The Parallel Boost Graph Library
spawn(Active Pebbles)

Nicholas Edmonds and Andrew Lumsdaine
Center for Research in Extreme Scale Technologies
Indiana University
Origins

- Boost Graph Library (1999)
  - Generic programming with templates
  - Good sequential performance with high-level abstractions
  - Algorithm composition, visitors, property maps, etc.

Origins

- Parallel BGL (2005)
  - Lifting abstraction
  - Make what we already have parallel
  - Same interfaces as sequential BGL
  - Coarse-grained, BSP


Case Study: Breadth-First Search

ENQUEUE(Q, s)
while (Q ≠ ∅)
    u ← DEQUEUE(Q)
    for (each v ∈ Adj[u])
        if (color[v] = WHITE)
            color[v] ← GRAY
            ENQUEUE(Q, v)
        else color[u] ← BLACK
SPMD Breadth-First Search

Process 0

Process 1

Process 2

Q.push(d)
SPMD Breadth-First Search

Rank 0
Get neighbors

Rank 1
Redistribute queues
Combine received queues

Rank 2

Rank 3
Results were run on Erdős-Renyi graphs using a cluster of 128 2.0Ghz Opteron 270 processors with four cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.
SPMD Breadth-First Search (Weak Scaling)

Results were run on Erdős-Rényi graphs using a cluster of 128 2.0GHz Opteron 270 processors with four cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.
ENQUEUE(Q, s)
while (Q ≠ ∅)
    u ← DEQUEUE(Q)
    for (each v ∈ Adj[u])
        if (color[v] = WHITE)
            color[v] ← GRAY
            ENQUEUE(Q, v)
    else color[u] ← BLACK
Find the Synchronization Trap

ENQUEUE\((Q, s)\)

while \((Q \neq \emptyset)\)

\(u \leftarrow \text{DEQUEUE}(Q)\)

for (each \(v \in \text{Adj}[u]\))

if \((\text{color}[v] = \text{WHITE})\)

\(\text{color}[v] \leftarrow \text{GRAY}\)

ENQUEUE\((Q, v)\)

else \(\text{color}[u] \leftarrow \text{BLACK}\)

for \(i\) in ranks: start receiving \(\text{in\_queue}[i]\) from rank \(i\)

for \(j\) in ranks: start sending \(\text{out\_queue}[j]\) to rank \(j\)

synchronize and finish communications
What’s Wrong Here?

Traditional Scientific Apps
- Coarse-grained
- Static
- Balanced
- Bandwidth-bound with large messages
- Communicates data

Data-driven Apps
- Fine-grained dependencies
- Dynamic, data-dependent
- Irregular
- Latency-bound with small messages
- Communicates control flow

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington
Messaging Models

- **Two-sided**
  - MPI
  - Explicit sends and receives

- **One-sided**
  - MPI-2 one-sided, ARMCI, PGAS languages
  - Remote put and get operations
  - Limited set of atomic updates into remote memory

- **Active messages**
  - GASNet, DCMF, LAPI, Charm++, X10, etc.
  - Explicit sends, implicit receives
  - User-defined handler called on receiver for each message
Active Messages

• Created by von Eicken et al, for Split-C (1992)
• Messages sent explicitly
• Receivers register handlers but not involved with individual messages
• Messages often asynchronous for higher throughput
Active Messages

- Move control flow to data
- Fine-grained
- Asynchronous
- Uniformity of access

**AM++**

- Generic
- User-level
- Flexible/modular
- Send to *targets*, not processors
Low-Level vs. High-Level AM Systems

• Active messaging systems (loosely) on a spectrum of features vs. performance
  • Low-level systems typically have restrictions on message handler behavior, explicit buffer management, etc.
  • High-level systems often provide dynamic load balancing, service discovery, authentication/security, etc.
The AM++ Framework

- AM++ provides a “middle ground” between low- and high-level systems
  - Gets performance from low-level systems
  - Gets programmability from high-level systems
- High-level features can be built on top of AM++

Active Pebbles

Low

DCMF  GASNet

AM++

Charm++/X10  Java RMI

High
Messages Can Send Messages

- Termination detection
  - Detect network quiescence
  - Pluggable
Active Pebbles

- Need to separate what the programmer expresses from what is actually executed
- A programming model and an execution model
Active Pebbles Features

- Programming model
  - Active messages (*pebbles*)
  - Fine-grained addressing (*targets*)

- Execution model
  - Flexible message coalescing
  - Message reductions
  - Active routing
  - Termination detection

- Features are synergistic
- AM++ is our implementation of Active Pebbles model
Programming Model

• Program with natural granularity
  • No need to artificially coarsen object granularity

• Transparent addressing
  • static and dynamic
  • local and remote

• Bulk, anonymous handling of messages and targets

• Epoch model
  • Enforce message delivery
  • Controlled relaxed consistency
Execution Model

- Message coalescing
  - Amortize latency
- Message reduction
  - Eliminate redundant computation
  - Distributed computation into network
- Active routing
  - Exploit physical topology
- Termination detection
  - Handlers send messages
  - Detect quiescence
Routing + Message Coalescing

- Coalescing buffers limit scalability
  - Communications typically all-to-all
- Impose a limited topology with fewer neighbors
- Better scalability, higher latency

Multi-source coalescing
Message Reductions + Routing

- Messages to same target can often be combined
- Reductions application-specific, user-defined
- Routing allows cache hits at intermediate hops

Automatically synthesize optimized collectives

Distribute computation into the network
Active Message Abstraction

- *Pebbles* are agnostic as to where they execute, operate on *targets*
- Independent of how messages are processed
  - Network communication (MPI, GASNet, DCMF, IB Verbs…)
  - Work stealing (Cilk++, Task parallelism)
  - OpenMP (over coalescing buffers)
  - Immediate execution in caller (of send())

- Thread-safe metadata allows weakening message order
  - Updates to *targets* must be atomic
  - Algorithms may have to tolerate weak consistency
Evaluation: Message Latency

![Graph showing message latency vs. message size for different network abstractions (GASNet/IBV, GASNet/MPI, AM++)]
Evaluation: Message Bandwidth

![Graph showing evaluation of message bandwidth with different sizes and protocols. The graph plots bandwidth in MB/s against message size. The x-axis represents message size ranging from 1024 to 65536, and the y-axis represents bandwidth in MB/s ranging from 0 to 800. Three protocols are compared: GASNet/IBV (red line), GASNet/MPI (green dotted line), and AM++ (blue dotted line). Each line shows different performance trends as the message size increases.]
Active Pebbles

• Meant to support all kinds of parallelism

• Started with optimizing distributed memory communication

• Same features allow integration of fine-grained parallelism
Active Messages for Work Decomposition

- Key idea is to find natural granularity
- Each *pebble* represents an independent computation that can be executed in parallel

```
ENQUEUE(Q, s)
while (Q ≠ ∅)
    u ← DEQUEUE(Q)
    for (each v ∈ Adj[u])
        if (color[v] = WHITE)
            color[v] ← GRAY
            ENQUEUE(Q, v)
        else
            color[u] ← BLACK
```

#pragma omp parallel for
```
for (each v ∈ Q)
    queue_push_msg → send()
```

“Inner-Loop” Parallelism
What’s Thread-safe in AP?

• Messaging  
• Epoch begin/end  
• Termination detection

• **NOT:**  
  • Message creation  
  • Modifying message features: routing, coalescing, reductions  
  • Modifying termination detection  
  • Modifying the number of threads
Parallel BGL Architecture

- AP makes messaging thread-safe.
- Property maps make metadata manipulation thread-safe and allow messages to be processed in arbitrary order
- Just as transactional memory generalizes processor atomics to arbitrary transactions, Active Pebbles generalizes one-sided operations to user-defined handlers
Active Pebbles Breadth-First Search

Process 0

Process 1

Process 2

explore(h, 2)

explore(h, 3)
BFS: Strong Scaling

Breadth-First Search ($2^{27}$ vertices $2^{29}$ edges)

Results were run on Erdős-Rényi graphs using a cluster of 128 2.0Ghz Opteron 270 processors with two cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.
BFS: Weak Scaling

Breadth-First Search (\(2^{25}\) vertices \(2^{27}\) edges per node)

Results were run on Erdős-Rényi graphs using a cluster of 128 2.0Ghz Opteron 270 processors with two cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.
Delta-Stepping: Strong Scaling

Delta-Stepping SSSP ($2^{27}$ vertices $2^{29}$ edges)

Results were run on Erdős-Rényi graphs using a cluster of 128 2.0Ghz Opteron 270 processors with two cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.
Delta-Stepping: Weak Scaling

Delta-Stepping SSSP ($2^{24}$ vertices $2^{26}$ edges per node)

Results were run on Erdős-Renyi graphs using a cluster of 128 2.0Ghz Opteron 270 processors with two cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.
Transactional Metadata updates

- Sometimes we want to update non-contiguous, dependent data atomically
- Predecessor and BFS level
  - Or arbitrary visitor code supplied by users
- Limited-scope transactions, are these any easier?

\[
\text{std::atomic<std::pair<T1, T2> > !?!
}\]

\[
\text{template <typename T> struct Foo;
std::atomic<Foo<T> > !!??!
}\]
Summary

• Active Messages
  • Express fine-grained, asynchronous operations elegantly
  • Well-matched to data-driven problems
  • Enable fine-grained parallelism (threads, GPUs, FPGAs, …)
  • Asynchrony allows latency hiding

• Concise expression and efficient execution
  • Separate programming and execution models

• Active Pebbles
  • Simple programming model
  • Execution model maps programs to hardware efficiently
    • AM++ is our implementation
    • Could be targeted as a runtime by languages (X10, Chapel, …)
Future Work

- Constrained parallelism in shared memory
  - Not entirely work queue-based
  - Not entirely recursion-based
- Acceleration
  - Coalesced message buffers offer great opportunities here… if getting to the accelerator is cheap
- Optimizing local memory transactions
  - Intel’s TSX in *Haswell* CPUs
  - Declarative language for transaction code generation
Questions?

- More info on Active Pebbles
  - Jeremiah Willcock, Torsten Hoefler, Nicholas Edmonds, and Andrew Lumsdaine. Active Pebbles: Parallel Programming for Data-Driven Applications. ICS '11.

- More info on AM++

- More info on the Parallel Boost Graph Library and graph applications:
  - http://www.osl.iu.edu/research/pbgl
  - http://www.boost.org
  - Watch for a new release of PBGL based on Active Pebbles, running on AM++ soon!

(Ask if you’d like access to a pre-release, very alpha copy)

ngedmond@cs.indiana.edu