Co-ops: Concurrent Algorithmic Skeletons for Erlang

Jay Nelson (@duomark)

DuoMark International, Inc.

Problem

The number of cores will soon exceed the number of processes in a typical erlang application.

Hypothesis

New libraries are needed which encourage the use of more processes. The number of processes should be closer to the number of functions than the number of modules. Existing OTP techniques are not easily and transparently mapped to enough hidden processes.

Issues

1. Code organization
   a. one source file per process is unmanageable
   b. related source code should be co-located
   c. communicating processes should be apparent in code

2. Process management
   a. explicit init/create is tedious at scale
   b. clusters of related processes instead of single processes
   c. desire to spawn/kill many processes at a time
   d. need implicit load-balancing across process clusters
   e. lattices of processes preferred over name registries
   f. visualization, tracing and debugging should be possible
   g. managing data flow vs. managing function execution

3. Application structure
   a. failure propagation should not default to total annihilation
   b. component services should dynamically adapt to demand

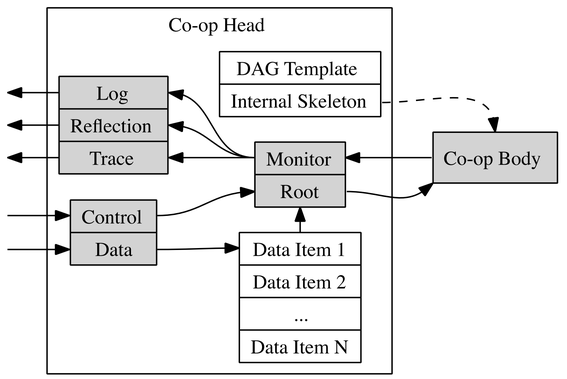
Approach

Directed Acyclic Graphs (DAGs) define an application component:

- Co-op Head is the external conduit to the DAG
- Co-op Body is the implementation of the DAG
- Kill switch process is linked to all internal processes
- Components are assembled into applications

Co-op Head:

- DAG has single logical root node for ingress of message data
- Two channels with control prioritized over data flow
- Tracing, logging and graphical display are out-of-band (pids in gray)
- Replicate with implicit load-balancing for increased throughput



Co-op Body:

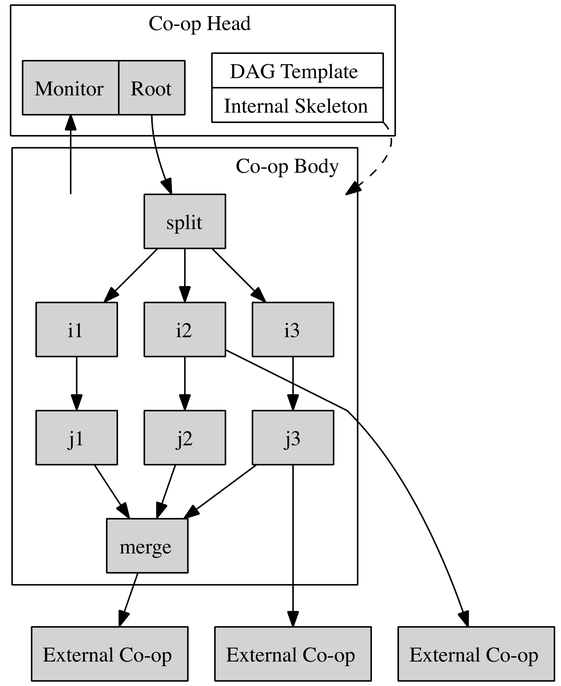
- Each node describes a process with a specific task
- Nodes are labeled with function implementing task
- Each edge represents one-way messaging between 2 processes
- Utilities provide fan out, fan in, pipeline topologies

Circularity is introduced external to DAGs:

- Nodes may optionally reference DAG head
- Output from a DAG may flow to any other DAG
- External processes may communicate to and from DAGs

Co-ops may be managed independently

- Suspend/resume Co-op Root to throttle message flow
- Code upgrade clones Co-op rather than killing old code



Conclusion

Programming with Co-ops provides a direct mapping from algorithm concept to application structure. Logical units are easier to manage and graphical display can mirror live graph execution and data flow. A single graph can hold thousands of processes, can be replicated in entirety and linked with other graphs for larger computations.

Proof-of-concept

https://github.com/duomark/erlangsp