PERFORMANCE PROGRAMMING & ITS PRODUCTIVITY

Basic Algorithm (sometimes informal)

Add data-structures (for serial algorithm)

Serial program (C)

Add parallel data-structures (for PRAM-like algorithm)

Parallel program (XMT-C)

Low overheads!

Standard Computer

Parallel Programming (Culler-Singh)

Decomposition

Assignment

Orchestration

Mapping

Parallel computer

XMT architecture (Simulator)

• 4 easier than 2
• Problems with 3
• 4 competitive with 1: cost-effectiveness; natural
  - HPCS puts to the test the practicality of the rich PRAM algorithmic theory and the UMD PRAM-On-Chip project
APPLICATION PROGRAMMING & ITS PRODUCTIVITY

Serial program (C)

- Application programmer’s interfaces (APIs)
  (OpenGL, VHDL/Verilog, Matlab)
- Compiler

Parallel program (XMT-C)
- XMT architecture
  (Simulator)

Automatic? Yes

Standard Computer

Parallel Programming (Culler-Singh)

- Decomposition
- Assignment
- Orchestration
- Mapping

Parallel computer

Yes

Maybe
PRAM: parallel random-access virtual machine model

- Ideal PRAM: latency for arbitrary number of memory accesses, same as for one access.
- Premise: algorithmicist states (or, does not hide..) what can be done concurrently.
- Algorithmic knowledge-base 2nd only to serial algorithms.
- Simplest parallel model.
EXAMPLE

Given: (i) All world airports. (ii) For each, all airports to which there is a non-stop flight.
Find: smallest number of flights from DCA to every other airport.

Basic algorithm
Step i: Given all airports that require i-1 flights, find (concurrently!) all those that require i flights.
Serial: uses “serial queue”.
- $O(T)$ time; $T$ – total # of flights

Parallel: parallel data-structures.
- Inherent serialization: $S$ - # of parallel steps: $\sim L$. Total # of operations: $\sim T$.

Gain relative to serial: (first cut)
$\sim T/S$!
Note: $\log(\deg(G))$ factors in parallel time and # of ops. Still: decisive!
- decisive gain also relative to coarse-grained parallelism

• Conclusion:
- Easy to program (natural!)
• **Course:** Parallel Algorithmics, Vishkin@UMD (CMSC751/ENEE759K Spring 2004)

• **Technology:** XMT C (C extension based on PRAM model)

• **Platform:** simulator of XMT architecture

• **Students:** 14

• **Assignments:** 2
  1. Array compaction
  2. Randomized selection

• **Hypotheses:**
  – For many problems, developing a parallel program is more natural than first developing a serial program, then parallelizing
  – XMT C programming is easier than other parallel programming approaches

• **Data collection:**
  – Data collection scripts capture reason for compile, compile timestamp, source code
• Name: Array Compaction

• Development Models: Transform given serial into XMT

• Grading Criteria: not specified to students

• Description:
  – Compact elements of array A into array D according to binary array C
    
    If C[i] (0<=i<=N) is equal to 1 then A[i] is copied into next free element of D
    
    To get the next free element in D a prefix sum (ps) instruction is used.
• **Name:** Randomized Selection

• **Development Models:** Derive a serial and a parallel implementation

• **Grading Criteria:** not specified to students

• **Description:**

  – Use XMT to implement parallel and serial programs for the following standard (expected linear time) iterative randomized algorithm:

    pick a random element \( r \)

    rank it

    check for termination

    iterate on a subset of the elements.
Reported speed-ups for a 1024-TCU (thread-control unit) XMT