Time Warp: Global Control
Distributed Snapshots and Fossil Collection

Richard M. Fujimoto
Professor

Computational Science and Engineering Division
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0765, USA

http://www.cc.gatech.edu/~fujimoto/

Copyright © Richard M. Fujimoto
Outline

• Consistent Cuts
  – Cut points
  – Cut messages
  – Cut values

• Mattern’s GVT Algorithm
  – Colors
  – Vector counters
  – Pipelined algorithm

• Fossil Collection
Mattern’s Algorithm

• Asynchronous: executes in background concurrent with time warp execution (does not require the simulation to block)
• Avoids message acknowledgements
• Based on techniques for creating distributed snapshots
**Mattern: Consistent Cuts**

*cut point*: an instant dividing computation into past and future

*cut*: set of cut points, one per processor

*cut message*: a message that was sent in the past, and received in the future

*consistent cut*: a cut where all messages crossing the cut are cut messages

*cut value*: minimum among (1) local minimum of each LP at its cut point and (2) time stamp of cut messages
Any message crossing cut from future to past must have a time stamp > the cut value, so they can be ignored when computing the cut value.

Message generated by an LP after its cut point must have time stamp greater than the minimum of

- The LP’s local minimum at its cut point
- The time stamp of messages received after the cut point

\[ T_1 > Y \]
\[ T_2 > \min (Y, U) \]
\[ T_3 > \min (Y, U, S) \]

\( S > X \), so

\[ T_3 > \min (Y, U, X) \]

(Cut value \( \leq X, Y, U \))
Observation 2

- Cut value equal to GVT(T) using synchronous GVT algorithm
- Events generated after cut have time stamp > cut value
- Cut value can be used as a GVT value

Cut value = \( \min (W, X, Y, Z, U, V) \)

GVT(T) = \( \min \) ts, unprocessed message @ T
\( = \min (W, X, Y, Z, U, V) \)
Mattern’s GVT Algorithm

Challenge: accounting for cut messages

Approach

• Construct two cuts C1, C2, approximate cut value along C2
  – Organize processes in ring, pass token around ring

• Ensure no message that crosses C1 also cross C2
  – Color LPs, change LP color at each cut point
  – Color each message to that of LP sending message (message tag)
  – Maintain send/receive message counters

• GVT = min (local min along C2, time stamp of red messages)
Algorithm Overview

The first cut
- Changes color of each process (green to red)
- Determine number of green messages sent to each process

The second cut:
- Each process makes sure all green messages sent to it have been received before laying down a cut point
- Compute GVT: min among local mins, red messages
How an LP knows it has received all its green messages

- \( V_i[1:N] \), where \( N = \#\text{LPs} \)
  - \( V_i[i] \) = number of green messages received by \( \text{LP}_i \)
  - \( V_i[r] \) = number of green messages sent by \( \text{LP}_i \) to \( \text{LP}_r \)

- \( \text{C2}: \text{LP}_i \) cannot pass token until
  - \( V_i[i] = \sum V_s[i] \) (summed over all \( s \neq i \))

- \( \text{C1}: \) Token includes vector to accumulate send counters
Example: Vector Counters

Vector counters for green messages (at C2):

<table>
<thead>
<tr>
<th></th>
<th>V₁</th>
<th>V₂</th>
<th>V₃</th>
<th>V₄</th>
</tr>
</thead>
</table>
Local Variables (in each logical process LP\(_i\)):
- \(T\text{\_red} = \text{min time stamp among red messages sent by LP} \) (even non-cut red messages!)
- \(V_i[1:N] = \text{message send / receive counters}\)

Token: CM\text{msg}
- \(\text{CM\text{msg}\_T\_min} = \text{accumulator, smallest local minimum so far}\)
- \(\text{CM\text{msg}\_T\_red} = \text{accumulator, smallest red message time stamp so far}\)
- \(\text{CM\text{msg}\_Count}[1:N] = \# \text{messages sent to each LP}\)
Sketch of Algorithm

- **Message send by green logical process from LP\(_i\) to LP\(_j\)**
  \[ V_{i[j]} = V_{i[j]} + 1 \]

- **LP\(_i\) receives a green message**
  \[ V_{i[i]} = V_{i[i]} + 1 \]

- **Control message, first cut:**
  - Change color of process to red
  - \( CMsg\_Count = CMsg\_Count + V_i \)
  - Forward control message to next process in ring

- **Message send with time stamp ts by a red LP**
  \[ T_{red} = \min (T_{red}, ts) \]

- **Control message, Second cut:**
  - wait until \( (V_{i[i]} = CMsg\_Count[i]) \)
  - \( CMsg\_T_{min} = \min (CMsg\_T_{min}, T_{min}) \)
  - \( CMsg\_T_{red} = \min (CMsg\_T_{red}, T_{red}) \)
  - forward token to next process in ring
Distributing GVT Values & Pipelining

Distribute computed GVT to each LP
• use third cut

Pipelined execution
• Overlap successive GVT computations: first GVT uses C1, C2, C3, second uses C2, C3, C4, etc.
• Each cut computes a new GVT value
• Continuously circulate GVT token
Fossil Collection

• Batch fossil collection
  – After GVT computation, scan through list of LPs mapped to processor to reclaim memory and commit I/O operations
  – May be time consuming if many LPs

• On-the-fly Fossil Collection
  – After processing event, place memory into “free memory” list
  – Before allocating memory, check that time stamp is less than GVT before reusing memory
Summary

• Consistent cuts
• Cut value can be used as an estimate of GVT
  – Local minimum at each LP
  – Cut messages
• Construct second consistent cut
  – Coloring LPs, messages
  – Vector counter to determine when an LP has received all relevant cut messages
• Pipeline GVT computation, continuously circulating token
• Numerous variations
  – Could implement cuts with other communication topologies, e.g., butterfly
  – Other ways to deal with transient messages, e.g., global count and abort/retry mechanism for second cut, etc.