Programming Spatial Computers

Jonathan Bachrach & Jacob Beal
October, 2006
Space-filling Computers

Sensor networks

Biological Computing

Distributed Control Systems

Robot Swarms

FPGAs

Programmable Matter
Amorphous Medium Approach
Amorphous Medium Approach
Amorphous Medium Approach

Program the space... approximate with a network
Amorphous Medium Approach

The discretization hardly matters!
Global v. Local v. Discrete

Compiler

Kernel

Program

Global

Local

Discrete
Global v. Local v. Discrete
Discrete Model

- Dozens to billions of simple, unreliable agents
- Distributed through space, communicating by local broadcast
- Agents may be added or removed
- No guaranteed global services (e.g. time, naming, routing, coordinates)
- Relatively cheap power, memory, processing
- Partial synchrony
Kernel

• Responsibilities:
  – Emulate amorphous medium
  – Time evolution
  – Interface with sensors, actuators
  – Viral reprogramming

• Current platforms:
  – Simulator
  – Mica2 Mote
  – McLurkin's Swarm
Amorphous Medium

- Manifold (locally Euclidean space)
  - Assume Riemannian, smooth, compact
  - Simple locally, complex globally
Amorphous Medium

- Points access past values in their neighborhood
  - Information propagates at a fixed rate $c$
- Evaluation is repeated at fixed intervals
Neighborhood Abstraction

- Aggregate access to best-effort estimate of neighbor state, space-time properties
- Neighbors decay without updates
Kernel Open Questions

• What is the optimal replacement policy when there are more neighbors than table memory?
• What is the optimal decay rate?
• How much energy can be saved by throttling update and decay rates?
• What are good ways to expose the cost/responsiveness tradeoff to the programmer?
• An expression maps a manifold to a field
  \( rgn : M \rightarrow (M \rightarrow \mathbb{R}) \)
Expressions

- An expression maps a manifold to a field

\[ \text{rgn}: M \rightarrow (M \rightarrow \mathbb{R}) \]
Operators map fields to fields \((\text{rgn} \ 2)\)

\[ (M \to R) \times (M \to R) \to (M \to B) \]
Composition & Abstraction

- Functional composition:
  - \( \text{operator} \circ \text{expressions} = \text{expression} \)
  - \( \text{operator} \circ \text{operators} = \text{operator} \)
  - \( \text{scope} \circ \text{expression} = \text{operator} \)

\} \text{ Lambda!} \}

Purely functional pointwise computation
Computation over Neighborhoods

(or ((n xor) (nbrval x) (local x)))
Computation over Neighborhoods

(or ((n xor) (nbrval x) (local x)))

- local, nbrval select fields of neighborhood fields
Computation over Neighborhoods

\[(\text{n xor})\]

- \(n\) applies an operator to neighborhood fields
Computation over Neighborhoods

\[(\text{or } ((n \text{ xor}) \text{ (nbrval } x) \text{ (local } x)))\]

- Measures (e.g. \text{or}, \text{integral}) reduce fields to values
- Sugar: \((\text{reduce-nbrs or (xor } x \text{ (local } x)))\)
Neighborhood Open Questions

- Are the summary operations \textit{and}, \textit{or}, \textit{min}, \textit{max}, and \textit{integral} sufficient for all approximatable continuous neighborhood computations?

- Are the field primitives \textit{local}, \textit{nbrval}, \textit{nbr-range}, \textit{nbr-lag}, and \textit{nbr-bearing} sufficient sources of neighborhood data?

- What is the discretization error of arbitrary composite neighborhood computations?
Conditional Computation

\[(\text{mux } x \ (\text{or} \ (\text{nbrval} \ (\text{restrict} \ x \ x \ \text{vent})) \ #F))\]
Conditional Computation

\[ \text{mux } x \ (\text{or} \ (\text{nbrval} \ (\text{restrict} \ x \ \text{vent})) \ #F) \]

- \text{restrict} limits the domain of a field
Conditional Computation

\[(\text{mux } x \ (\text{or} \ (\text{nbrval} \ (\text{restrict} \ x \ \text{vent})) \ #F))\]

- operations proceed normally in the restricted field
Conditional Computation

\( (\text{mux } x \ (\text{or} \ (\text{nbrval} \ (\text{restrict} \ x \ \text{vent})))) \ \#F \)

- \text{mux} constructs a field piecewise from inputs
- Sugar: \((\text{if } x \ (\text{or} \ (\text{nbrval} \ \text{vent})))\)
Computation with State

(delay default init)

- Previous values, current domain
Computation with State

(delay default init)

- Previous values, current domain
Computation with State

(delay default init)

- Previous values, current domain
Computation with State

(delay default init)

- Previous values, current domain
Computation with State

(letf fed ((n 0 (+ n 1))) n)
Putting it all together

- State chains neighborhoods to arbitrary regions
  - Example: relaxation to calculate distance

\[
\text{(lambda (src)}
\text{(letfед)
  \text{((d } \infty \text{ (mux src 0
    (reduce-nbсs min (+ d nbr-range))))))}
\text{d))}
\]
Putting it all together

- State chains neighborhoods to arbitrary regions
  - Example: relaxation to calculate distance

```
(lambda (src)
  (letfed
    ((d  (∞ (mux src 0
      ((d  (∞ (reduce-nbrs min (+ d nbr-range)))))
    d))
```

Diagram of a region with a path from 0 to infinity.
Putting it all together

- State chains neighborhoods to arbitrary regions
  - Example: relaxation to calculate distance

```lisp
(lambda (src)
  (letf!d
    (((d ∞ (mux src 0
        (reduce-nbrs min (+ d nbr-range)))
      d)))
```
Putting it all together

- State chains neighborhoods to arbitrary regions
  - Example: relaxation to calculate distance

\[
\text{letfed}
\]
\[
\left( (d \infty (\text{mux src 0})
\right)
\]
\[
(\text{reduce-nbrs min (+ d nbr-range)))
\]
\[
d
\]
Program Open Questions

- Under what conditions does continuous convergence imply discrete convergence?
- How do convergence properties compose?
- Given a continuous program and desired error bounds, what discretization will suffice?
- Given a continuous program and a discretization, what will the error bounds be?
Programs scale gracefully

100 nodes

(and (green (dilate (sense 1) 30))
  (blue (dilate (sense 2) 20)))
Programs scale gracefully

1,000 nodes

(and (green (dilate (sense 1) 30))
  (blue (dilate (sense 2) 20)))
Programs scale gracefully

10,000 nodes

(and (green (dilate (sense 1) 30))
  (blue (dilate (sense 2) 20)))
Target Tracking
(def local-average (v) (/ (reduce-nbrs v integral) (reduce-nbrs integral 1)))
(def gradient (src)
  (letfed ((n infinity
              (+ 1 (mux src 0 (reduce-nbrs min (+ n nbr-range)))))))
  (- n 1)))
(def grad-value (src v)
  (let ((d (gradient src))
         (x 0 (mux src v (2nd (reduce-nbrs min (tup d x)))))
       x))
(def distance (p1 p2) (grad-value p1 (gradient p2)))
(def channel (src dst width)
  (let* ((d (distance src dst))
         (trail (<= (+ (gradient src) (gradient dst)) d))
         (dilate width trail)))
(def track (target dst coord)
  (let ((point
    (if (channel target dst 10)
      (grad-value target
       (mux target
        (tup (local-average (1st coord))
         (local-average (2nd coord))
         (tup 0 0)))
        (tup 0 0))))
    (mux dst (vsub point coord) (tup 0 0)))))
Threat Avoidance
Threat Avoidance

(defun exp-gradient (src d)
  (letf (\n    ((n src (max (* d (reduce-nbrs max n)) src)))
    \n    (def sq (x) (* x x))
  (def dist (p1 p2)
    (sqrt (+ (sq (- (1st p1) (1st p2)))
      (sq (- (2nd p1) (2nd p2))))))
  (def l-int (p1 v1 p2 v2)
    (pow (/ (- 2 (+ v1 v2)) 2) (+ 1 (dist p1 p2))))
  (def max-survival (dst v p)
    (letf (\n      ((ps 0 (mux dst
        1
        (reduce-nbrs max (* (l-int p v (local p) (local v)) ps)))
        ps)))
  (def greedy-ascent (v coord)
    (- (2nd (reduce-nbrs max (tup v coord))) coord))
  (def avoid-threats (dst coords)
    (greedy-ascent
      (max-survival
        dst
        (exp-gradient (sense :threat) 0.8) coords) coords))
Future Directions

- Continuous time evaluation
- Analysis of distortion from space discretization
- Evaluation on a changing manifold
- Actuation of the manifold
- Applications!