Dynamic Task Speculation Support Through Divide-and-Merge Memory Allocation

Chen Ding, Benjamin O'Halloran, Jacob Bisnett, Joel Kottas and Colin Pronovost

Outline

- Standard memory management / malloc
- Speculative memory management
- Givy Allocator & System
- Divide & Merge (DM) Malloc
- Use cases

Standard Malloc

- Modern allocators use size classes / free lists (FIFO stack)
- block: memory region used for program data
- malloc: get data block
- free: program finished with block
  - Can be a nop
- Thread local reserves

Speculative Memory Management

- Tasks need to be isolated
- Custom malloc required to avoid false conflicts
- Partition GAS into per-task specific reserves
  - Either divide free lists or the VAS itself
- Communication generally requires either:
  - Expensive IPC protocol
  - Pre-allocated (finite) shared memory
  - File-backed shared memory
    - Security concerns

Givy

- ISMM 2016 by Gindraud et. al. for CnC / distributed embedded systems
- Givy executes dynamic task graphs
- Raw C pointers define reference
- Interval of VM addresses given to each node in system
  - Allocations require no network communication
- Eager free

DM Malloc

- Part of process-based speculative parallelization framework
  - Page (4KB) granularity will waste physical memory
- When speculation begins divide the virtual address space
  - Noncontiguous, finite regions
  - Divide free lists of each size class between the number of speculative tasks
  - No communication between tasks

DM Malloc

- Once committed, merge the free lists
  - Only the top of each node’s region will have been modified
- Deferred Free
  - Needed to eliminate communication, there is no shared memory used by allocator

Comparison

DM Malloc

- Finite memory / no heap growth
  - Speculative, so abort when OOM
- Design generalizes for distributed environment
  - Current framework is not distributed

Givy Allocator

- Solves distributed allocation problem
- No hard limit for per-node allocations
- Non speculative

Use Cases

- Redundant Execution
  - Faulty or Asymmetric hardware
- Simplify implementation of fully decentralized design
  - Consensus through majority vote, re-try minority voters
- "undo" and "redo" operations that may expand the programming interface of CnC.
  - Unsure of parallelism or the exact dependence between tasks
  - Check and enforce parallelism or dependence after a task is completed.