, double.class);
foreach (point i; i < n) phased {
  int iv = 2 * i + j;
  double dv = -1.5 + j;
  a.send(iv);
  b.send(dv);
}
result = send a value to accumulator
next = Result is accessible after next

result = Get a result value

Implementation Challenges: Garbage Collection

Memory Requirements for 2000x2000 Cholesky Factorization w/ and w/o Garbage Collection of Dead Items

The problem for Joe

- Most serial languages over-constrain orderings
  - Require arbitrary serialization
  - Allow for overwriting of data
  - Decision of if and when to execute are bound together
  - This makes parallel programming overhead
- Most parallel programming languages are embedded within serial languages
  - Inheret problems of serial languages
  - Too specific wrt type of parallelism in the application
  - And wrt the type target architecture
- Concurrent Collections Approach: introduce a coordination language that
  - Systematically eliminates over-constraints
  - Explicitly specify required constraints

Intel Concurrent Collections

Domain Expert:
- General
- Has domain knowledge
  - Not programming artifacts
  - Domain-specific knowledge

Sharing Expert:
- General
- Has domain knowledge
- Not programming artifacts
  - Domain-specific knowledge

Implementation Challenges: Garbage Collection

Concurrent Collections Program
- Explicit parallelism across and within steps
- Denial
- Overhead
- Overhead
- Overhead
- Overhead
- Overhead
- Overhead
- Overhead

Explicit parallel program (Intel TBB, Habanero/X10, or Microsoft TPL)

Source: Kathleen Yeates

Copyright © 2009 Rice University. All rights reserved.