TORSTEN HOEFLER

MPI Beyond 3.0
and Towards Larger-Scale Computing
MPI-3.0 Overview

Nonblocking Collectives

```c
MPI_Ibcast(..., req);
for(i=0; i<iters; ++i) {
    ...
    MPI_Test(&req);
}
...
MPI_Wait(&req);
```

Neighborhood Collectives

Noncollective Comm Creation

```c
MPI_Comm_create_group()
```

Software Engineering 😊

- Fortran
- Tools Interface
- many more …

and …
MPI-3.0 Remote Memory Access

Synchronization Modes
- Fence
- Lock
- PSCW

Data Movement Operations
- Put
- Get

New Window Creation

New Accumulate Operations

MPI-3.0 Shared Memory

New Memory Models
- Private copy
- Load, Store
- Public copy
- Put, Get
- Sync
- Process
- Private/public copy
- Store, Load
Larger-scale (Exascale)?

**Point-to-point Scalability**
- Scalable groups and communicators
- Limited buffering

**Collective Scalability**
- Scalable interfaces
- Only *v collectives are non-scalable

---

**Topology Mapping**
- Scalable graph topology interface
- MPI-2.2

**RMA Scalability**
- Scalable interfaces
- RMA protocols in [1]

---

[1]: Gerstenberger et al.: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided (SC13)
HPC Today and Towards Data-driven Problems
A brave new (data-driven) world!

**Tiny Computations**

```java
vertex_handler(vert v, dist d) {
  if(d < v.dist) {
    v.dist = d;
  }
}
```

**Lack of Structure**

```java
bfs_walker(vert v) {
  for(vert u in v.neighbors) {
    bfs_walker(u);
  }
}
```

**Dominated by Memory**

**Poor Locality**

**Poor Load-Balancing**
Can we use our old tools?

Structured Problems
- Regular data dependencies
- Static control flow
- Balanced load
- Simple messages
- Communicates data
- Implemented with Affine loops, Matlab

Data-driven Problems
- Irregular, fine-grained dependencies
- Dynamic, data-dependent control flow
- Irregular load
- Communicates data and control flow
- Implemented with Pregel, GraphLab, MPI, OpenMP parallel for
- ...

Data-driven Problems have very different requirements! We need to reconsider Programming, Runtime System, and Architecture
Our Proposal - Active Pebbles

... A Programming and Execution Model for Data-driven computations ...

Programmers Specify High-Level Algorithms

Leave the Execution Details to Runtime

Willcock, Hoefler, Edmonds, Lumsdaine: “Active Pebbles: Parallel Programming for Data-Driven Applications”, PPoPP’11
The Programming Level

Data-Flow Programming

```cpp
vertex_handler(vert v, dist d) {
    if(d < v.dist) {
        v.dist = d;
    }
    for(vert u in v.neighbors) {
        vertex_handler(u);
    }
}
```

Properties of Data-Flow Programming

- Easy to develop (textbook)
- Intuitive correctness analysis
- Label-setting vs. label-correcting is simple matter of synchronization!
- Automated termination detection
- …
Execution Model – The Magic

- Active messages are the basis

- Plus a bag of synergistic tricks
  - Message Coalescing
  - Active Routing
  - Message Reductions
  - Termination detection
Message Coalescing

- Injection rate may be a limiting factor
- Message coalescing trades latency for injection rate
Active Routing

- Coalescing buffers limit scalability
- Impose a limited topology with fewer neighbors
  - Trades latency for memory scalability and congestion control
  - Needs to align with underlying network routing
  - Cf. optimized alltoall algorithms
Message Reductions

- Combine messages to same target (assumes associative op)
  - Uses caching strategies
- Routing allows reductions at intermediate hops
4) Termination Detection

- When does the algorithm terminate?
  - When no messages are in flight and no handler runs

- Standard algorithms: $\Theta(\log P)$

- Limited-depth termination detection [1]: $\Theta(\log k)$

- Epoch model
  - Label setting: wait for TD at each level
  - Label correcting: never wait for TD
  - Depth-$k$ algorithms: wait after $k$ levels (e.g., Delta Stepping)

[1]: Hoefler, Siebert, Lumsdaine: Scalable Communication Protocols for Dynamic Sparse Data Exchange (PPoPP’10)
Some Early Performance Results

**Breadth First Search**
(2^{19} vertices per process)

**Random Access**
(2^{19} vertices per process)

Test system: 128 Nodes, 2x2GHz Opteron 270 CPUs, InfiniBand SDR, OpenMPI 1.4.1
Lessons Learned (so far)

- Data-driven executions can rely on fine-grained AMs
- Needs some tricks to make it fast:
  - Message Coalescing
  - Active Routing
  - Message Reductions
  - Termination detection

- Issues:
  - Message Passing is slow
  - Simple PGAS is not sufficient (buffering issues)
  - Handlers need to be linearizable (execute atomically)

- Fixes?
  - Redesign the network to support data-driven computations
Does that belong in MPI?

- AP can be (is) implemented as a DSL over MPI
  - Is this efficient?

- MPI two-sided imposes some unnecessary constraints:
  - In order matching
  - User-managed receive buffers
  - Interoperation with threads is complex (locking or thread_multiple issues)
  - Control transfer?

- MPI one-sided imposes other constraints:
  - Sender-managed remote buffers (ugs)
  - Control transfer?

- What would I want?
  - Active messages 😊 - also discussed in [1]

[1]: Zhao et al. “Towards Asynchronous and MPI-Interoperable Active Messages (CCGrid 2013)