A Resilient Framework for Iterative Linear Algebra Applications in X10

Sara S. Hamouda
Australian National University

Josh Milthorpe
IBM T.J. Watson Research Center

Peter E. Strazdins
Australian National University

Vijay Saraswat
IBM T.J. Watson Research Center

2015 ACM SIGPLAN
X10 Workshop at PLDI
Programmability vs. Resilience

- **Global Memory View**
  - Async Task Parallelism
    - X10, Chapel
  - SPMD
    - UPC, Titanium, CAF

- **Local Memory View**
  - SPMD, Actor
    - MPI, Charm++, Erlang
Resilient X10

Efficient failure-aware programming

David Cunningham\(^2\) *, David Grove\(^1\), Benjamin Herta\(^1\), Arun Iyengar\(^1\), Kiyokuni Kawachiya\(^3\), Hiroki Murata\(^3\), Vijay Saraswat\(^1\), Mikio Takeuchi\(^3\), Olivier Tardieu\(^1\)

\(^1\)IBM T. J. Watson Research Center
\(^2\)Google Inc.
\(^3\)IBM Research - Tokyo

dcunnin@google.com, \{groved, bherta, aruni, vsaraswa, tardieu\}@us.ibm.com, \{kawatiya, mrtlrk, mtake\}@jp.ibm.com
X10 Domain Specific Libraries

- GML (Global Matrix Library)
- ANUChem
- ScaleGraph
- M3RLite (Main Memory Map Reduce Lite)
- Megaffic (Traffic flow simulation)
- SatX10 (Parallel boolean satisfiability)
X10 Domain Specific Libraries

- **Resilient** GML (Global Matrix Library)
- ANUChem
- ScaleGraph
- M3RLite (Main Memory Map Reduce Lite)
- Megaffic (Traffic flow simulation)
- SatX10 (Parallel boolean satisfiability)
Outline

• Resilient X10
• GML
  – API Overview
  – Resilience Limitations
  – Resilience Enhancements
  – Performance Results
// Task A
try {
    at(p) {
        // Task B
        finish {
            at(q) async {
                // Task C
            }
        }
    }
} catch (dpe: DeadPlaceException) {
    // recovery step
}
// D
Resilient X10

// Task A
try {
  at(p) {
    // Task B
    finish {
      at(q) async {
        // Task C
      }
    }
  }
} catch (dpe: DeadPlaceException) {
  // recovery step
}
// D
Resilient X10

```
// Task A
try {
    at(p) { // Task B
        finish {
            at(q) async {
                // Task C
            }
        }
    }
} catch (dpe: DeadPlaceException) {
    // recovery step
}
// D
```

- Resilient X10 supports only the **sockets** backend
- Resilient Store
  - Centralized Store
  - Distributed Store (currently not supported)
Outline

• Resilient X10

• GML
  – API Overview
  – Resilience Limitations
  – Resilience Enhancements
  – Performance Results
Global Matrix Library (GML)

- Distributed matrix library in X10
- Simple programming model
  - Matrix based
  - Sequential style programming
  - Efficient iterative processing
- Potential compilation target for high-level array languages
  - Provides fundamental vector/matrix routines
  - Supports dense and sparse matrix formats
  - Uses BLAS and LAPACK
## GML Vector/Matrix Classes

<table>
<thead>
<tr>
<th>Single Place</th>
<th>Multi-Place</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Duplicated</td>
<td>1 Block/Place</td>
</tr>
<tr>
<td>DenseMatrix</td>
<td>DupDenseMatrix</td>
<td>DistDenseMatrix</td>
</tr>
<tr>
<td>SymDense</td>
<td>DupSparseMatrix</td>
<td>DistSparseMatrix</td>
</tr>
<tr>
<td>TriDense</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SparseCSC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SparseCSR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector</td>
<td>DupVector</td>
<td>DistVector</td>
</tr>
</tbody>
</table>
PageRank Implementation in GML

/* Matrix dimensions */
var m:Long, n:Long;

/* Matrix partitioning configurations */

/* Create GML objects */
val G:DistBlockMatrix = DistBlockMatrix.make(m, n, rowBlocks, colBlocks, rowPlaces, colPlaces);
val P:DupVector = DupVector.make(n);
val U:DistVector = DistVector.make(n, G.getAggRowBs());
val GP:DistVector = DistVector.make(n, G.getAggRowBs());

/* Data initialization code omitted */

Algorithm:
for (1..k)
P = α G P + (1 − α) E U^T P
PageRank Implementation in GML

/* Data initialization code omitted */

for (1..k) {
    GP.mult(G, P).scale(alpha);
    val UtP1a = U.dot(P) * (1-alpha);
    GP.copyTo(P.local());
    P.local().cellAdd(UtP1a);
    P.sync();
}

Algorithm:

for (1..k)

\[ P = \alpha G P + (1 - \alpha) E U^T P \]
Outline

- Resilient X10
- GML
  - API Overview
  - Resilience Limitations
  - Resilience Enhancements
  - Performance Results
GML Resilience Limitations

- Fixed place distribution

- Failure of a place resulted in loss of GML objects
  - no built-in mechanism for restoring objects
Resilience Enhancements (1)

- Arbitrary and dynamic place distribution
  - make(..., places: PlaceGroup)
  - remake(..., newPlaces: PlaceGroup)
DistVector Redistribution

```java
val pg = make_P0_P2_group();

A.remake(pg);
```

<table>
<thead>
<tr>
<th>Before remake</th>
<th>After remake</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Place 0</strong></td>
<td><strong>Place 0</strong></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td><strong>Place 1</strong></td>
<td><strong>Place 1</strong></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td><strong>Place 2</strong></td>
<td><strong>Place 2</strong></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
Resilience Enhancements (2)

• Added in-memory snapshot / restore capability to GML classes

```java
interface Snapshottable {
    makeSnapshot(): Snapshot;
    restoreSnapshot(Snapshot): void;
}
```
val A = DistVector.make(6);
A.init((i:Long)=> i*2.0);
val A = DistVector.make(6);
A.init((i:Long) => i * 2.0);

val snap = A.makeSnapshot();
val A = DistVector.make(6);
A.init((i:Long)=> i*2.0);

val snap = A.makeSnapshot();

DistVector Snapshot/Restore

<table>
<thead>
<tr>
<th>Place 0</th>
<th>Place 1</th>
<th>Place 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>value</td>
<td>key</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

A PlaceLocalHandle

---

Copy from Snapshot
val A = DistVector.make(6);
A.init((i:Long)=> i*2.0);

val snap = A.makeSnapshot();

/* Place 1 failed */
val A = DistVector.make(6);
A.init((i:Long)=> i*2.0);

val snap = A.makeSnapshot();

/* Place 1 failed */

val pg = make_P0_P2_group();
A.remake(pg);
val A = DistVector.make(6);
A.init((i:Long)=> i*2.0);

val snap = A.makeSnapshot();

/* Place 1 failed */

val pg = make_P0_P2_group();

A.remake(pg);

A.restoreSnapshot(snap);
interface ResilientIterativeApp {
    def step(): void;
    def isFinished(): void;
    def checkpoint(store: AppResilientStore): void;
    def restore(newPlaces: PlaceGroup,
                 store: AppResilientStore,
                 snapshotIter: Long): void;
}
val store:AppResilientStore;
while (!isFinished()) {
    try {
        if (restoreRequired) {
            val newPlaces = createRestorePlaceGroup();
            restore(newPlaces, store, checkpointIter);
        }
        step();
        if (iter % checkpointInterval == 0) {
            checkpoint(store);
            checkpointIter = iter;
        }
        iter++;
    } catch (dpe:DeadPlaceException) {
        restoreRequired = true;
    }
}
(3) Application Resilient Store

- Concurrent and atomic snapshot/restore for multiple GML objects

```python
class AppResilientStore {
    def startNewSnapshot();
    def save(obj: Snapshottable);
    def saveReadOnly(obj: Snapshottable);
    def commit();
    def cancelSnapshot();
    def restore();
}
```
def checkpoint(store: AppResilientStore){
    store.startNewSnapshot();
    store.saveReadOnly(G);
    store.saveReadOnly(U);
    store.save(P);
    store.commit();
}

def restore(newPlaces: PlaceGroup, store: AppResilientStore, snapshotIter: Long){
    G.remake(..., newPG);
    U.remake(..., newPG);
    P.remake(newPG);
    store.restore();
    //restore other primitive variables
}
(4) Restore Modes

- Restoration Modes
  - Shrink
  - Shrink-Rebalance
  - Replace Redundant
Shrink

Before remake

<table>
<thead>
<tr>
<th>Place 0</th>
<th>Place 1</th>
<th>Place 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b0</td>
<td>b1</td>
<td>b2</td>
</tr>
<tr>
<td>b3</td>
<td>b4</td>
<td>b5</td>
</tr>
</tbody>
</table>

After remake

<table>
<thead>
<tr>
<th>Place 0</th>
<th>Place 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b0`</td>
<td>b2`</td>
</tr>
<tr>
<td>b3`</td>
<td>b5`</td>
</tr>
</tbody>
</table>

b0, b1, b2, b3, b4, b5
Shrink Rebalance

**Before remake**

<table>
<thead>
<tr>
<th>Place 0</th>
<th>Place 1</th>
<th>Place 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Diagram of Place 0]</td>
<td>![Diagram of Place 1]</td>
<td>![Diagram of Place 2]</td>
</tr>
</tbody>
</table>

**After remake**

<table>
<thead>
<tr>
<th>Place 0</th>
<th>Place 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Diagram of Place 0']</td>
<td>![Diagram of Place 2']</td>
</tr>
</tbody>
</table>
Outline

• Resilient X10
• GML
  – API Overview
  – Resilience Limitations
  – Resilience Enhancements
  – Performance Results
Experimental Setup

• SoftLayer Cluster host hosted at IBM Almaden Research Center
  – 11 nodes: four-core 2.6 GHz Intel Xeon E5-2650 CPU with 8 GB of memory

• X10:
  – Native X10, version 2.5.2
  – 4 places per node, X10_NTHREADS=1
  – X10RT sockets backend

• GML:
  – OpenBLAS version 0.2.13
    (OPENBLAS_NUM_THREADS=1)
Checkpoint and Restore Overheads

• Checkpoint every 10 iterations (3 checkpoints per run)
• A single place failure at iteration 15
• Repeat the experiments with different restore modes:
  – Shrink
  – Shrink-Rebalance
  – Redundant
Applications

• Dense
  – LinReg (50,000 X 500 per place)
  – LogReg (50,000 X 500 per place)

• Sparse
  – PageRank (2M edges per place)
Resilient X10 Overhead

Linear Regression
Overhead on 44 places: ~120%

Logistic Regression
Overhead on 44 places: ~100%

PageRank
Overhead on 44 places: ~3%
Checkpoint and Restore Overheads

• Checkpoint every 10 iterations (3 checkpoints per run)
• A single place failure at iteration 15
• Repeat the experiments with different restore modes:
  – Shrink
  – Shrink-Rebalance
  – Redundant
Time per Checkpoint

- **Linear Regression**
  - Overhead from 12 to 44 places: ~8%

- **Logistic Regression**
  - Overhead on 44 places: ~8%

- **PageRank**
  - Overhead from 12 to 44 places: ~18%
Linear Regression

From 12 to 44 places in Redundant mode: ~25%

Logistic Regression

From 12 to 44 places in Redundant mode: ~9%

PageRank

From 12 to 44 places in Redundant mode: ~67%
3 Checkpoints + 1 Restore

**Linear Regression**
Overhead to Non-Resilient X10 mode: ~273%

**Logistic Regression**
Overhead to Non-Resilient X10 mode: ~219%

**PageRank**
Overhead to Non-Resilient X10 mode: ~31%
Conclusions

• We presented a framework for developing resilient linear algebra applications using X10
  – Simple to use
  – Generic enough to be used in other libraries (i.e. ScaleGraph)
  – Assumptions:
    • Place 0 is immortal
    • Fails when 2 neighbouring processes fail
  – Reasonable scalability for dense matrix checkpoint / restore
  – *Main source of the performance overhead is due to Resilient X10 mode itself*

• Full source code freely available at http://x10-lang.org as part of GML version 2.5.2
Future Work

- Improve Resilient GML's performance:
  - Enhancements in Resilient X10
    - Support MPI, avoid the centralized resilient store
    - More efficient fault tolerance techniques
    - Use Elastic X10

- Compare Resilient GML with other frameworks (i.e. Spark).