TStreams

A Model of Parallel Computation

HP Cambridge Research Lab

Kath Knobe
Dave Panariti
Alex Nelson
Carl Offner
Dick Flower
**Background**

- **Streams**
  - + very efficient
  - - limited to FIFO

- **Functional**
  - + scheduling flexible
  - - regular accesses not efficient

- **Tuple spaces**
  - + decoupled producer/consumer
  - - not functional: scheduling constrained by overwriting
Static Components

space of tags: execution space and data space

space of steps: computation

space of items: data
Example 1: Gene Matching

Chromo data

Patterns to match

Match

Matched sites on chromo
Dynamic Instances

Each tag: a point in execution/data space

Each step: Mapping and scheduling

Each item: allocation, synchronization, communicated
Example 2: Divide & Conquer
Divide & Conquer

- Whole tree
- All but leaves
- Items to divide
- Divide or pass on
- All but root
- Items to combine
- Combine or pass on
Properties

◆ State
  – tags in tag spaces, steps executed, produced items

◆ Atomicity
  – Tag: in a tag space or not
  – Step: executed or not
  – Item: produced or not

◆ Implications
  – redundant tag production, step execution, item production
  – start, drop and restart execution of steps
    start, drop and restart production of a tag or an item

◆ Dynamic, continuous, transparent checkpointing
# Execution Styles

<table>
<thead>
<tr>
<th></th>
<th>grain</th>
<th>map</th>
<th>schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>data parallel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>task parallel</td>
<td>static</td>
<td>static</td>
<td>static</td>
</tr>
<tr>
<td>pipeline parallel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data flow</td>
<td>static</td>
<td>static</td>
<td>dynamic</td>
</tr>
<tr>
<td>reliable</td>
<td>static</td>
<td>dynamic</td>
<td>dynamic</td>
</tr>
<tr>
<td>adaptive</td>
<td></td>
<td></td>
<td>dynamic</td>
</tr>
</tbody>
</table>

- **Current implementation**
- **In progress**
Compiler Algorithms: Transform Normal Programs to TStreams

- Subspace Analysis (Bill Dally)
- Array SSA (Vivek Sarkar)
- DSA (Carl Offner)
Original Code

\[
X(\ldots) = \ldots
\]

\[
\ldots = X(\ldots)
\]

\[
X(\ldots) = \ldots
\]

\[
\ldots = X(\ldots)
\]
Static Single Assignment

\[ X_1(...) = \ldots \]

\[ \ldots = X (...) \]

\[ X_3(...) = \ldots \]

\[ X_5(...) = \ldots \]

\[ \ldots = X (...) \]
Static Single Assignment

\[
\begin{align*}
X_1(\ldots) &= \ldots \\
X_2 &= \phi(X_0, X_1) \\
\ldots &= X(\ldots) \\
X_3(\ldots) &= \ldots \\
X_4 &= \phi(X_2, X_3) \\
X_5(\ldots) &= \ldots \\
X_6 &= \phi(X_1, X_0) \\
X_7 &= \phi(X_4, X_6) \\
\ldots &= X(\ldots)
\end{align*}
\]
Static Single Assignment

- $X_1(...)=\ldots$
- $X_2=\phi(X_0, X_1)$
  
  $\ldots = X_2(\ldots)$

- $X_3(...)=\ldots$
- $X_4=\phi(X_2, X_3)$

- $X_5(...)=\ldots$
- $X_6=\phi(X_1, X_0)$

- $X_7=\phi(X_4, X_6)$

  $\ldots = X_7(\ldots)$
Dynamic Single Assignment

\[
\begin{align*}
&X_1(...)[k] = \ldots \\
&X_2 = \phi(X_0, X_1) \\
&\ldots = X_2(...)[k] \\
&X_3(...)[k] = \ldots \\
&X_4 = \phi(X_2, X_3) \\
&X_5(...)[k] = \ldots \\
&X_6 = \phi(X_1, X_0) \\
&X_7 = \phi(X_4, X_6) \\
&\ldots = X_7(...)[k] 
\end{align*}
\]
Tags

Item: X(d1, d2, d3, d4)

Step: Loop i1
  Loop i2
  Loop i3
  Loop i4

Within item

Item tag

Step tag

Within step
Summary

- Best of streaming languages, functional languages and tuple-spaces
- Versatile execution styles
- Automatic generation
- (Hierarchical)
- (real-time)
END
Mappings

- **mapping:** producer step to produced item
- **mapping:** consumed item to consumer step
- **mapping:** tag to step or item
- **Identity mapping:** Each item produced by single step

Arbitrary