Automatic Adaptive Prefetching for Fine-grain Communication in Chapel

Thomas B. Rolinger (UMD/LPS), Alan Sussman (UMD)
Contact: tbrolin@cs.umd.edu
CHIUW 2023
Motivation: Fine-grain Communication

```python
for all v in G {
    var val = 0.0;
    const ref neighbors = v.neighbors;
    for i in neighbors.domain {
        ref t = G[neighbors[i]];
        val += t.pr_read / t.out_degree;
    }
    v.pr_write = (val * d) + ((1.0-d)/num_vertices);
}
```

PageRank (graph analytic)
Shared- and distributed-memory parallel
Motivation: Fine-grain Communication

```
forall v in G {
  var val = 0.0;
  const ref neighbors = v.neighbors;
  for i in neighbors.domain {
    ref t = G[neighbors[i]];
    val += t.pr_read / t.out_degree;
  }
  v.pr_write = (val * d) + ((1.0-d)/num_vertices);
}
```

- **Irregular** memory access to distributed array → fine-grain communication (i.e., small messages sent over network)

- This memory access pattern also found in some scientific applications
Motivation: Fine-grain Communication

PageRank (graph analytic)
Shared- and distributed-memory parallel

```c
forall v in G {
    var val = 0.0;
    const ref neighbors = v.neighbors;
    for i in neighbors.domain {
        ref t = G[neighbors[i]];
        val += t.pr_read / t.out_degree;
    }
    v.pr_write = (val * d) + ((1.0-d)/num_vertices);
}
```
Motivation: Fine-grain Communication

Fine-grain communication leads to excessive stalls waiting for data to arrive over the network.

High productivity does not always lead to good performance.

```
for all v in G {
    var val = 0.0;
    const ref neighbors = v.neighbors;
    for i in neighbors.domain {
        ref t = G[neighbors[i]];
        val += t.pr_read / t.out_degree;
    }
    v.pr_write = (val * d) + ((1.0-d)/num_vertices);
}
```

PageRank (graph analytic)
Shared- and distributed-memory parallel
Motivation: Fine-grain Communication

Fine-grain communication leads to excessive stalls waiting for data to arrive over the network.

High productivity does not always lead to good performance.

Can we achieve **better performance** for these types of codes in Chapel **WITHOUT losing productivity**?
Outline

- **Optimization**: Adaptive Remote Prefetching
- **Implementation within compiler:**
  - Static analysis and code transformations
- **Performance evaluation:**
  - PageRank
  - SSSP
Outline

• **Optimization**: Adaptive Remote Prefetching
  
  • Implementation within compiler:
    • Static analysis and code transformations
  
  • Performance evaluation:
    • PageRank
    • SSSP
Adaptive Remote Prefetching

• What is **prefetching**?
  • **hide** communication latency by **overlapping** it with other communication/computation
  • issue **non-blocking** reads for remote data that will be needed in the **future**
Adaptive Remote Prefetching

• What is prefetching?
  • hide communication latency by overlapping it with other communication/computation
  • issue non-blocking reads for remote data that will be needed in the future

indirect/irregular access pattern

```plaintext
1 forall i in ... {
2   C[i] = A[B[i]];
3 }
```

no prefetching

Adaptive Remote Prefetching

• What is **prefetching**?
  • **hide** communication latency by **overlapping** it with other communication/computation
  • issue **non-blocking** reads for remote data that will be needed in the **future**

```plaintext
forall i in ... {
  C[i] = A[B[i]];
}
```

indirect/irregular access pattern

**no prefetching**

<table>
<thead>
<tr>
<th>i</th>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fetch A[B[0]]</td>
<td>stall</td>
</tr>
<tr>
<td>1</td>
<td>fetch A[B[1]]</td>
<td>stall</td>
</tr>
<tr>
<td>2</td>
<td>fetch A[B[2]]</td>
<td>stall</td>
</tr>
</tbody>
</table>

**prefetching**

<table>
<thead>
<tr>
<th>i</th>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fetch A[B[0]]</td>
<td>stall</td>
</tr>
<tr>
<td>1</td>
<td>fetch A[B[1]]</td>
<td>stall</td>
</tr>
<tr>
<td>2</td>
<td>fetch A[B[2]]</td>
<td>cache hit</td>
</tr>
</tbody>
</table>
Adaptive Remote Prefetching

• What is **prefetching**?
  • **hide** communication latency by **overlapping** it with other communication/computation
  • issue **non-blocking** reads for remote data that will be needed in the **future**

```plaintext
1 forall i in ... {
2   C[i] = A[B[i]];
3 }
```

indirect/irregular access pattern

- **prefetch distance** of 2

---

**no prefetching**

- i = 0, fetch A[B[0]]
- i = 1, fetch A[B[1]]
- i = 2, fetch A[B[2]]

**prefetching**

- i = 0, fetch A[B[0]]
- i = 1, fetch A[B[1]]
- i = 2, fetch A[B[2]] cache hit
Adaptive Remote Prefetching

• What is **prefetching**?
  • hide communication latency by **overlapping** it with other communication/computation
  • issue **non-blocking** reads for remote data that will be needed in the **future**

• What are we prefetching into?
  • Chapel’s **remote cache**
  • Each core (**task**) on a locale has its own software managed remote cache
  • As a result, each task has its **own prefetch distance** that must be determined independently from other other tasks
Adaptive Remote Prefetching (cont.)

• How to pick a “good” *prefetch distance*: $A[B[i+??]]$
  • Very difficult to *statically* pick for a given workload/dataset → *memory access patterns change throughout the program*
  • The “best” value will often be different across *applications, datasets* and *systems*
Adaptive Remote Prefetching (cont.)

• How to pick a “good” prefetch distance: A[B[i+??]]
  • Very difficult to statically pick for a given workload/dataset → memory access patterns change throughout the program
  • The ”best” value will often be different across applications, datasets and systems

• Solution: adaptive prefetching
  • Adapt (increase/decrease) the prefetch distance as the program executes
  • Uses runtime information about the memory access pattern and effectiveness of the prefetches issued thus far
    • how many prefetches were issued? how many were late?
Outline

- **Optimization**: Adaptive Remote Prefetching
- **Implementation within compiler**:  
  - Static analysis and code transformations
- **Performance evaluation**:  
  - PageRank  
  - SSSP
Implementation within Compiler

• **Static analysis**
  • *Automatically* identifies potential fine-grain communication in *forall* loops
  • Specifically looks for $A[B[i]]$ patterns where $A$ is a distributed-array
  • Ensures that we can reason about how the loop iterations *progress* (important for *bounds checking*)
Implementation within Compiler

• **Static analysis**
  • **Automatically** identifies potential fine-grain communication in **forall** loops
  • Specifically looks for \texttt{A[\texttt{B[i]}]} patterns where \texttt{A} is a distributed-array
  • Ensures that we can reason about how the loop iterations progress (important for **bounds checking**)

• **Code transformations**
  • Creates variables for bounds checking, the prefetch distances, etc.
  • Inserts bounds checking around prefetch
  • Adds code to periodically adjust the prefetch distances
  • Generates prefetch call to remote cache
Implementation within Compiler

• Static analysis
  • Automatically identifies potential fine-grain communication in `forall` loops
  • Specifically looks for `A[B[i]]` patterns where `A` is a distributed array
  • Ensures that we can reason about how the loop iterations progress (important for bounds checking)

• Code transformations
  • Creates variables for bounds checking, the prefetch distances, etc.
  • Inserts bounds checking around prefetch
  • Adds code to periodically adjust the prefetch distances
  • Generates prefetch call to remote cache

**Take away:** Applying this optimization **manually decreases productivity**
Outline

• **Optimization**: Adaptive Remote Prefetching

• **Implementation within compiler**:
  • Static analysis and code transformations

• **Performance evaluation**:
  • PageRank
  • SSSP
Experimental Setup

- **Workloads**: PageRank and SSSP

## Data sets

<table>
<thead>
<tr>
<th>Name</th>
<th># Vertices</th>
<th># Edges</th>
<th>Density (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale-24</td>
<td>16M</td>
<td>536M</td>
<td>1.9e-4</td>
</tr>
<tr>
<td>scale-25</td>
<td>33M</td>
<td>1B</td>
<td>9.5e-5</td>
</tr>
<tr>
<td>scale-26</td>
<td>67M</td>
<td>2B</td>
<td>4.8e-5</td>
</tr>
<tr>
<td>arabic-2005</td>
<td>23M</td>
<td>631M</td>
<td>1.2e-4</td>
</tr>
<tr>
<td>webbase-2001</td>
<td>118M</td>
<td>992M</td>
<td>7.1e-6</td>
</tr>
<tr>
<td>GAP-twitter</td>
<td>61M</td>
<td>1.5B</td>
<td>3.9e-5</td>
</tr>
<tr>
<td>sk-2005</td>
<td>50M</td>
<td>2B</td>
<td>7.5e-5</td>
</tr>
<tr>
<td>MOLIERE_2016</td>
<td>30M</td>
<td>6.6B</td>
<td>7.3e-4</td>
</tr>
</tbody>
</table>

```cpp
for all v in G {
    var val = 0.0;
    const ref neighbors = v.neighbors;
    for i in neighbors.domain {
        ref t = G[neighbors[i]];
        val += t.pr_read / t.out_degree;
    }
    v.pr_write = (val * d) + ((1.0-d)/num_vertices);
}
```

```cpp
for all u in cq {
    foreach i in G[u].neighbors.domain {
        const v_weight = G[u].weights[i];
        if (v_weight < delta) {
            const w = G[u].dist + v_weight;
            ref v = G[neighbors[i]];
            if (v.dist < 0 || v.dist > w) {
                v.dist = w;
                if (v.visited && w < max_delta) {
                    v.visited = true;
                    nextQs[G[v.id].locale.id] += idx;
                }
            }
        }
    }
}
```
Experimental Setup (cont.)

• **Platforms:** Three different distributed-memory systems

<table>
<thead>
<tr>
<th>Name</th>
<th>CPUs</th>
<th># Cores/node</th>
<th>Memory/node</th>
<th>Interconnect</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDR-IB</td>
<td>Intel Xeon E5-2650</td>
<td>20</td>
<td>512 GB</td>
<td>FDR Infiniband</td>
</tr>
<tr>
<td>HDR-IB</td>
<td>AMD EPYC 7763</td>
<td>16</td>
<td>64 GB</td>
<td>HDR Infiniband</td>
</tr>
<tr>
<td>Cray XC</td>
<td>Intel Xeon E5-2699</td>
<td>44</td>
<td>128 GB</td>
<td>Cray Aries</td>
</tr>
</tbody>
</table>
PageRank: MOLIERE_2016

SSSP: scale-26

<table>
<thead>
<tr>
<th># nodes</th>
<th>FDR-IB</th>
<th>HDR-IB</th>
<th>Cray XC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.8</td>
<td>1.4</td>
<td>1.3</td>
</tr>
<tr>
<td>4</td>
<td>1.4</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>8</td>
<td>2.1</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>16</td>
<td>2.3</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>32</td>
<td>2.6</td>
<td>2.3</td>
<td>1.7</td>
</tr>
</tbody>
</table>

speed-up over baseline
PageRank: MOLIERE_2016

**Speed-up over baseline**

<table>
<thead>
<tr>
<th># nodes</th>
<th>FDR-IB</th>
<th>HDR-IB</th>
<th>Cray XC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.8</td>
<td>1.4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1.6</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>8</td>
<td>1.9</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>16</td>
<td>2.1</td>
<td>2</td>
<td>1.4</td>
</tr>
<tr>
<td>32</td>
<td>2.2</td>
<td>1.6</td>
<td>2</td>
</tr>
</tbody>
</table>

PageRank: MOLIERE_2016

- Speed-ups as high as **3.2x** and **1.7x**

SSSP: scale-26

**Speed-up over baseline**

<table>
<thead>
<tr>
<th># nodes</th>
<th>FDR-IB</th>
<th>HDR-IB</th>
<th>Cray XC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.3</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>1.4</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>8</td>
<td>1.4</td>
<td>1.4</td>
<td>1.1</td>
</tr>
<tr>
<td>16</td>
<td>1.7</td>
<td>1.1</td>
<td>1.3</td>
</tr>
<tr>
<td>32</td>
<td>1.1</td>
<td>1.1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Note:**
- The diagrams display speed-up over baseline for different numbers of nodes. The red box highlights the maximum speed-up for each benchmark.
Results: Is Adapting the Distance Helping?
Results: Is Adapting the Distance Helping?
Results: Is Adapting the Distance Helping?

different tasks/cores across the system – each has their own cache/prefetch distance
Results: Is Adapting the Distance Helping?

different values of the prefetch distance used

- 1
- 10
- 20
- 30
- 40

# prefetched (millions)

- 7
- 6
- 5
- 4
- 3
- 2
- 1
- 0

- task ID
Results: Is Adapting the Distance Helping?

Most cores/tasks favor a prefetch distance of 12
Results: Is Adapting the Distance Helping?

Different patterns across workloads and systems

→ distances are adapting to changing environments
Results: Is Adapting the Distance Helping?

Manually picking a “good” distance offline and use that throughout the entire program:
- **up to 44% worse** performance vs. adapting the distance
Discussion and Future Work

- More sophisticated heuristics to adjust the distance
- Auto-tuning to intelligently select the tunable parameters
  - How often to adjust distance, tolerance of late prefetches
- Evaluate more architectures/systems/workloads
Summary

PageRank: Runtime Scalability

- Better performance -- how? (adaptive prefetching)
- Poor performance -- why? (fine-grain remote communication)

Runtime speed-ups

# of nodes

Baseline

Optimized