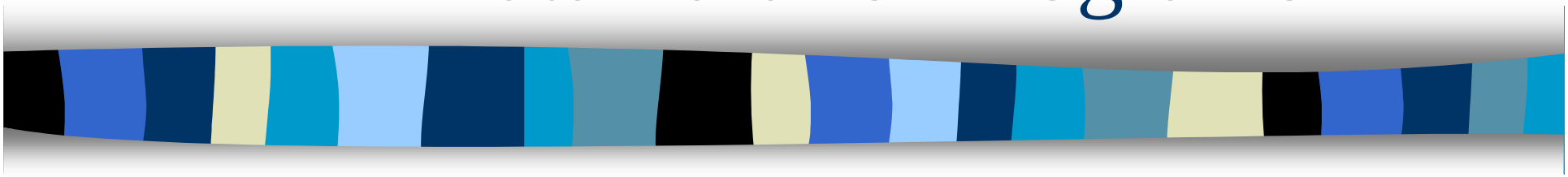
A Novel Execution Model for Data Parallel Programs



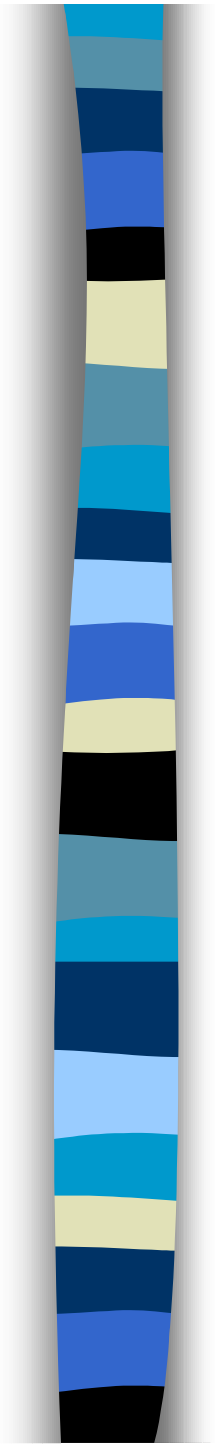
Arun Chauhan

with

Kath Knobe

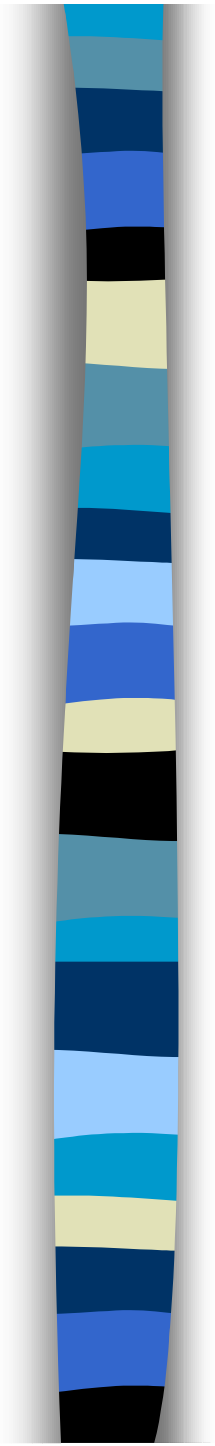
Reasons for the Talk

- Efficient way to get the message across
- Obtain feedback and comments
- Improve communication within the group ☺



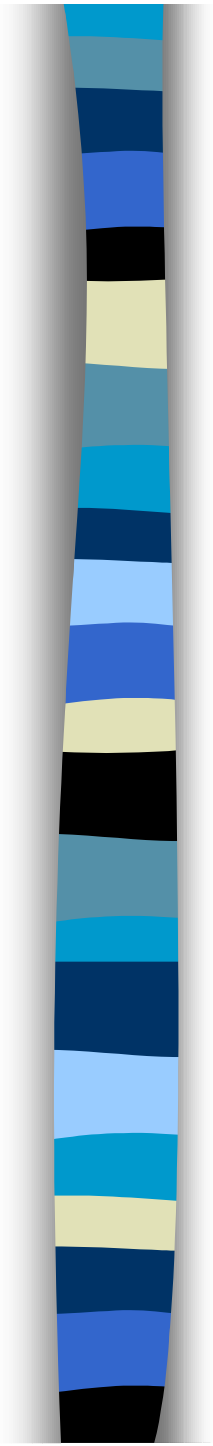
Motivation

- Irregular applications are hard to handle completely statically
- Completely general run-time system may become inefficient
- Compiler Analysis + light-weight efficient run-time system



Perspective for the Talk

- Idea for the run-time system based on CRL's Space-Time Memory
- Need input for integration with compiler analysis
- Work at a preliminary stage
- Immediate goals?



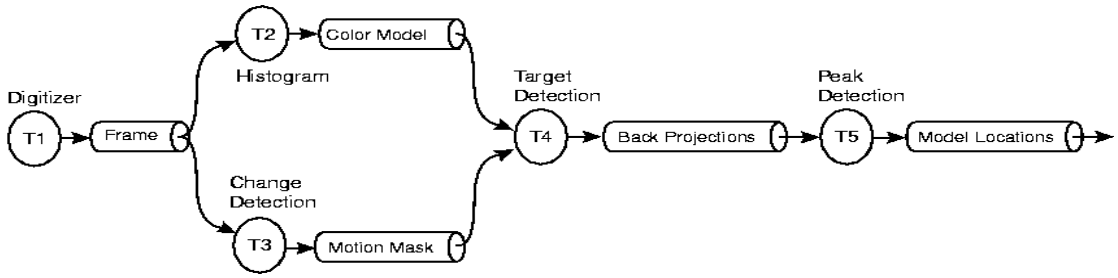
Space-Time Memory

- Originally designed to efficiently handle time-synchronized data
- Channel abstraction for data comm. as well as synchronization
- Different from message-passing and shared-memory

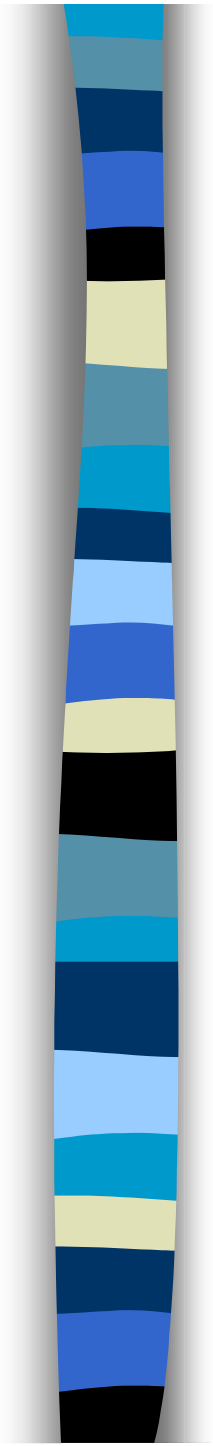
Stampede: STM Implementation

- Implemented as an API on top of Distributed Shared Objects as well as Message Passing
- Works on Digital Unix and Win-NT
- Being ported to MPI
- Uses CLF on top of Memory Channel or TCP

Typical use of STM



- Streaming data
- Time-synchronized data access
- Semi-automatic garbage collection
- Other features not relevant

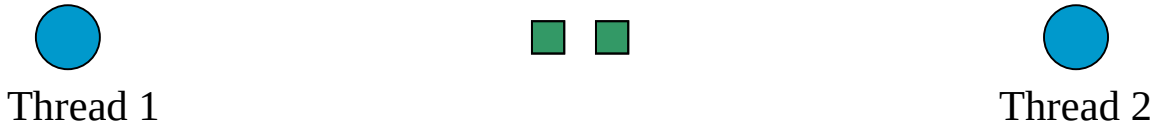


Channels

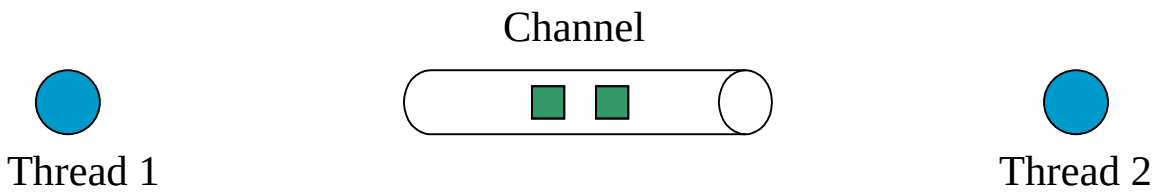
- Different from TCP sockets or Unix pipes as well as Shared Memory
- Data indexed (addressed) using time-stamps
- Data attributes
  - time-stamp: application generated
  - ref-count: aids in garbage collection

Primitives

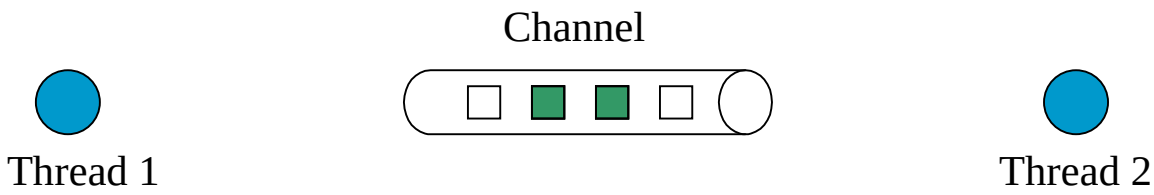
Primitives



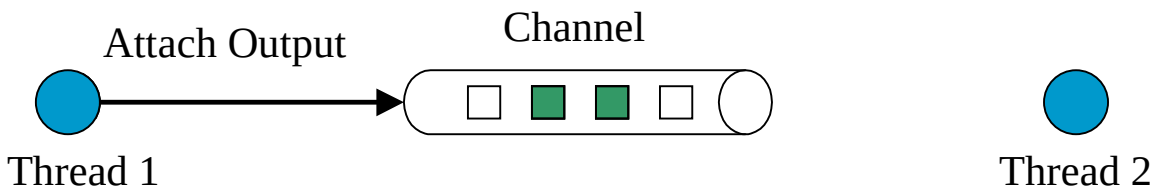
Primitives



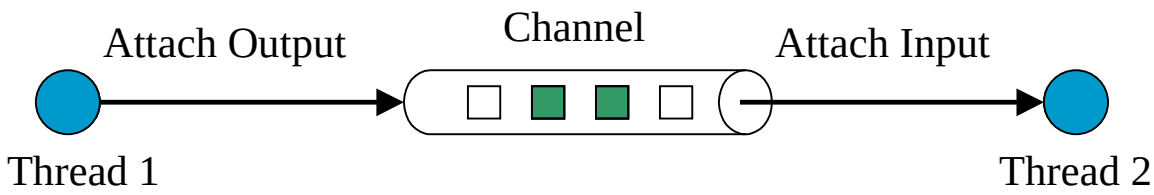
Primitives



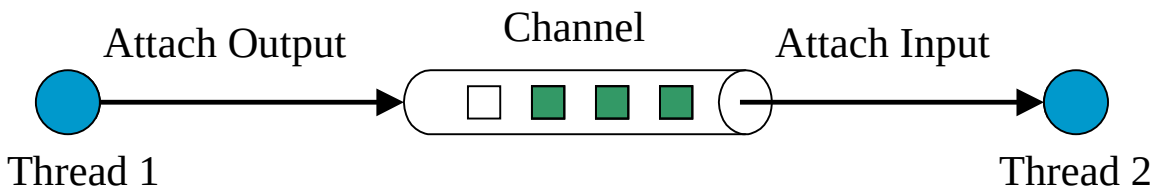
Primitives



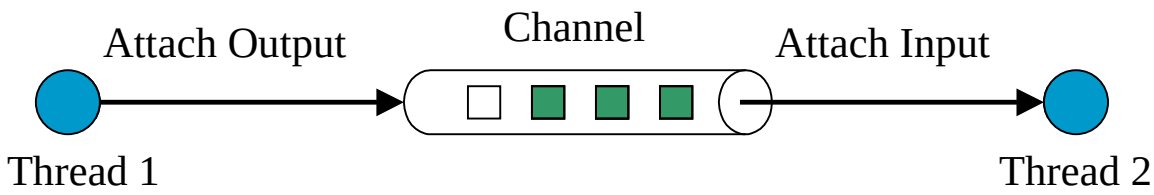
Primitives



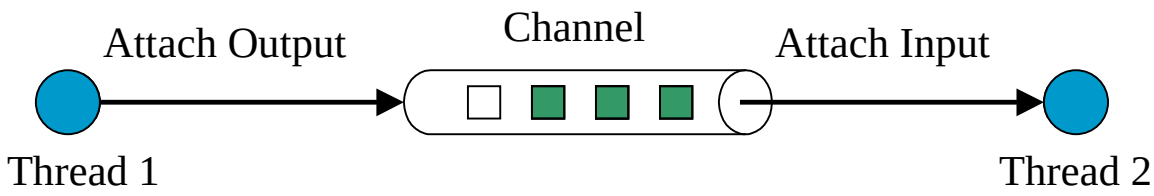
Primitives



Primitives



Primitives



thread create

channel create

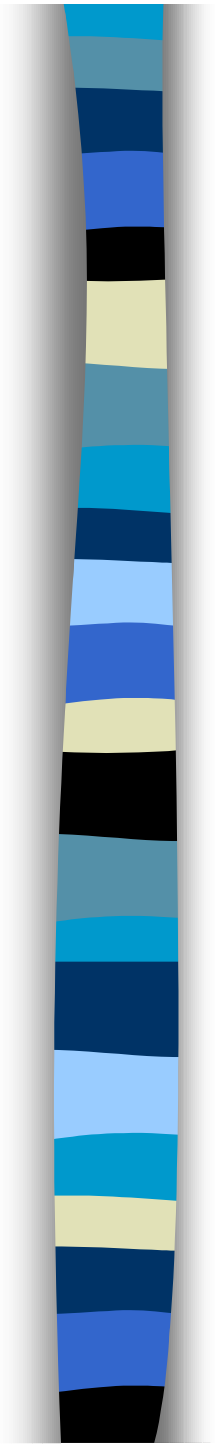
attach input

attach output

get data

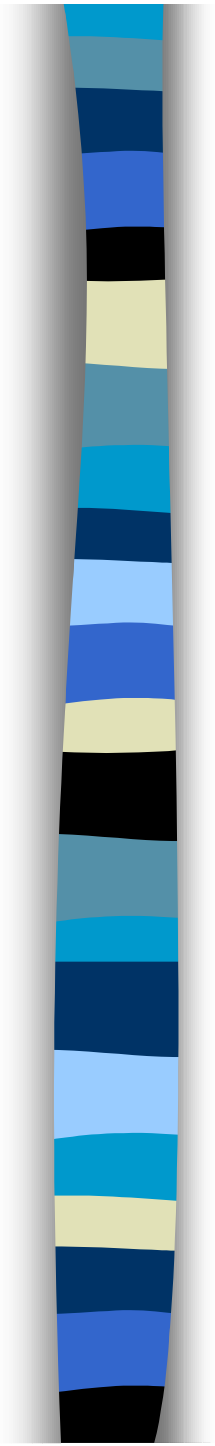
put data

consume data



Data Parallel Programs

- Mechanism to address data
- Mechanism to distribute and share data
- Mechanism to map data dependencies
- Handle dynamic environment



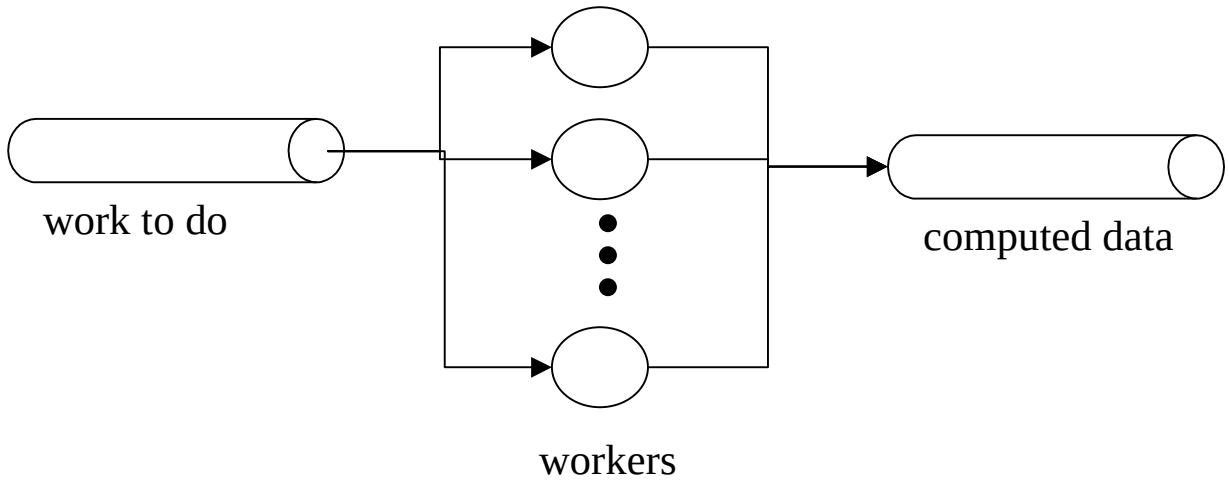
Data Parallel Programs

- Mechanism to address data
- Mechanism to distribute and share data
- Mechanism to map data dependencies
- Handle dynamic environment

Do this efficiently!

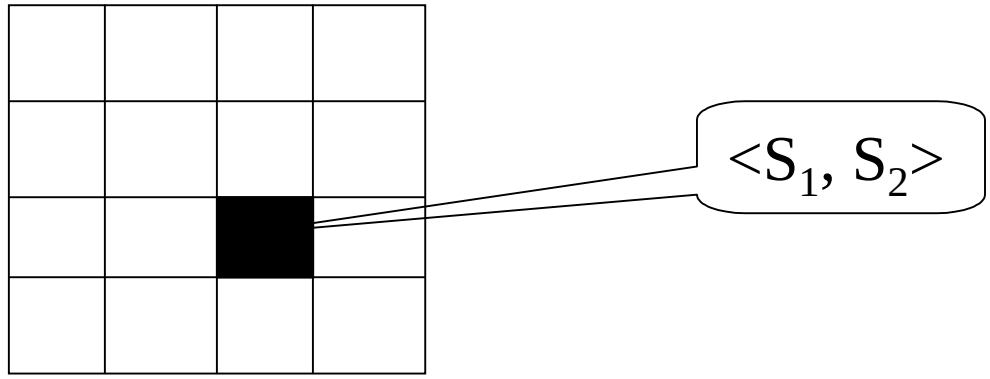
Why STM

- Experience with work-queue based data-parallelism



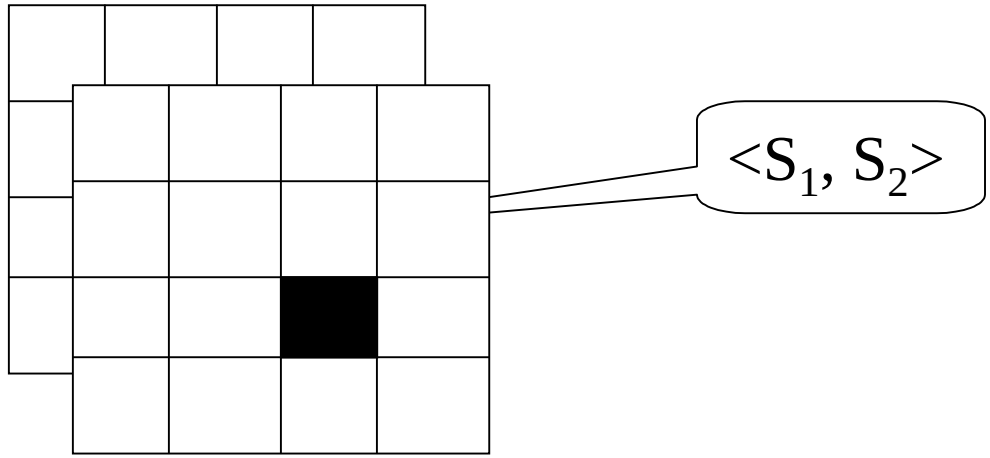
Why STM

- Iteration space = more complex version of time-stamps



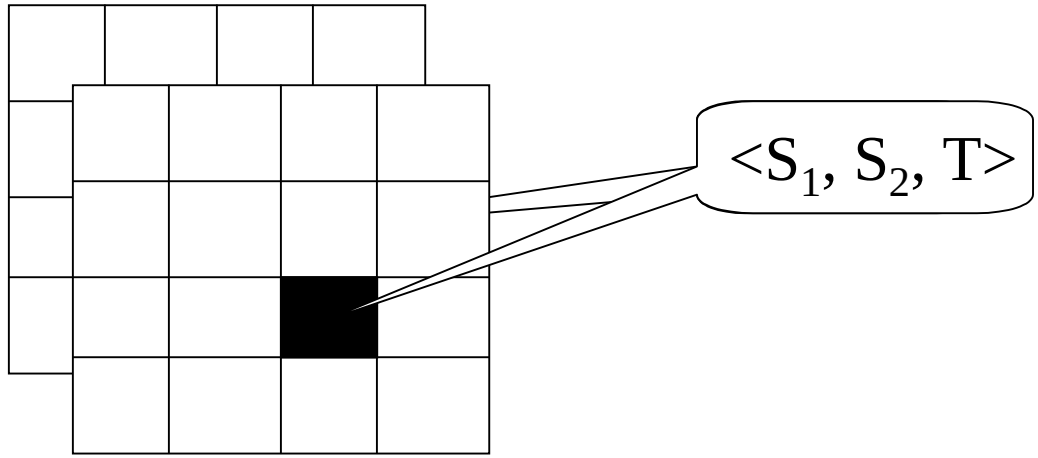
Why STM

- Iteration space = more complex version of time-stamps



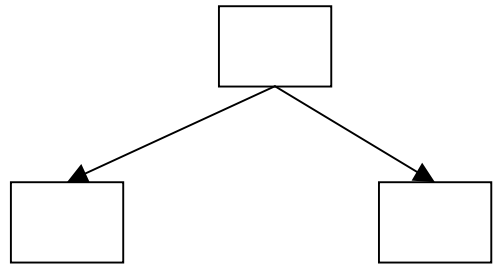
Why STM

- Iteration space = more complex version of time-stamps



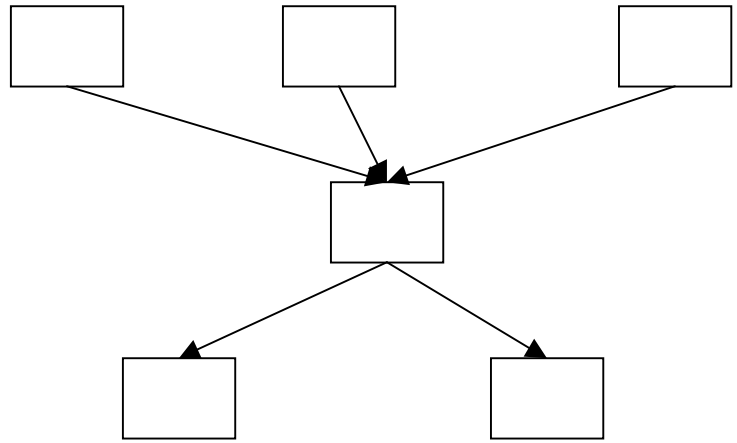
Why STM

- Ref counts similar to the notion of triggers for computation



Why STM

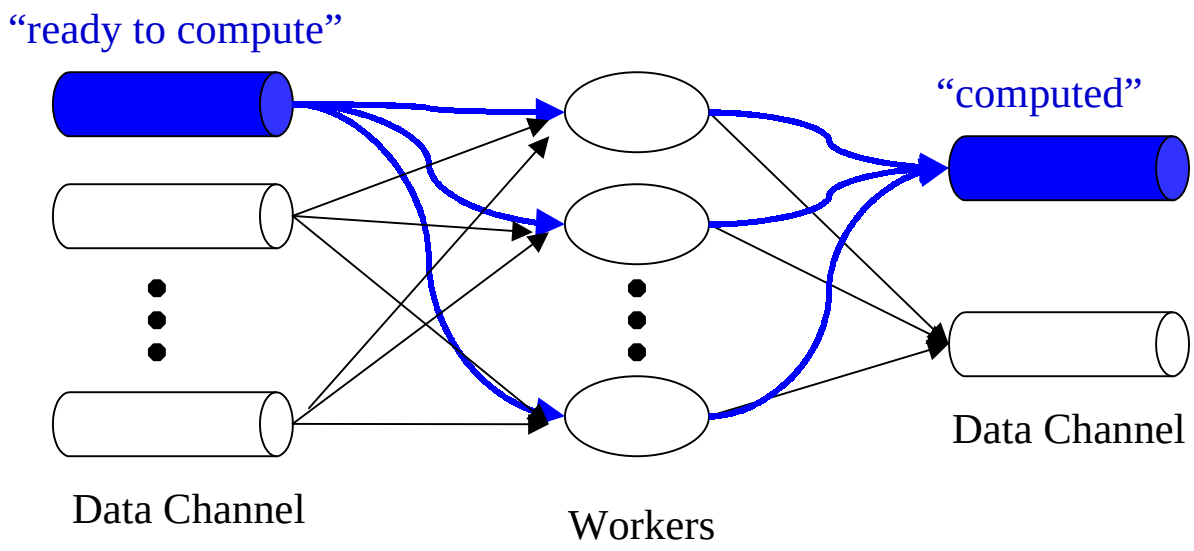
- Ref counts similar to the notion of triggers for computation



Key Ideas

- Use blocking to define small pieces of work
- Use an enhanced time-stamp mechanism to address data, and channels to communicate data
- Capture dependence relations between blocks
- Distribute pieces of work dynamically

Proposed Model



Proposed Model: Multi-dimensional Time Stamps

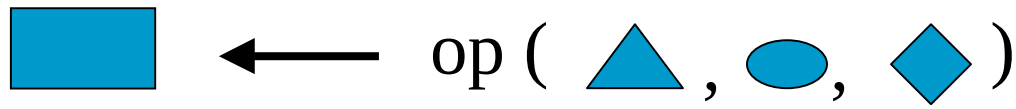
- $<T_1, T_2, T_3, \ldots, T_n>$
- Each dimension represents a spatial or temporal dimension in original program
- Time-stamps (multi-dimensional labels) identify data *values* – blocks, to be precise

Proposed Model: Multi-dimensional Time Stamps

- $<T_1, T_2, T_3, \ldots, T_n>$
- Each dimension represents a spatial or temporal dimension in original program
- Time-stamps (multi-dimensional labels) identify data *values* – blocks, to be precise

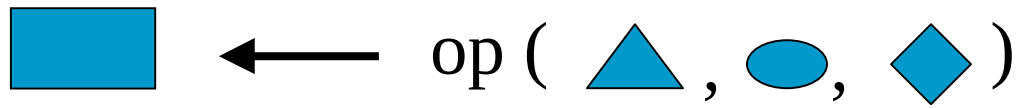
Mechanism to address data

Proposed Model: Enabling Reference Count



- Associate a dependence list with each data item
- Associate an enabling ref-count with each data item
- Generate a “ready-to-compute” item when all dependencies are satisfied

Proposed Model: Enabling Reference Count



- Associate a dependence list with each data item
- Associate an enabling ref-count with each data item
- Generate a “ready-to-compute” item when all dependencies are satisfied

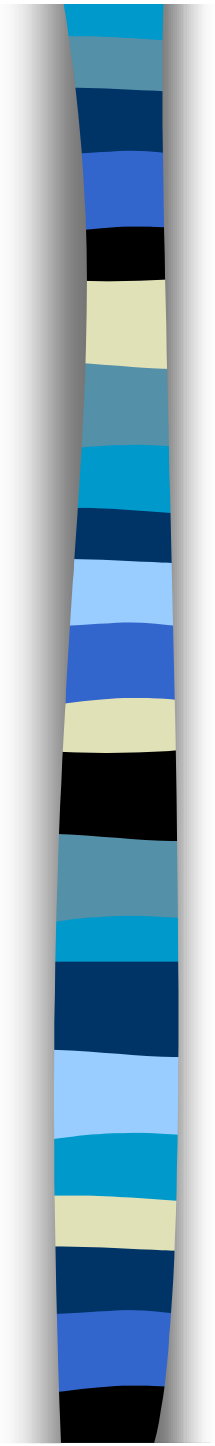
Mechanism to handle dependencies

Proposed Model: Handling Dynamic Environment

- Workers load-balance automatically
- A single queue could become bottleneck on a slow network
- Hierarchical approach to match the computing workers graph with network topology
- Has worked for multi-media applications
- Need more designing work!

Proposed Model: Recap

- Data divided into blocks representing units of work
- Channels to communicate data and control information
- Multi-dimensional time-stamps to capture data values
- Enabling ref-counts to capture data dependencies

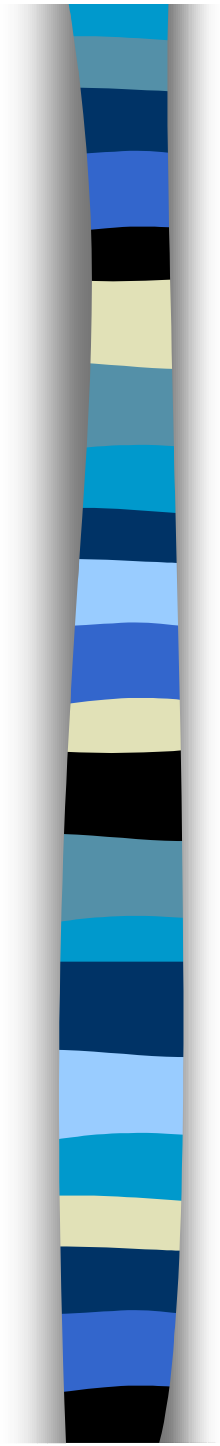


Categories of Applications

- Jacobi, Cholesky
- Fixed Irregular Mesh (e.g., airplane wing simulation)
- Dynamic Irregular (e.g., N-body)
- More ...

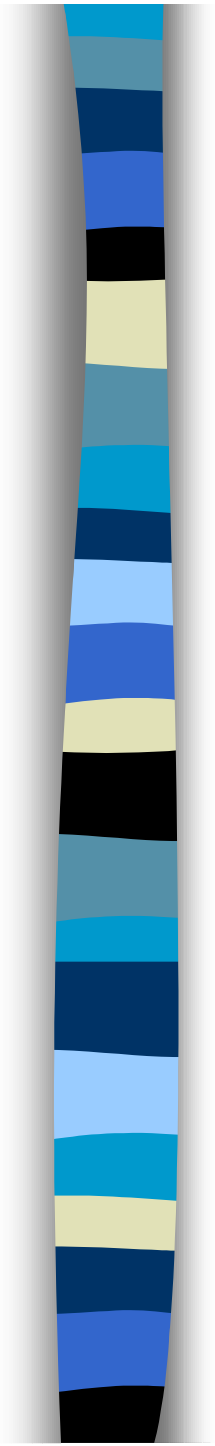
Advantages of our approach

- Simple model
- Good space usage
- Potential for good data locality
- Potential for good scalability
- Can take advantage of compiler analysis in data distribution



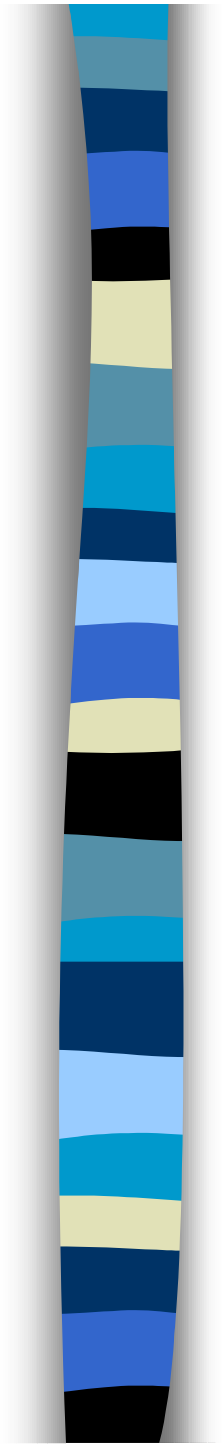
Potential Pitfalls

- Performance of Stampede
- Ability to capture a variety of parallel programs
- Scalability



Inputs

- Are we missing anything in terms of requirements for data parallel programs?
- How can we make good use of compiler analysis techniques?
- What should be the immediate goal?



End of Talk