Adaptive Reduction
Parallelization Techniques

Hao Yu and Lawrence Rauchwerger

Department of Computer Science,
Texas A&M University
http://www.cs.tamu.edu/people/rwerger, h0y8494
h0y8494, rwerger@cs.tamu.edu
Reduction and Irregular Reduction

Reduction is

- Associative and commutative operation of the form: $x = x \otimes \text{exp}$
- $x$ does not occur in $\text{exp}$ or anywhere else in the loop (except for other reduction statements.)
- $x$ can be scalar, array reference, or vector (in FORTRAN 90).

Irregular Reduction

- Refer to the reductions in the following example

```fortran
integer X(1:N) do I = 1, N
  real A(1:M) A(X(I)) = A(X(I)) \otimes \text{exp}
enddo
```

- Access pattern can not be extracted at compile-time.
Reduction Parallelization

Example:

Serial Irregular Reduction Loop

```
    do I = 1, n
    S1:   A(K(I)) = ...
    S2:   ... = A(L(I))
    S3:   A(R(I)) = A(R(I)) + exp()
    enddo
```

Parallelized Irregular Reduction Loop

```
    doall I = 1, p
        pA(1:n) = 0
        enddoall
    doall I = 1, n
        S1:   A(K(I)) = ...
        S2:   ... = A(L(I))
        S3:   pA(R(I)) = pA(R(I)) + exp()
        enddoall
        doall I = 1, n
            A(I) = A(I) + pA(I, 1:p)
        enddoall
```

- Parallel reduction is done on conformable privatized space
- Initialization and cross-processor merging may have bad cache effect

May 8, 2000

ICS 2000: Adaptive Reduction
Parallelization. --- Texas A&M Univ.
Irregular Reduction Access Patterns

Example Loop:

\[
\begin{align*}
A(1:M) \\
\text{DO } I = 1, N \\
A(B(I)) &= A(B(I)) + \exp \\
\text{ENDDO}
\end{align*}
\]

\[
pA(1:M, 1:P) \\
\text{doall } I = 1, N \\
pid = \text{GET_PID()} \\
pA(B(I), pid) &= pA(B(I), pid) + \exp \\
\text{endoall} \\
\text{...}
\]

Parameters & Index arrays

M=3, N=9, P=3

1. B(1:9) = 1,1,1, 2,2,2, 3,3,3
2. B(1:9) = 1,1,2, 2,2,3, 3,3,1
3. B(1:9) = 1,1,2, 2,2,2, 2,3,3
4. B(1:9) = 1,2,3, 1,2,3, 1,2,3

M=9, N=9, P=3

5. B(1:9) = 1,1,9, 9,9,5, 5,5,5

Reference Patterns:

(1) | (2) | (3) | (4) | (5)
---|---|---|---|---
X | X | X | X | X
X | X | X | X | X
X | X | X | X | X
X | X | X | X | X
X | X | X | X | X
X | X | X | X | X

May 8, 2000
Irregular Reduction Access Patterns

Reference Patterns from Real Codes:

1. CHARMM: Input Size = 322,288, Connectivity = 17.9, P=8
2. NBF: Input Size = 1,280,000, Connectivity = 2, P=8
3. SPARK98: Input Size = 30.169, Connectivity = 5.0, P=8
4. MOLDYN: Input Size = 70,304, Connectivity = 6.75, P=8
5. IRREG: Input Size = 1,000,000, Connectivity = 1, P=8

May 8, 2000

Motivations

- No method fits for all access patterns
- Need to classify patterns
- Library of reduction parallelization schemes.
- Adaptively select schemes from the library.
- Compile-time / Run-Time.
Some Characteristics of Irregular Memory Reference Patterns

C - Degree of Contention (of a reduction element)
\[ = \text{the number of iterations (processors) reference it.} \]

CH - the Histogram of C (based on processors).

CHR - the Ratio of CH (Contention Distribution)
\[ \frac{\sum_{p=1}^{p} CH_p}{\text{Size of replicated array}} \]

CONnectivity = \[ \frac{\text{Number of Iterations}}{\text{Number of distinct memory references}} \]

SParsity = \[ \frac{\text{Number of referenced elements}}{\text{Dimension of the array}} \]

DIMension = \[ \frac{\text{Dimension of reduction array}}{\text{Cache size}} \]

MObility = number of distinct elements referenced by one iteration

Reusability = \[ \frac{\text{Number of instantiations pattern NOT changing}}{\text{Number of instantiations pattern changing}} \]
Known Reduction Parallelization Methods

- **Direct Update**: Use unordered critical sections
  - *Advantage*: Simple, good for infrequent updates of scalars
  - *Disadvantage*: Not scalable with number of processors and high contention patterns

- **Replicated buffer**: accumulate in private conformable arrays and merge at the end.
  - *Advantage*: Simple
  - *Disadvantage*: Not scalable for sparse/low contention patterns

- **Local Write**: Iterations referencing elements on multiple processors are replicated. No global update need
  - *Advantage*: Increases locality (based on ‘owner compute’ rule)
  - *Disadvantage*: Needs inspector loop, may heavily replicate work
Replicated Buffer with Links

Method

```
    do i = 1,6
        ...  
        A(B(i)) = A(B(i)) + 1
        ... 
    enddo

    B(1:6) = (9 110 251 700 251 9)
    P = 3 (static scheduling)
```

A: Original Reduction Array

P: Replicated Buffer with an extra ‘pointer’ field in each elements

---

May 8, 2000

Replicated Buffer with Links

**Advantage**
- At the cross-processor merging phase, We only visit the touched private units along the links.
- Performs well when replicated space is sparse.

**Disadvantage**
- Requires more memory due to the additional link field.
Selective Privatization

Method

1. Instrument new index array BP to reference both A and P in parallel loop.

2. In the parallel loop, use BP as index array to do reduction on both A and P.

3. Use the built-in links to collect partial reduction results in P to A.
Selective Privatization

Advantage

• Better than ‘replicated buffer w/ links’ when data set is large.
• Good to deal with small portion of hot spots

Disadvantage

• The setup phase is expensive, so the performance depends on reusability.
• When access pattern is dense, building and traversing the extra index array are expensive.
Hash Table Privatization

Method: Phase 1

1. Partition the index array
2. Hashing the indices
3. Do reduction on private storage
4. Collect partial reduction results from P to A.

Example:

<table>
<thead>
<tr>
<th>B(1:6)</th>
<th>H(3,3)</th>
<th>P(3,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 110</td>
<td>9 110</td>
<td>1.0 2001</td>
</tr>
<tr>
<td>251 700</td>
<td>700 251</td>
<td>1.0 2004</td>
</tr>
<tr>
<td>251 9</td>
<td>9 251</td>
<td>1.0 2007</td>
</tr>
<tr>
<td>251 9</td>
<td>251 9</td>
<td>1.0 2009</td>
</tr>
</tbody>
</table>

A(1:1000)

... 0000
2.0 0009
... 0110
2.0 0251
... 0700
Hash Table Privatization

Method: Phase 2

1. Instrument index array BP.
2. Use BP as index array to do reduction on both A and P arrays.
3. Aggregate partial reduction result in P to A.

May 8, 2000

Hash Table Privatization

Advantage

- When \textit{Number of reduction references} \textless\textless \textit{Dimension of array}, the setup phase consume less memory than that of Selective Privatization.

Disadvantage

- Setup phase is expensive due to hash function when SP is not small enough
Reduction Parallelization Schemes

- Five Reduction Parallelization Schemes:
  - Replicated Buffer
  - Replicated Buffer with Links
  - Selective Privatization
  - Hash Table Privatization
  - Local Write
- Each scheme works well for certain access patterns.
- Apply the proper scheme for particular reduction access pattern
- Done at run-time
Collect Static Info.

Hash Table?

No

Parallel Loop (RepBuf)

Get Para. for Access Pattern

Choose Scheme (Decision Tree)

Yes

Parallel Loop with Hash Table Priv.

Library for Characterizing Patterns

Compile time

Run time

Parallel Loop with Chosen scheme

End
Adaptive Model Based Algorithm Selection

Decision Tree

Evaluate parameters of reference pattern

T1: SP < sp  
T2: CHR > chr1  
T3: CON < c1 && DIM > dim && MO < mo  
T4: (CHR < chr2 || HCHR > h || DIM > dim) && CON < c2

Hash Table

Local Write  |  Replicated Buffer  |  Selective Priv.  |  RepBuf w/ Links

SP: Sparsity  
CO: Connectivity  
MO: Mobility  
CHR: Contention Distribution  
HCHR: Hotspot distribution  
MO: Mobility  
DIM: Dimension
Adaptive Model Based Algorithm Selection

- The thresholds are generated at compile-time as below:

  Have a simple reduction loop
  FOR (each reduction parallelization scheme) DO
      FOR (each parameter of access pattern) DO
          Find a threshold value of the parameter when performance changes.
          DONE
      DONE
  DONE
  Adjust the threshold values by comparing across schemes.

- The decision technique is checked by applying different reduction schemes on six applications.
- Parameters are also decided by applications.
- The characteristics of access patterns can be used for other algorithm optimization.
Experimental Results

Experimental Setup

- **Experimental environment:**
  - 16 processor HP-V class
  - 4GB memory
  - HPUX11 operation system.

- **Implemented Reduction Algorithms:**
  - Replicated Buffer
  - Replicated Buffer with Links
  - Selective Privatization
  - Hash Table Privatization
  - Local Write

- **Codes and Loops:**

<table>
<thead>
<tr>
<th>Application</th>
<th>Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Irreg</td>
<td>DO 100</td>
</tr>
<tr>
<td>NBF</td>
<td>DO 50</td>
</tr>
<tr>
<td>Moldyn</td>
<td>ComputeForces loop</td>
</tr>
<tr>
<td>Charmm</td>
<td>DO 78</td>
</tr>
<tr>
<td>Spark98</td>
<td>smvpthread() loop</td>
</tr>
<tr>
<td>SPICE 2G6</td>
<td>BJT (GOTO 100) loop</td>
</tr>
</tbody>
</table>
### Experimental Results

#### Run-Time Algorithm Selection Results

<table>
<thead>
<tr>
<th>App</th>
<th>DIM</th>
<th>CON</th>
<th>MO</th>
<th>SP</th>
<th>CHR</th>
<th>Recom. Scheme</th>
<th>Experimental Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Irreg</td>
<td>100,000</td>
<td>100</td>
<td>2</td>
<td>25</td>
<td>0.92</td>
<td>rep</td>
<td>rep &gt; ll &gt; sel &gt; lw</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>20</td>
<td>5</td>
<td>5</td>
<td>0.71</td>
<td>lw</td>
<td>lw &gt; rep &gt; ll &gt; sel</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>5</td>
<td>1.25</td>
<td>0.40</td>
<td>lw</td>
<td>lw &gt; rep &gt; ll &gt; sel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2,000,000</td>
<td>1</td>
<td>0.25</td>
<td>0.26</td>
<td>sel</td>
<td>sel &gt; lw &gt; ll &gt; rep</td>
<td></td>
</tr>
<tr>
<td>Nbf</td>
<td>25,600</td>
<td>200</td>
<td>1</td>
<td>25</td>
<td>0.25</td>
<td>ll</td>
<td>sel &gt; ll &gt; rep &gt; lw</td>
</tr>
<tr>
<td></td>
<td>128,000</td>
<td>50</td>
<td>6.25</td>
<td>0.25</td>
<td>sel</td>
<td>sel &gt; ll &gt; rep &gt; lw</td>
<td></td>
</tr>
<tr>
<td></td>
<td>256,000</td>
<td>5</td>
<td>0.625</td>
<td>0.25</td>
<td>sel</td>
<td>sel &gt; ll &gt; rep &gt; lw</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1,280,000</td>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td>sel</td>
<td>sel &gt; ll &gt; rep &gt; lw</td>
<td></td>
</tr>
<tr>
<td>Moldyn</td>
<td>16,384</td>
<td>95.75</td>
<td>2</td>
<td>23.94</td>
<td>0.41</td>
<td>rep</td>
<td>rep &gt; ll &gt; sel &gt; lw</td>
</tr>
<tr>
<td></td>
<td>42,592</td>
<td>31</td>
<td>7.75</td>
<td>0.36</td>
<td>rep</td>
<td>rep &gt; ll &gt; sel &gt; lw</td>
<td></td>
</tr>
<tr>
<td></td>
<td>70,304</td>
<td>6.75</td>
<td>1.169</td>
<td>0.33</td>
<td>ll</td>
<td>ll &gt; rep &gt; sel &gt; lw</td>
<td></td>
</tr>
<tr>
<td></td>
<td>87,808</td>
<td>1.5</td>
<td>0.375</td>
<td>0.29</td>
<td>ll</td>
<td>ll &gt; rep &gt; sel &gt; lw</td>
<td></td>
</tr>
<tr>
<td>Spark98</td>
<td>30,109</td>
<td>5</td>
<td>1</td>
<td>0.625</td>
<td>0.18</td>
<td>ll</td>
<td>sel &gt; ll &gt; rep &gt; lw</td>
</tr>
<tr>
<td></td>
<td>7,294</td>
<td>4.8</td>
<td>0.6</td>
<td>0.2</td>
<td>sel</td>
<td>sel &gt; ll &gt; rep &gt; lw</td>
<td></td>
</tr>
<tr>
<td>Charmm</td>
<td>332,288</td>
<td>17.9</td>
<td>2</td>
<td>35.88</td>
<td>0.14</td>
<td>sel</td>
<td>ll &gt; sel &gt; rep &gt; lw</td>
</tr>
<tr>
<td></td>
<td>8.97</td>
<td></td>
<td>17.94</td>
<td>0.15</td>
<td>sel</td>
<td>ll &gt; sel &gt; rep &gt; lw</td>
<td></td>
</tr>
<tr>
<td></td>
<td>664,578</td>
<td>4.48</td>
<td>1.12</td>
<td>0.13</td>
<td>sel</td>
<td>ll &gt; sel &gt; rep &gt; lw</td>
<td></td>
</tr>
<tr>
<td>Spice</td>
<td>186,943</td>
<td>0.04</td>
<td>28</td>
<td>0.14</td>
<td>0.125</td>
<td>hash</td>
<td>hash &gt; ll &gt; rep</td>
</tr>
<tr>
<td></td>
<td>99,190</td>
<td>0.06</td>
<td>0.20</td>
<td>0.125</td>
<td>hash</td>
<td>hash &gt; ll &gt; rep</td>
<td></td>
</tr>
<tr>
<td></td>
<td>89,925</td>
<td>0.05</td>
<td>0.16</td>
<td>0.125</td>
<td>hash</td>
<td>hash &gt; ll &gt; rep</td>
<td></td>
</tr>
<tr>
<td></td>
<td>33,725</td>
<td>0.05</td>
<td>0.16</td>
<td>0.126</td>
<td>hash</td>
<td>hash &gt; ll &gt; rep</td>
<td></td>
</tr>
</tbody>
</table>
### Experimental Results

#### Irreg & Nbf

<table>
<thead>
<tr>
<th>App</th>
<th>DIM</th>
<th>CON</th>
<th>MO</th>
<th>SP</th>
<th>CHR</th>
<th>Recom. Scheme</th>
<th>Experimental Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Irreg</td>
<td>100,000</td>
<td>100</td>
<td>2</td>
<td>25</td>
<td>0.92</td>
<td>rep</td>
<td>rep &gt; 11 &gt; sel &gt; 1w</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>20</td>
<td></td>
<td>5</td>
<td>0.71</td>
<td>lw</td>
<td>lw &gt; rep &gt; 11 &gt; sel</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>5</td>
<td></td>
<td>1.25</td>
<td>0.40</td>
<td>lw</td>
<td>lw &gt; rep &gt; 11 &gt; sel</td>
</tr>
<tr>
<td></td>
<td>2,000,000</td>
<td>1</td>
<td></td>
<td>0.25</td>
<td>0.26</td>
<td>sel</td>
<td>sel &gt; lw &gt; 11 &gt; rep</td>
</tr>
<tr>
<td>Nbf</td>
<td>25,600</td>
<td>200</td>
<td>1</td>
<td>25</td>
<td>0.25</td>
<td>11</td>
<td>sel &gt; 11 &gt; rep &gt; 1w</td>
</tr>
<tr>
<td></td>
<td>128,000</td>
<td>50</td>
<td></td>
<td>6.25</td>
<td>0.25</td>
<td>sel</td>
<td>sel &gt; 11 &gt; rep &gt; 1w</td>
</tr>
<tr>
<td></td>
<td>256,000</td>
<td>5</td>
<td></td>
<td>0.625</td>
<td>0.25</td>
<td>sel</td>
<td>sel &gt; 11 &gt; rep &gt; 1w</td>
</tr>
<tr>
<td></td>
<td>1,280,000</td>
<td>2</td>
<td></td>
<td>0.25</td>
<td>0.25</td>
<td>sel</td>
<td>sel &gt; 11 &gt; rep &gt; 1w</td>
</tr>
</tbody>
</table>

#### IRREG – do100: Performance – Connectivity
- ReplicateBuffer
- Linked-List RepBuf
- Selective Privatization
- Local Write

#### NBF – do50: Performance – Connectivity
- ReplicateBuffer
- Linked-List RepBuf
- Selective Privatization
- Local Write

May 8, 2000

ICS 2000: Adaptive Reduction
Parallelization. --- Texas A&M Univ.
Experimental Results

<table>
<thead>
<tr>
<th>App</th>
<th>DIM</th>
<th>CON</th>
<th>MO</th>
<th>SP</th>
<th>CHR</th>
<th>Recom. Scheme</th>
<th>Experimental Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moldyn</td>
<td>16,384</td>
<td>95.75</td>
<td>2</td>
<td>23.94</td>
<td>0.41</td>
<td>rep</td>
<td>rep &gt; ll &gt; sel &gt; lw</td>
</tr>
<tr>
<td></td>
<td>42,592</td>
<td>31</td>
<td></td>
<td>7.75</td>
<td>0.36</td>
<td>rep</td>
<td>rep &gt; ll &gt; sel &gt; lw</td>
</tr>
<tr>
<td></td>
<td>70,304</td>
<td>6.75</td>
<td></td>
<td>1.169</td>
<td>0.33</td>
<td>ll</td>
<td>ll &gt; rep &gt; sel &gt; lw</td>
</tr>
<tr>
<td></td>
<td>87,808</td>
<td>1.5</td>
<td></td>
<td>0.375</td>
<td>0.29</td>
<td>ll</td>
<td>ll &gt; rep &gt; sel &gt; lw</td>
</tr>
<tr>
<td>Spark98</td>
<td>30,109</td>
<td>5</td>
<td>1</td>
<td>0.625</td>
<td>0.18</td>
<td>ll</td>
<td>sel &gt; ll &gt; rep &gt; lw</td>
</tr>
<tr>
<td></td>
<td>7,294</td>
<td>4.8</td>
<td></td>
<td>0.6</td>
<td>0.2</td>
<td>ll</td>
<td>ll &gt; sel &gt; rep &gt; lw</td>
</tr>
</tbody>
</table>

MOLDYN – ComputeForces loop: Performance – Connectivity

SPARK98 – smvpthread() loop: Performance – Connectivity

May 8, 2000

ICS 2000: Adaptive Reduction
Parallelization. --- Texas A&M Univ.
## Experimental Results

### Charmm & Spice

<table>
<thead>
<tr>
<th>App</th>
<th>DIM</th>
<th>CON</th>
<th>MO</th>
<th>SP</th>
<th>CHR</th>
<th>Recom. Scheme</th>
<th>Experimental Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charmm</td>
<td>332,288</td>
<td>17.9</td>
<td>2</td>
<td>35.88</td>
<td>0.14</td>
<td>sel</td>
<td>sel</td>
</tr>
<tr>
<td></td>
<td>8.97</td>
<td></td>
<td></td>
<td>17.94</td>
<td>0.15</td>
<td>sel</td>
<td>sel</td>
</tr>
<tr>
<td></td>
<td>664,578</td>
<td>4.48</td>
<td></td>
<td>1.12</td>
<td>0.13</td>
<td>sel</td>
<td>sel</td>
</tr>
<tr>
<td>Spice</td>
<td>186,943</td>
<td>0.04</td>
<td>28</td>
<td>0.14</td>
<td>0.125</td>
<td>hash</td>
<td>hash &gt; 11 &gt; rep</td>
</tr>
<tr>
<td></td>
<td>99,190</td>
<td>0.06</td>
<td></td>
<td>0.20</td>
<td>0.125</td>
<td>hash</td>
<td>hash &gt; 11 &gt; rep</td>
</tr>
<tr>
<td></td>
<td>89,925</td>
<td>0.05</td>
<td></td>
<td>0.16</td>
<td>0.125</td>
<td>hash</td>
<td>hash &gt; 11 &gt; rep</td>
</tr>
<tr>
<td></td>
<td>33,725</td>
<td>0.05</td>
<td></td>
<td>0.16</td>
<td>0.126</td>
<td>hash</td>
<td>hash &gt; 11 &gt; rep</td>
</tr>
</tbody>
</table>

### CHARMMM – do78: Performance – Connectivity

- ReplicateBuffer
- Linked-List RepBuf
- Selective Privatization
- Local Write

Connectivity: 17.94, 8.97, 4.485

Reusability = 99

### SPICE – bjt100: Performance

- ReplicateBuffer
- Linked-List RepBuf
- Hash table Privatization

Connectivity: 0.04–613, 0.05–425, 0.06–3544

NI: number of iterations

NI = 1584, 4224, 7392, 5760
Experimental Results

May 8, 2000

Experimental Results

Overhead of techniques

Read the Graph

- Relative Execution Time (RET) = \frac{\text{Actual Parallel Execution Time} \times P}{\text{Sequential Execution Time}}

- Loop: RET of parallelized reduction loop
- Overhead: RET of the overheads of reduction parallelization outside the reduction loop.

Observation

- Sel, Hash: expensive privatized storage reconstruction
- Local Write: expensive inspector
Conclusion

Contributions:

• New techniques for efficiently parallelizing irregular, sparse reductions:
• Method for characterizing irregular access patterns
• Adaptive model based reduction algorithm selection allows us to choose the best scheme available to us. — Decision Tree

Future Works:

• Method for slow pattern change.
• More optimizations by modeling program behavior for irregular codes could be done in a run-time adaptive manner, and this work is just the beginning.