Enabling High-Level Graph Processing via Dynamic Tasking

Maurizio Drocco, Vito Giovanni Castellana, Marco Minutoli, Antonino Tumeo, John Feo

Motivation
- Data-intensive computing induces irregular and unbalanced workloads
- e.g., graph processing yields fine-grained unpredictable access patterns
- Large-scale problems require distributed systems (e.g., multi-core clusters)
- Managing dynamic workloads while dealing with load imbalance on HPC clusters is an open problem

Challenge
- Several task-oriented runtime systems chase high performance by multithreading and asynchronous coordination
- Queueing systems play a key role to provide efficient dispatching of tasks, both within and across nodes
- The distribution of tasks across the nodes varies broadly over time, triggering the pitfalls of statically managed resources in the system
- The goal: designing strategies for task queueing (on-node) and spawning (across nodes) to deal with dynamic and unbalanced workloads

Approach

**Common Configuration**
- A reservation protocol is used to allocate remote execution slots, avoiding overflows at the price of extra communication traffic.

**Dynamic Schema**
- Tasks are spawned and possibly buffered only at the executor side, relying on backpressure feedback and sensing to avoid overflows.

**On-node Allocation**
- Common Configuration
  - A single fixed-size MPMC pool serves as centralized task allocator, suffering from high MPMC contention.
- Dynamic Schema
  - In our decentralized schema, tasks are allocated by each worker from dynamic private pools and recycled through one-to-one, fast, cache-friendly SPSC recycle bins.

Evaluation

- Graph triangle counting on a 16-node cluster of 20-core Intel(R) Xeon(R) E5-2680 v2 @2.80GHz

**Configurations**
- Static: high-level SHAD + GMT backend
- Dynamic: high-level SHAD + GMT backend, with the novel dynamic on-node allocation and remote execution schemas
- Baseline: low-level custom GMT implementation

Strong Scaling (dynamic)
- Average 1.5x speedup vs static GMT
- Less memory waste
- Less configuration parameters

About GMT
- Global Memory and Threading
- Originally designed for Partitioned Global Address Space (PGAS)
- API: fork/join control model with constructs for task and data parallelism
- Key performance aspects: fine-grained multithreading combined with data aggregation for latency tolerance

Implementation
- Three types of threads on each GMT node: communication servers for MPI networking, workers for task execution, and helpers for serving remote tasking
- Tasking system based on multiple task queues shared by worker threads + an off-node reservation/execution protocol

About SHAD
- Scalable high-performance data-structures
- Inspired by C++17 STL
- Supports applications on commodity HPC clusters with a NUMA abstraction
- Modular design with multiple backends: plain C++, Intel TBB, GMT

Contributions
- A novel task allocator providing optimal on-node memory usage with no need for configuration tuning
- Elastic off-node spawning based on executor-side buffering and backpressure sensing
- Proof-of-concept implementation within the GMT tasking system
- Performance evaluation on high-level graph processing with SHAD

ABOUT SHAD

 Scalable High-Performance Data-Structures
- The PNNL library of HPC containers and algorithms
- Inspired by C++17 STL
- Supports applications on commodity HPC clusters with a NUMA abstraction
- Modular design with multiple backends: plain C++, Intel TBB, GMT

github.com/pnnl/SHAD

www.pnnl.gov