CRAY
THE SUPERCOMPUTER COMPANY

Chapel: Task Parallelism

SC12
Salt Lake City, Utah

CRAY
CHAPEL

Task Creation: Begin

- Syntax

```
begin-stmt:
  begin stmt
```

- Semantics
  - Creates a task to execute *stmt*
  - Original ("parent") task continues without waiting
- Example

```
begin writeln("hello world");
writeln("good bye");
```

- Possible output

```
hello world
good bye
```

```
good bye
hello world
```

Last week's Pthreads addOne() example in Chapel

```
var result: int;

proc addone(arg: int) {
  writeln("task running addone(", arg, ")");
  result = arg+1;
}

sync {
  begin addone(3);
}
writeln("result was ", result);
```

Block-Structured Task Creation: Cobegin

- Syntax

```
cobegin-stmt:
  cobegin { stmt-list }
```

- Semantics
  - Creates a task for each statement in *stmt-list*
  - Parent task waits for *stmt-list* tasks to complete
- Example

```
cobegin {
  foo(1);
  foo(2);
  bar();
}  // wait here for both foo()s and bar() to return
```

Loop-Structured Task Invocation: Coforall

- Syntax

```
coforall-loop:
  coforall index-expr in iteratable-expr { stmt-list }
```

- Semantics
  - Create a task for each iteration in *iteratable-expr*
  - Parent task waits for all iteration tasks to complete
- Example

```
config const numTasks = here.numCores;

coforall tid in 0..#numTasks do
  writeln("Hello, world! ",
          "from task ", tid, " of ", numTasks);
```

Comparison of Begin, Cobegin, and Coforall

begin:

- Use to create a dynamic task with an unstructured lifetime
- "fire and forget"

cobegin:

- Use to create a related set of heterogeneous tasks
- …or a small, finite set of homogenous tasks
- The parent task depends on the completion of the tasks

coforall:

- Use to create a fixed or dynamic # of homogenous tasks
- The parent task depends on the completion of the tasks

Note: All these concepts can be composed arbitrarily

Joining Sub-Tasks: Sync-Statements

- Syntax

```
sync-statement:
  sync stmt
```

- Semantics
  - Executes *stmt*
  - Waits for all *dynamically-scoped* begins to complete
- Example

```
sync {
  for i in 1..numFoos {
    begin foo(i);
  }
  bar();
}
```

```
proc search(N: TreeNode) {
  if (N != nil) {
    begin search(N.left);
    begin search(N.right);
  }
}
sync { search(root); }
```

Sync-Statements and Dynamic Scoping

Where the cobegin statement is static…

```
cobegin {
  functionWithBegin();
  functionWithoutBegin();
}  // waits on these two tasks, but not any others
```

…the sync statement is dynamic.

```
sync {
  begin functionWithBegin();
  begin functionWithoutBegin();
}  // waits on these tasks and any other descendents
```

Sync-Statements and Program Termination

Program termination is defined by an implicit sync on the main() procedure:

```
sync main();
```