CCC'97 Presentation

IMPLEMENTING HIGHER-ORDER CONCURRENCY IN PVM

Erik Demaine
Dept. of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
eddemaine@uwaterloo.ca

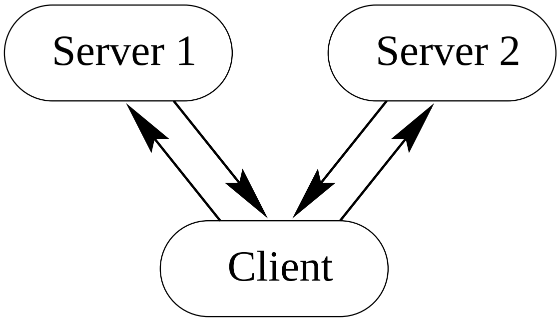
March 10, 1997

OUTLINE

- Higher-order concurrency
- Implementing in PVM
  - Channels
  - Events
- Results
- Conclusion

HIGHER-ORDER CONCURRENCY

- Reppy proposed concurrent-programming model called **higher-order concurrency**
- Flexible construction of high-level communication constructs
- Example:



  - Define a call to the server (including protocol, etc.)
  - Don't want to recode for client to simultaneously talk to multiple servers
  - Especially if servers are of different types
  - First-class allows combination of protocols without recoding

DETAILS

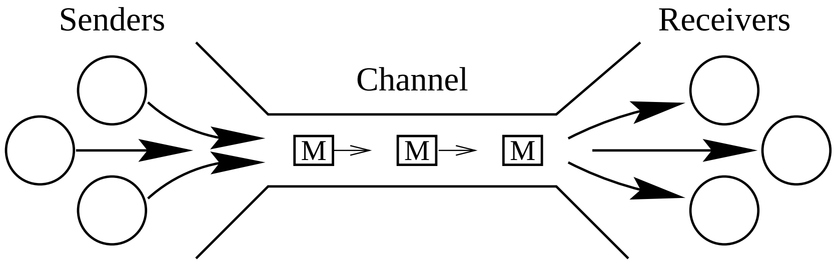
- Distributed memory (extension to Hoare's CSP)
- Channels: unidirectional "pipe"
  - Can pass channels over channels
- **Events**: synchronous communications on channels
  - Primitive events:
    - ☐ **transmit** (chan, data)
    - ☐ **receive** (chan)
  - Execute events via **sync** operation
  - Combinators:
    - ☐ **choose** (e_1, ..., e_n): selection between events
    - ☐ **wrap**, **wrapAbort**, **guard**: sequence of events
- Processes are dynamically created and garbage collected
- Features are flexible; few needed

ADVANTAGES

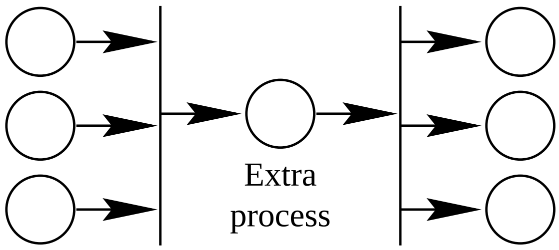
- Abstraction into **first-class events**
  - Define complex operations by functions returning events
  - Can treat new operations as if they are built-in
- Can implement:
  - Buffered channels (asynchronous communication)
  - Multicast channels and thus multicast operations
  - Condition variables (write-once variables)
  - Ada-style rendezvous, plus allowing nested transactions
  - Locks and semaphores
  - Multilisp futures
- Even further: can selectively choose between Ada entry calls or obtaining locks, etc.
- Concurrent-programming language doesn't have to "guess" what concurrency structures might be needed

IMPLEMENTING CHANNELS IN PVM

- In higher-order concurrency, channels are many-to-many



- Difficult to maintain group membership in distributed system
- Instead: many-to-one and one-to-many channels
  - **Owner**: is the single receiver/sender
  - **Members**: other processes connected to channel
- Can build up many-to-many channels using extra process



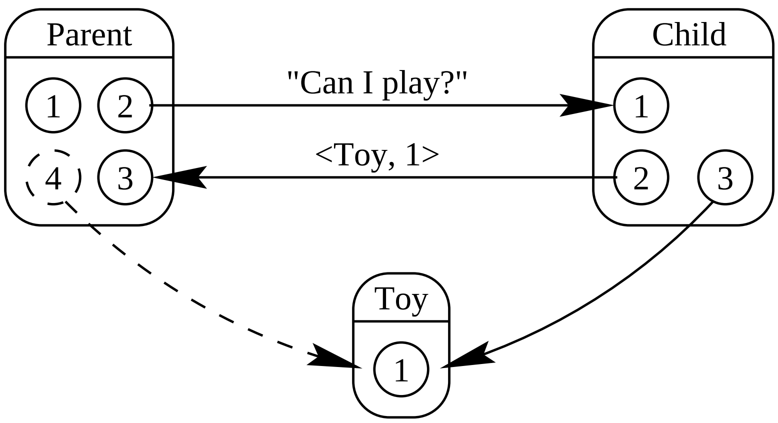
  - With higher-order concurrency, can treat these channels as built-in

REPRESENTING CHANNELS

- Representation: $\langle \mathsf{owner}, \mathsf{id} \rangle$ pair
  - id is local to owner
  - Sequence number per member
  - Owner maintains queue of messages that may travel through channel
- Key feature: channels can be passed in messages

CREATING CHANNELS

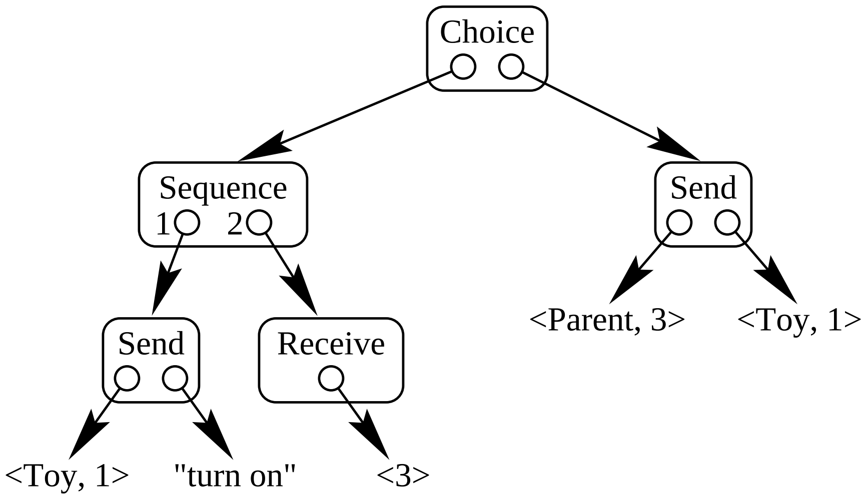
- In higher-order concurrency, define channel in scope accessible by several processes
  - Doesn't make sense in distributed (loosely-coupled) system
- Instead define ways to create channels and connect processes to them:
  - Any process p can dynamically create channel with owner p, either many-to-one or one-to-many
  - When spawning, obtain two channels between self and child
  - Other channel creation is done via message passing
- Example:



REPRESENTING EVENTS

- Event $\longleftrightarrow$ tree
  - Internal nodes correspond to combinators
  - Leaves correspond to basic events
  - External nodes correspond to channels or data items
  - Children must be ordered in case of sequence

- Example:



RESULTS

- Evaluate cost of abstraction
- Ping-pong benchmark on single processor

	Indirect		Direct	
Test	Abs.	Rel.	Abs.	Rel.
Full abstraction	8.4029	4.7207	6.2711	3.6624
Event re-use	8.2087	4.6116	6.1405	3.5861
No `malloc`	7.8974	4.4367	5.9473	3.4733
PVM	1.7800	1.0000	1.7123	1.0000

- Current negotation protocol uses 4 messages per synchronous communication
  - Can be improved using alternative protocols
- 11% overhead from event creation
- 17% overhead from traversing event tree
- Can be improved by avoiding dynamic allocation
- Absolute values ($\approx$ 200 μs) are small

CONCLUSION

- Main features of higher-order concurrency implemented in PVM:
  - First-class events
  - Garbage collection of processes
- Bringing ideas from one domain (programming languages) to another (message passing/parallel computing)
- Powerful concurrent programming can be achieved in existing message-passing systems
- Future
  - TPVM: mixed threads/processes version
  - Implementation of other protocols