Consistent Cuts

- **Asynchronous**
  - Executes in background concurrent with time warp execution (does not require the simulation to "freeze" (i.e., block the LPs).
  - Avoids message acknowledgements
  - Approach: Based on techniques for creating distributed snapshots (consistent cut)

- Can some asynchronous algorithms compute exact GVT(t)?
- What about synchronous algorithms?

Consistent cut:

- Cut point:
- Cut message:
- Consistent cut:

Cut value:

Global Virtual Time

Wallclock time $T$ (GVT) during the execution of a Time Warp simulation is defined as the *minimum* time stamp among all unprocessed and partially processed messages and anti-messages in the system at wall-clock $T$.
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
Consistent Cuts

Cut value: minimum among (1) local minimum of each LP at its cut point and (2) time stamp of cut messages

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
Consistent Cuts

Cut value: minimum among (1) local minimum of each LP at its cut point and (2) time stamp of cut messages

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
Consistent Cuts

Cut value: minimum among (1) local minimum of each LP at its cut point and (2) time stamp of cut messages

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
Consistent Cuts

Cut value: minimum among (1) local minimum of each LP at its cut point and (2) time stamp of cut messages

**Observation:** Time stamp of a message sent after a cut point at wallclock time $T$ must be at least as large as the minimum of:
- the smallest time stamp of any unprocessed event in the processor at time $T$
- the smallest time stamp of any message received by the processor after time $T$.

**GVT must be smaller than or equal to both of these quantities.**
**Observation 1**

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

**Mattern’s GVT Algorithm**

 belang: accounting for cut messages

- 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 (green/red) each message to that of LP sending message (message tag)
- Maintain send/receive message counters
- GVT = \( \min(\text{local min along C2, time stamp of red messages}) \)

**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 global minimum (GVT value)

**Example: Vector Counters**

- LP\(_i\) maintains vector \( V_i[1:N] \), where \( N = \#\text{LPs} \)
  - \( V_i[1] \) = number of green messages received by LP\(_i\)
  - \( V_i[r] \) = number of green messages sent by LP\(_i\) to LP\(_r\)
- C2: LP\(_i\) cannot pass token until
  - \( V_i[\hat{i}] = \sum V_s[\hat{i}] \) (summed over all \( s \neq \hat{i} \))
- C1: Token includes vector to accumulate send counters

<table>
<thead>
<tr>
<th>( V_1 )</th>
<th>( V_2 )</th>
<th>( V_3 )</th>
<th>( V_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_1[4] = 0 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Vector Counters

Vector counters for green messages (at C2) $i = j$ received:

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
</tr>
</thead>
</table>

Example: Vector Counters

Vector counters for green messages (at C2) $i = j$ received:

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
</tr>
</thead>
</table>

Example: Vector Counters

Vector counters for green messages (at C2) $i = j$ received:

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
</tr>
</thead>
</table>

Example: Vector Counters

Vector counters for green messages (at C2) $i = j$ received:

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
</tr>
</thead>
</table>

Example: Vector Counters

Vector counters for green messages (at C2) $i = j$ received:

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
</tr>
</thead>
</table>

Example: Vector Counters

Vector counters for green messages (at C2) $i = j$ received:

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
</tr>
</thead>
</table>
Mattern’s GVT Algorithm

- **Local Variables (in each logical process LPi):**
  - $T_{\text{red}} = \min$ time stamp among red messages sent by LP (even non-cut red messages!)
  - $V_{[1:N]} = \#$ messages sent to each LP
- **Token:** $\text{CMsg}$
  - $\text{CMsg}_\text{min} = \text{accumulator, smallest local minimum so far}$
  - $\text{CMsg}_\text{red} = \text{accumulator, smallest red message time stamp so far}$
  - $\text{CMsg}_\text{Count}[1:N] = \#$ messages sent to each LP

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.

Distributing GVT Values & Pipelining

- **Pipelined execution**
  - **Overlap successive GVT computations:** first GVT uses $C_1, C_2, C_3$, second uses $C_2, C_3, C_4$, etc.
  - Each cut computes a new GVT value
  - Continuously circulate GVT token