Managing Hierarchy with Teams in the SPMD Programming Model

Amir Kamil
Lawrence Berkeley Lab
Berkeley, CA, USA
April 28, 2014
The single program, multiple data (SPMD) execution model is the dominant programming model at scale:
- A fixed set of threads execute the same program.
- Synchronization and communication primarily through collective operations.

```java
public static void main(String[] args) {
    System.out.println("Hello from " + Ti.thisProc());
    Ti.barrier();
    if (Ti.thisProc() == 0) System.out.println("Done.");
}
```
Thread Teams

- Thread *teams* are basic units of cooperation
  - Groups of threads that cooperatively execute code
  - Collective operations over teams

- Flat teams provided by MPI, GASNet

- Hierarchical teams used in Titanium, UPC++, HCAF, DASH
  - Expressive: match structure of algorithms, machines
  - Safe: eliminate many sources of deadlock
  - Composable: enable clean composition of multiple algorithms or tasks
  - Efficient: allow users to take advantage of machine structure, resulting in performance gains
Team Data Structure

- Teams represented as tree structure
- Team structure can be created manually or automatically based on machine hierarchy

\[ \text{Team } T = \text{Ti.defaultTeam}(); \]

- Unbalanced structures can be created manually
Team-Usage Constructs

- Syntactic constructs specify dynamic scope of teams
  ```java
teamsplit(rowTeam) {
    Reduce.add(mtmp, myResults0, rpivot);
  }
  ```

- Collectives and queries such as `T1.thisProc()` are with respect to currently scoped team

- Constructs can be nested, and recursion can be used to dynamically handle team hierarchy of arbitrary depth
Example: Sorting

• Titanium distributed sorting application using new hierarchical constructs

• Three pieces: sequential, shared memory, and distributed
  - Sequential quick sort from Java 1.4 library
  - Shared memory merge sort
    • Hierarchical teams used to express recursive algorithm
  - Distributed memory sample sort
    • Teams used to optimize communication and to compose with shared-memory sort

• Goal: better performance than flat sample sorting, which assumes no threads share memory
Shared-Memory Team Hierarchy

• Team hierarchy for shared-memory part of computation is binary tree

• Trivial construction

```java
static void divideTeam(Team t) {
    if (t.size() > 1) {
        t.splitTeam(2);
        divideTeam(t.child(0));
        divideTeam(t.child(1));
    }
}
```

• Threads walk down to bottom of hierarchy, sort, then walk back up, merging along the way
Shared-Memory Computational Logic

• Control logic for sorting and merging

```java
static void sortAndMerge(Team t) {
    if (Ti.numProcs() == 1) {
        allRes[myProc] = sequentialSort(myData);
    } else {
        teamsplit(t) {
            sortAndMerge(t.myChildTeam());
        }
        Ti.barrier();
        if (Ti.thisProc() == 0) {
            int otherProc = myProc + t.child(0).size();
            int[1d] myRes = allRes[myProc];
            int[1d] otherRes = allRes[otherProc];
            int[1d] newRes = target(t.depth(), myRes, otherRes);
            allRes[myProc] = merge(myRes, otherRes, newRes);
        }
    }
}
```

Sort at bottom
Walk down team hierarchy
Walk up, merging along the way
Distributed-Memory Logic

• Flat distributed code

```java
static void flatSort() {
    myData = sampleAndDistribute(myData, Ti.thisProc());
    sequentialSort(myData);
}
```

• Hierarchical distributed code

```java
static void hierarchicalSort() {
    Team team = Ti.defaultTeam();
    myData = sampleAndDistribute(myData, team);
    teamsplit(t) {
        sharedMemorySort(myData);
    }
}
```

Split into shared-memory domains

Parallelize and aggregate communication between shared-memory domains

Shared-memory sort within shared-memory domains
Performance of Flat vs. Hierarchical Sort

Distributed Sort (Cray XE6)
(10,000,000 elements/core, 10,000 samples/core)

- blue: flat (distribution)
- green: hierarchical (distribution)
- red: flat (sort)
- purple: hierarchical (sort)

Time (s)
NUMA Nodes (6 cores/node)
Limitations of Hierarchical Teams

• Hierarchical teams have proven to be very effective in structuring execution in SPMD programs
  - Represent logical view of execution
  - Can be synthesized from physical structure of execution resources

• However, they are not sufficient to represent the structure of data
  - Data are located at specific memory locations in a machine
  - Not necessarily one-to-one mapping between execution and memory units

• Need combination of physical memory structure (where data should be located) and logical execution structure (how data will be operated on)
Hierarchical Resources

- A \textit{place} (X10, UPC++) or \textit{locale} (Chapel, HCAF) represents a location in the machine
  - Includes memory and/or execution resources

- Hierarchical places model hierarchy of resources

- Places can have memory units, execution units, or both
  - e.g. cache hierarchy modeled using memory-only places
Data Hierarchy

- **Data hierarchy**: full hierarchical structure of data
  - Encompasses hierarchical teams and places
  - Allow synthesis from hierarchical teams, places, or both

- Multidimensional hierarchy required in order to match multidimensional data structures
Hierarchically Tiled Arrays (HTAs) are well-suited to managing hierarchy in data-parallel programs.

Likely to be useful as distributed data structure in SPMD
- Support global-view/team-view collective operations
- Also support local-view computation
HTA Creation

- An HTA is created over a rectangular index space and a data hierarchy

\[
hta<T, N> \text{ array}(RD(PT(0, 0), PT(8, 8)), \text{ data}_\text{hierarchy});
\]

- Support block-cyclic and user-defined distributions, ghost/shadow regions, and replication for .5D algorithms
HTA Operations

- Access sub-tile (e.g. bottom-left tile)
  
  \[\text{array}(1, 0)(1, 0)\]

- Access element
  
  \[\text{array}[\text{PT}(7, 0)]\]
  \[\text{array}(1, 0)[\text{PT}(7, 0)]\]
  \[\text{array}(1, 0)(1, 0)[\text{PT}(7, 0)]\]

- Collective operations over tile or slice of HTA
  
  \[\text{array}(1, 0).\text{reduce}(\text{op})\]
  \[\text{array}.\text{slice}(1, 1).\text{map}_\text{reduce}(\text{mop}, \text{rop})\]

- Update ghost regions
  
  \[\text{array}.\text{update}()\]
  \[\text{array}.\text{async}_\text{update}()\]
Summary

• Hierarchical teams have been successful in expressing hierarchical algorithms and mapping execution onto hierarchical machines in SPMD programs

• Hierarchical places provide an abstraction of the resources in a machine

• Hierarchically Tiled Arrays (HTAs) proven to be valuable in data-parallel programming, likely will be in SPMD as well

• We are working on unifying these concepts in the DEGAS project
  - UPC++ at LBL, HCAF at Rice
  - In the process of finalizing design/interface, starting on implementation