Chapel Futures

Shams Imam, Vivek Sarkar
shams@rice.edu, vsarkar@rice.edu
Department of Computer Science, Rice University

Chapel Lightning Talks 2013
Futures: Begin Expressions

Example

// Parent task creates child task (begin-expression)
const top_sum: future(int) = begin compute_sum(X, low, mid);
...

// Later, parent examines the return value
var sum = top_sum.read() + ...;

Parent Task

const top_sum = begin ...;
...
top_sum.read()

Child Task

proc compute_sum(...) {
  return ...
}

container

return value
Computation Graphs

- Express Computation DAGs
- More general than regular begin tasks + sync statements
- More structured than begin tasks + sync variables

1. const A = \texttt{begin} fA(...);
2. const B = \texttt{begin} fB(A.read(), ...);
3. const C = \texttt{begin} fC(..., A.read());
4. const D = \texttt{begin} fD(B.read(), C.read(), ...);
5. const E = \texttt{begin} fE(C.read(), ...);
6. const F = \texttt{begin} fF(..., D.read(), E.read());
Futures

- Add support for begin expressions
  - Task with return values
  - `read()` operation blocks until value becomes available
  - No race conditions in container accesses
  - Can support arbitrary computation DAGs

- Operations, such as assignment and parameter passing, performed on unresolved futures without blocking

- No deadlock cycle can be created with `read()` calls
  - If all futures are stored in immutable variables
Futures and Single variables

- Futures are higher level constructs than single variables
  - Fits into Chapel's multi-resolution language design philosophy

- A future is guaranteed to have a specific producer task

- Easy for the implementation to determine the producer task for a given future.

- Can help in relating locales of future objects to locales for future tasks.

- Any one of a number of tasks can assign to the single variable

- Allows nondeterminism in the producer of the single value

- A single value may never be assigned (i.e. not guaranteed to have a producer) if it is not required
Current Status and Future Work

- **Implement future as a library**
  - Implemented using sync variables
  - Users can instantiate and use future objects

- **Add future types and implement begin expressions**
  - Compiler support for begin expressions

- **Add automatic coercions from future T to T**
  - Compiler adds type inference support and generates calls to read()
    ```
    const x = begin foo() + begin bar();
    ```

- **Add some form of statement-block expression**
  - Multiple statements inside begin expressions

- **Compiler/Runtime Optimizations**
  - Avoid task creation for short-lived computations
Resources

• **Available as a Subversion branch on sourceforge**
  
  — [http://svn.code.sf.net/p/chapel/code/branches/collaborations/futures](http://svn.code.sf.net/p/chapel/code/branches/collaborations/futures)
  
  — Contains multiple examples including SmithWaterman and Cholesky benchmarks (which show promising speedup)

• **Mailing list thread**
  
  