Automatic Adaptive Prefetching for Fine-grain Communication in Chapel

Thomas B. Rolinger (UMD/LPS), Alan Sussman (UMD)

Contact: tbrolin@cs.umd.edu

CHIUW 2023

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

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);
}
```

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);
}
```

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

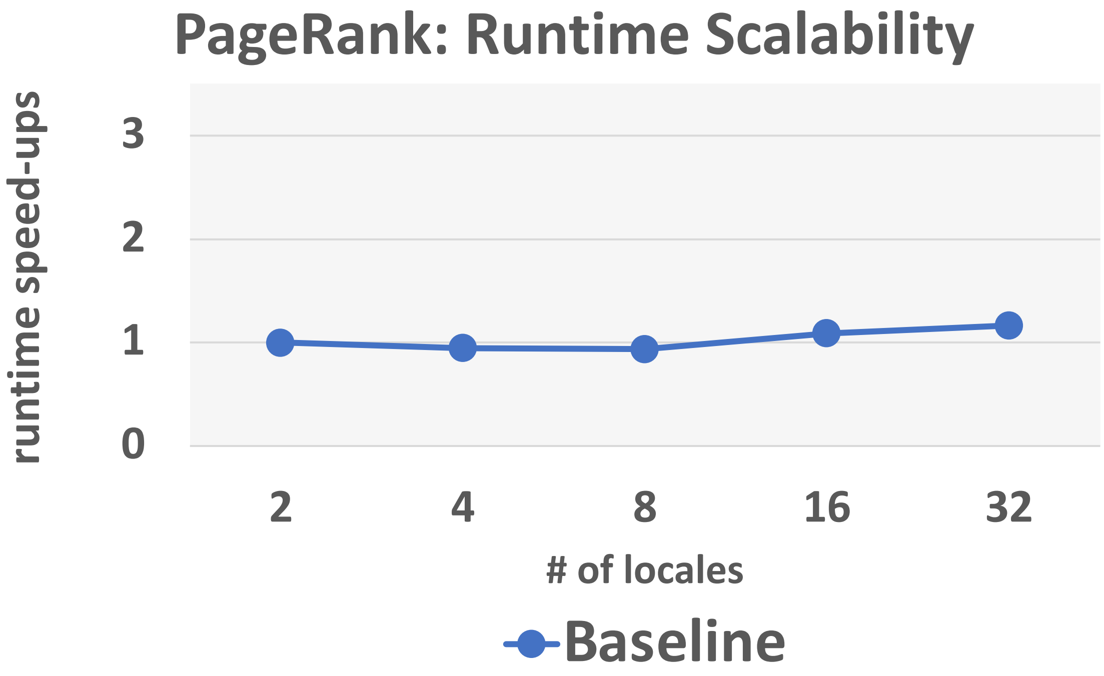
Irregular memory access to distributed array → fine-grain communication (i.e., small messages sent over network)

This memory access pattern also found in some scientific applications

Motivation: Fine-grain Communication

```
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);
}
```

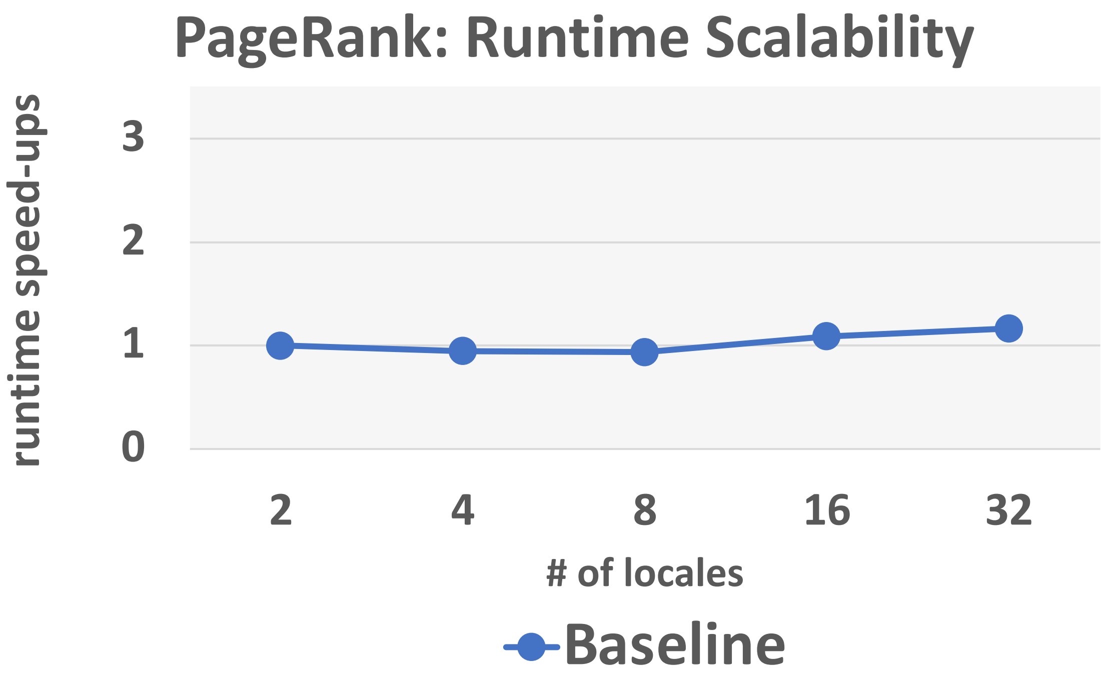
PageRank (graph analytic)
Shared- and distributed-memory parallel



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);
}
```

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



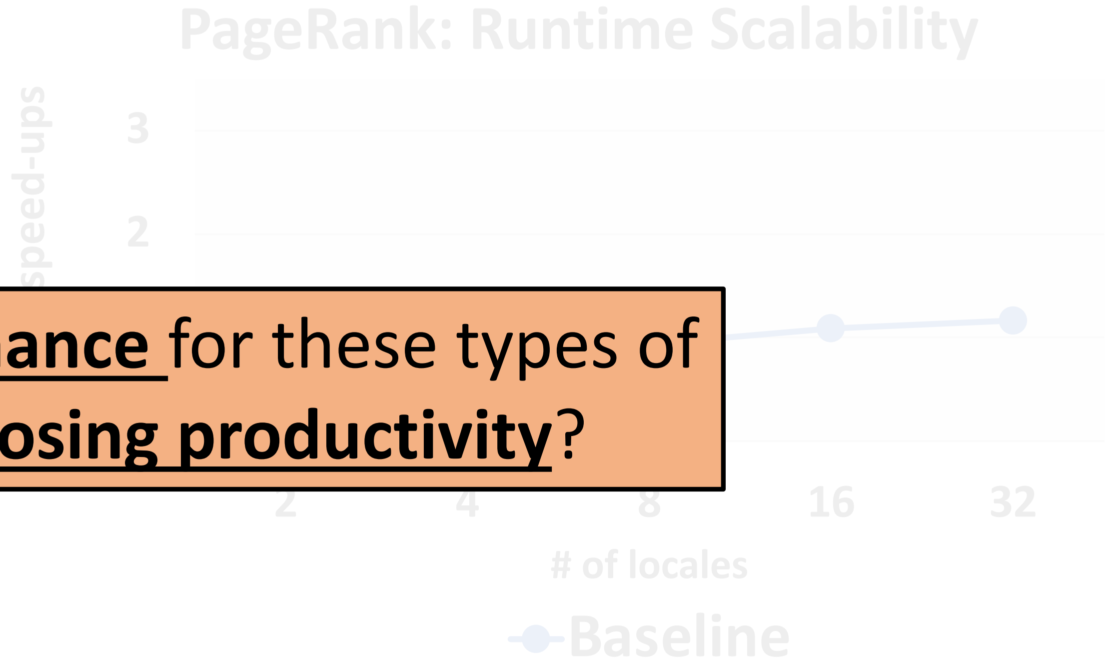
Fine-grain communication leads to <u>excessive stalls</u> waiting for data to arrive over the network

High productivity does not always lead to good performance

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.p
  }
  v.pr_write =
}
```

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



Can we achieve **<u>better performance</u>** for these types of codes in Chapel WITHOUT **<u>losing productivity</u>**?

Fine-grain communication leads to <u>excessive stalls</u> waiting for data to arrive over the network

High productivity does not always lead to good performance

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

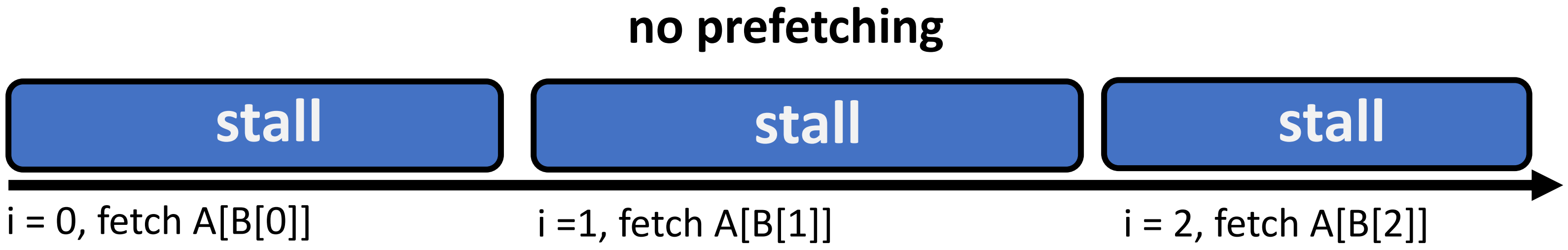
- What is **prefetching**?
  - **hide** communication latency by **overlapping** it with other communication/computation
  - issue **non-blocking** reads for remote data that will be needed in the **future**

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**

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

indirect/irregular access pattern

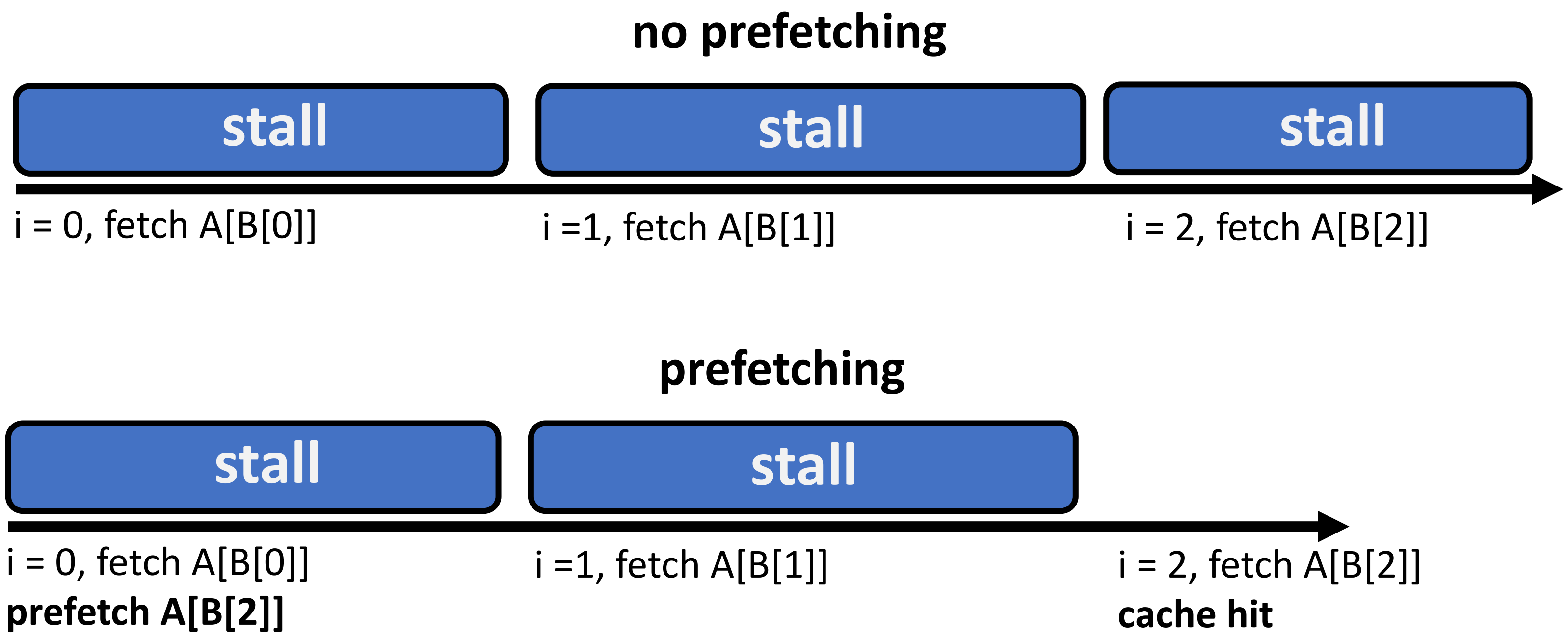


Adaptive Remote Prefetching

- 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

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

indirect/irregular access pattern



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

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

indirect/irregular access pattern

no prefetching

prefetching

prefetch distance of 2

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 **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

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
  - Ensur ... portant for **bo**

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

- **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

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

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

PageRank kernel

```
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);
}
```

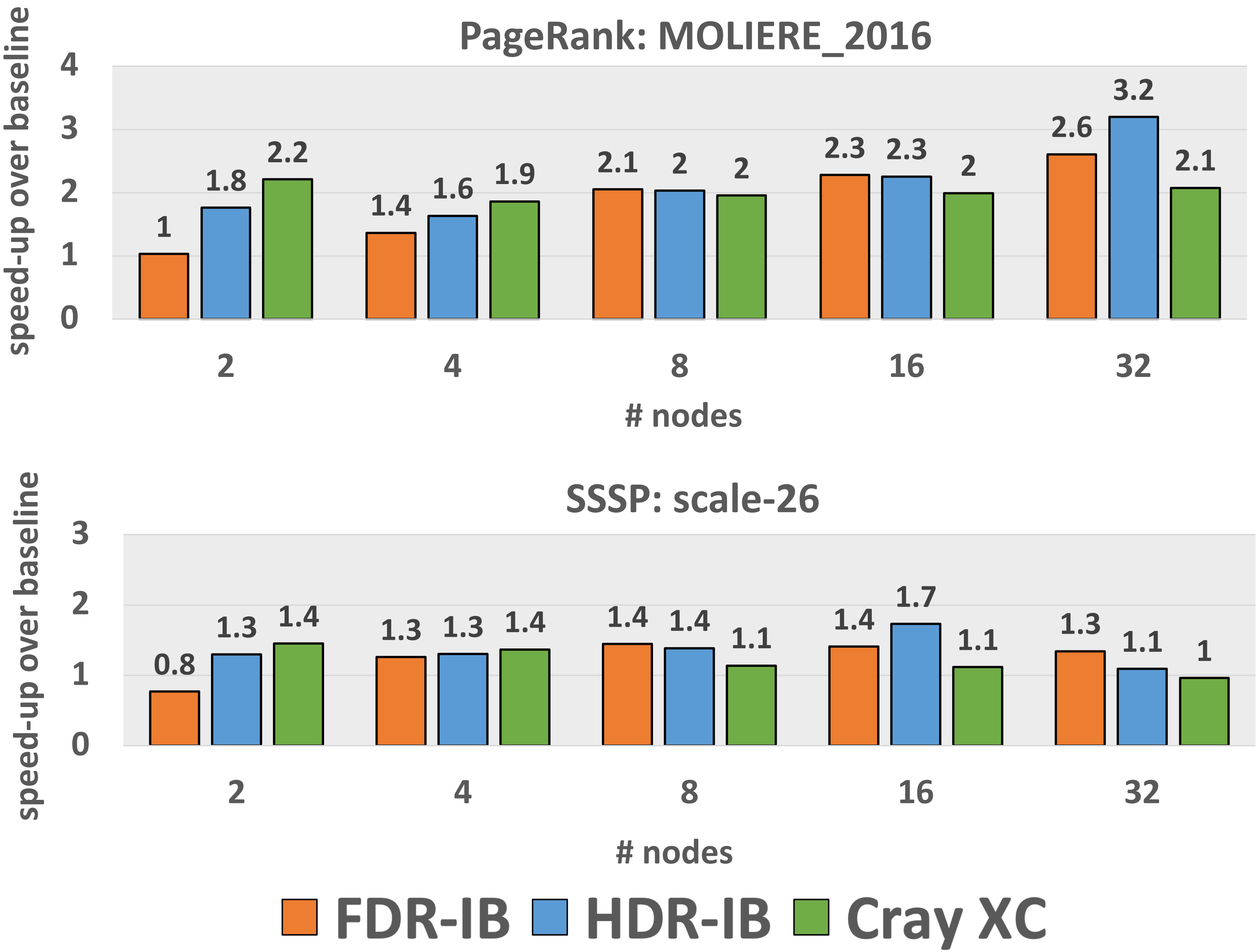
SSSP main kernel

```
forall 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;
        }
      }
    }
  }
}
```

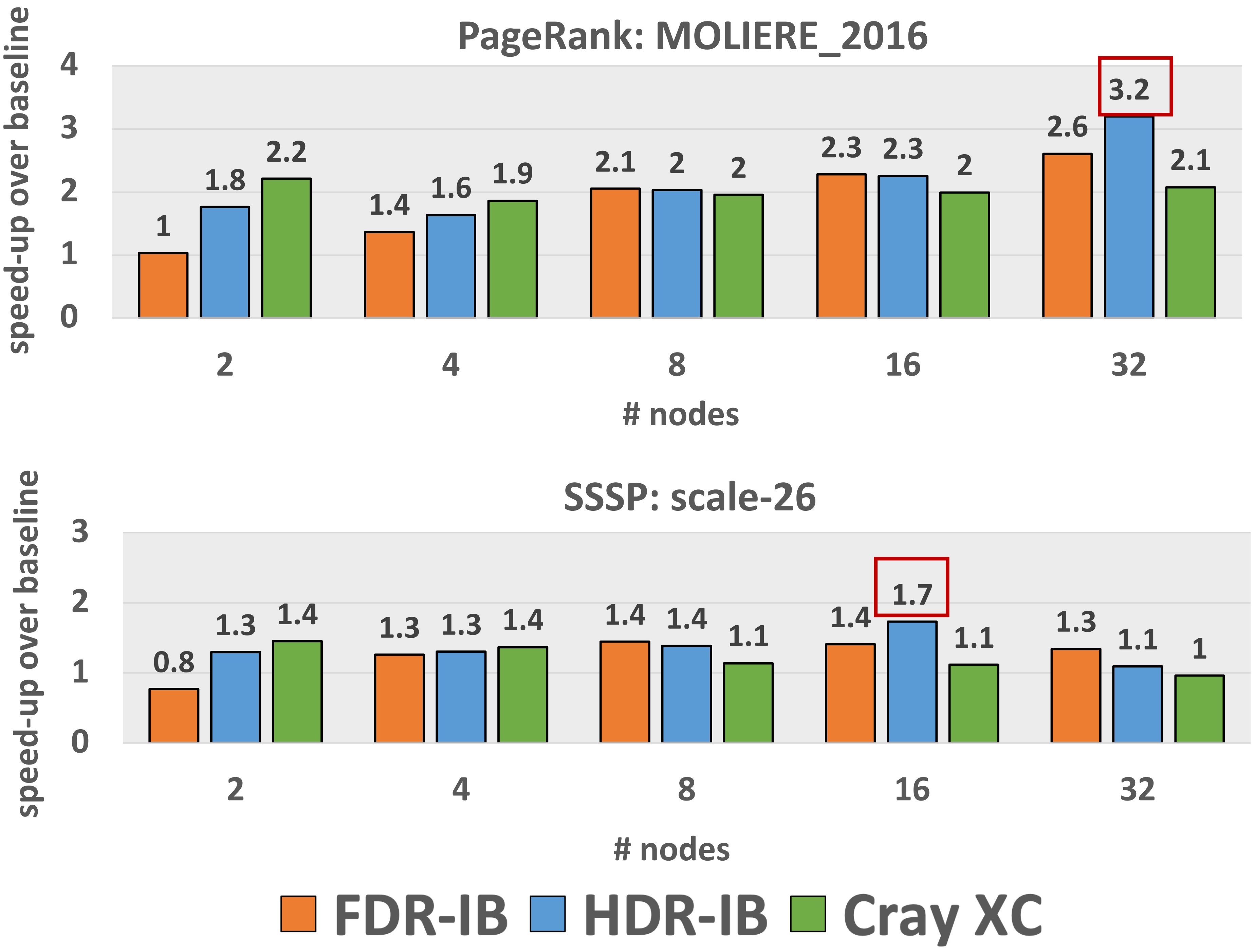
Experimental Setup (cont.)

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

Name	CPUs	# Cores/node	Memory/node	Interconnect
FDR-IB	Intel Xeon E5-2650	20	512 GB	FDR Infiniband
HDR-IB	AMD EPYC 7763	16	64 GB	HDR Infiniband
Cray XC	Intel Xeon E5-2699	44	128 GB	Cray Aries



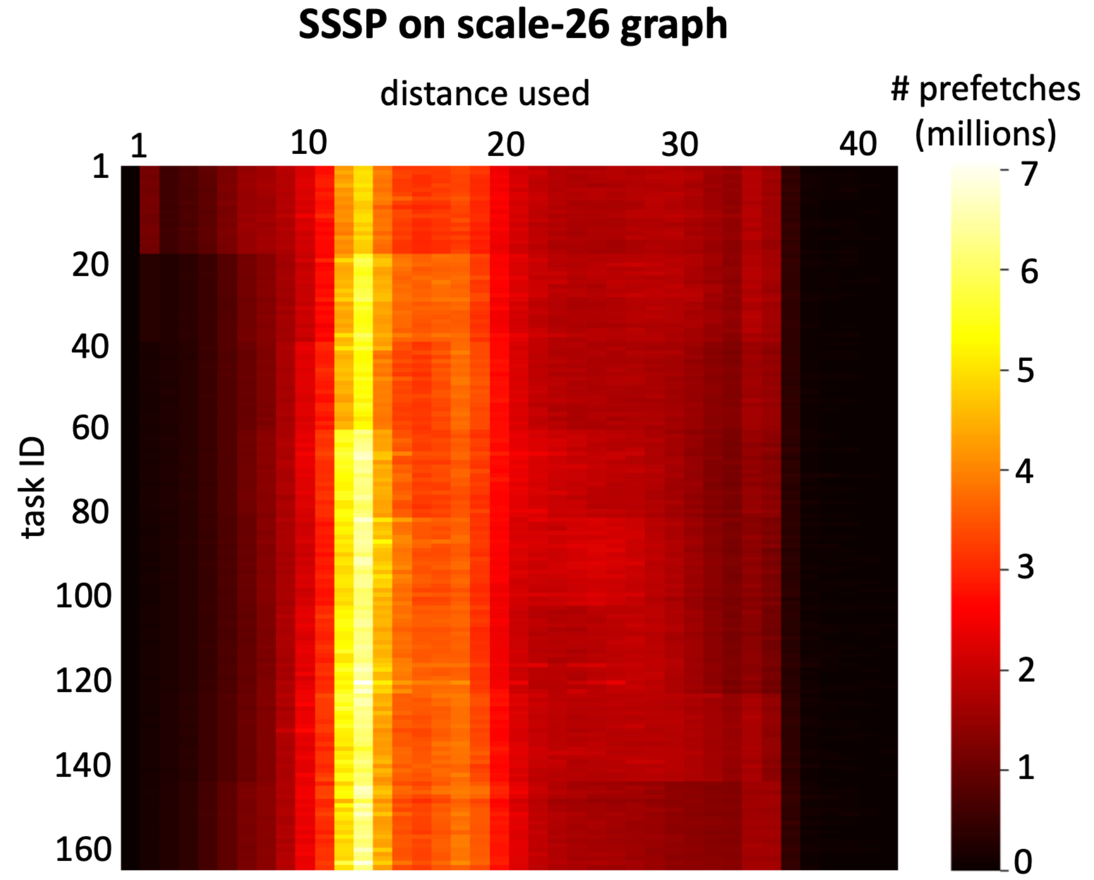
PageRank: MOLIERE_2016
speed-up over baseline
4
3
2
1
0
1
1.8
2.2
1.4
1.6
1.9
2.1
2
2
2.3
2.3
2
2.6
3.2
2.1
2
4
8
16
32
nodes
SSSP: scale-26
speed-up over baseline
3
2
1
0
0.8
1.3
1.4
1.3
1.3
1.4
1.4
1.4
1.1
1.4
1.7
1.1
1.3
1.1
1
2
4
8
16
32
nodes
FDR-IB
HDR-IB
Cray XC



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

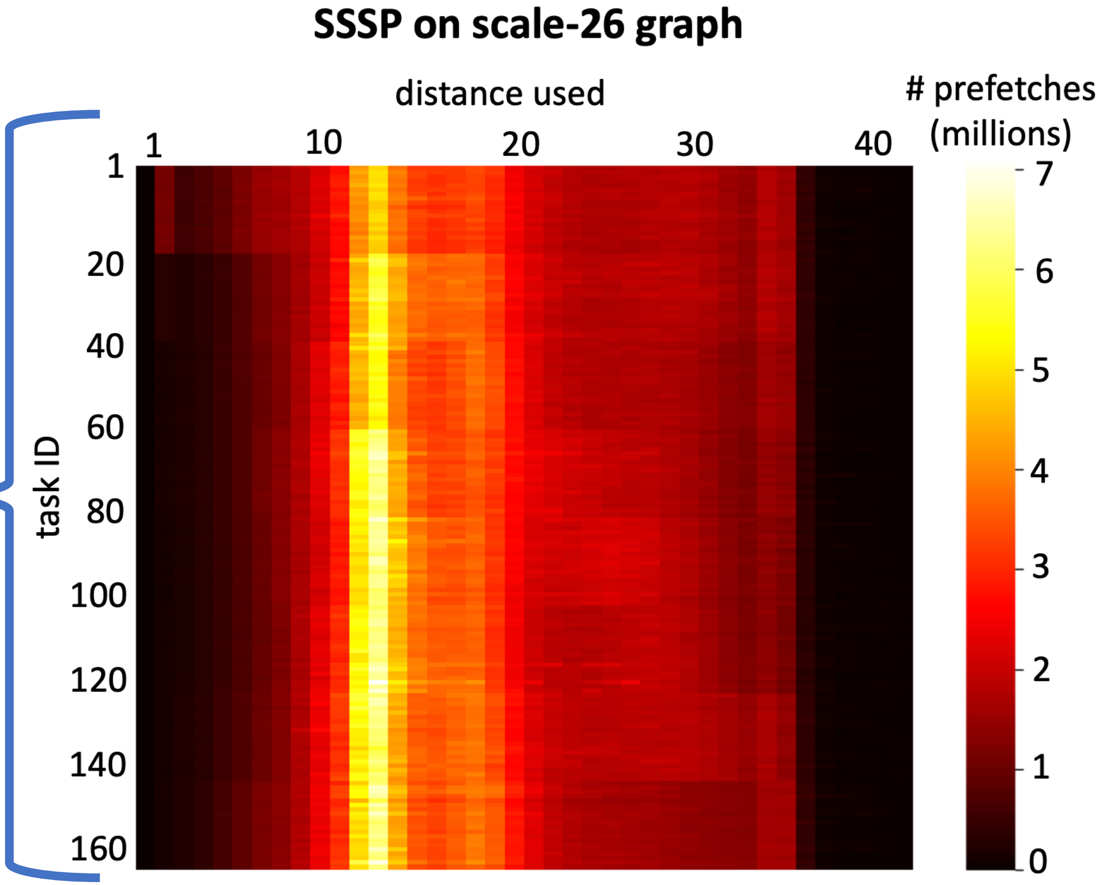
Results: Is Adapting the Distance Helping?

Results: Is Adapting the Distance Helping?



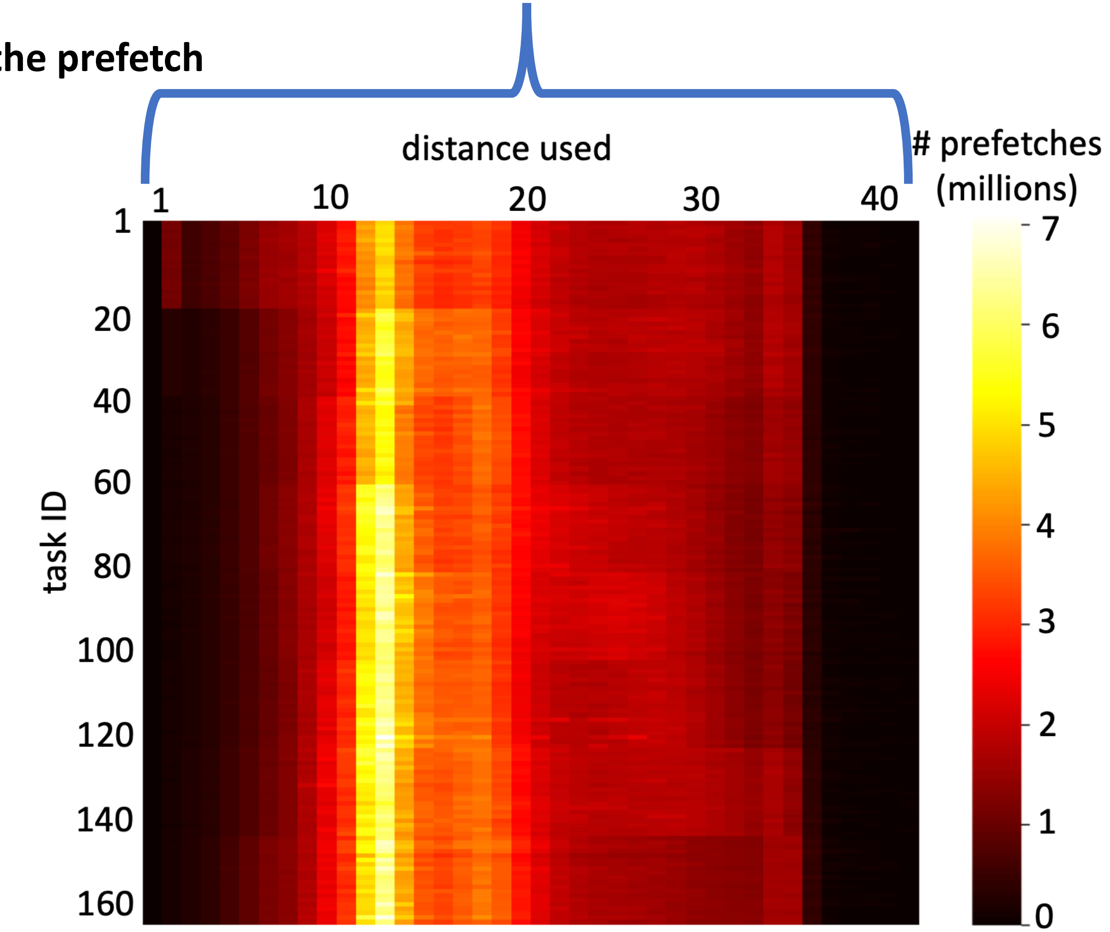
Results: Is Adapting the Distance Helping?

different tasks/cores across the system – each has their own cache/prefetch distance

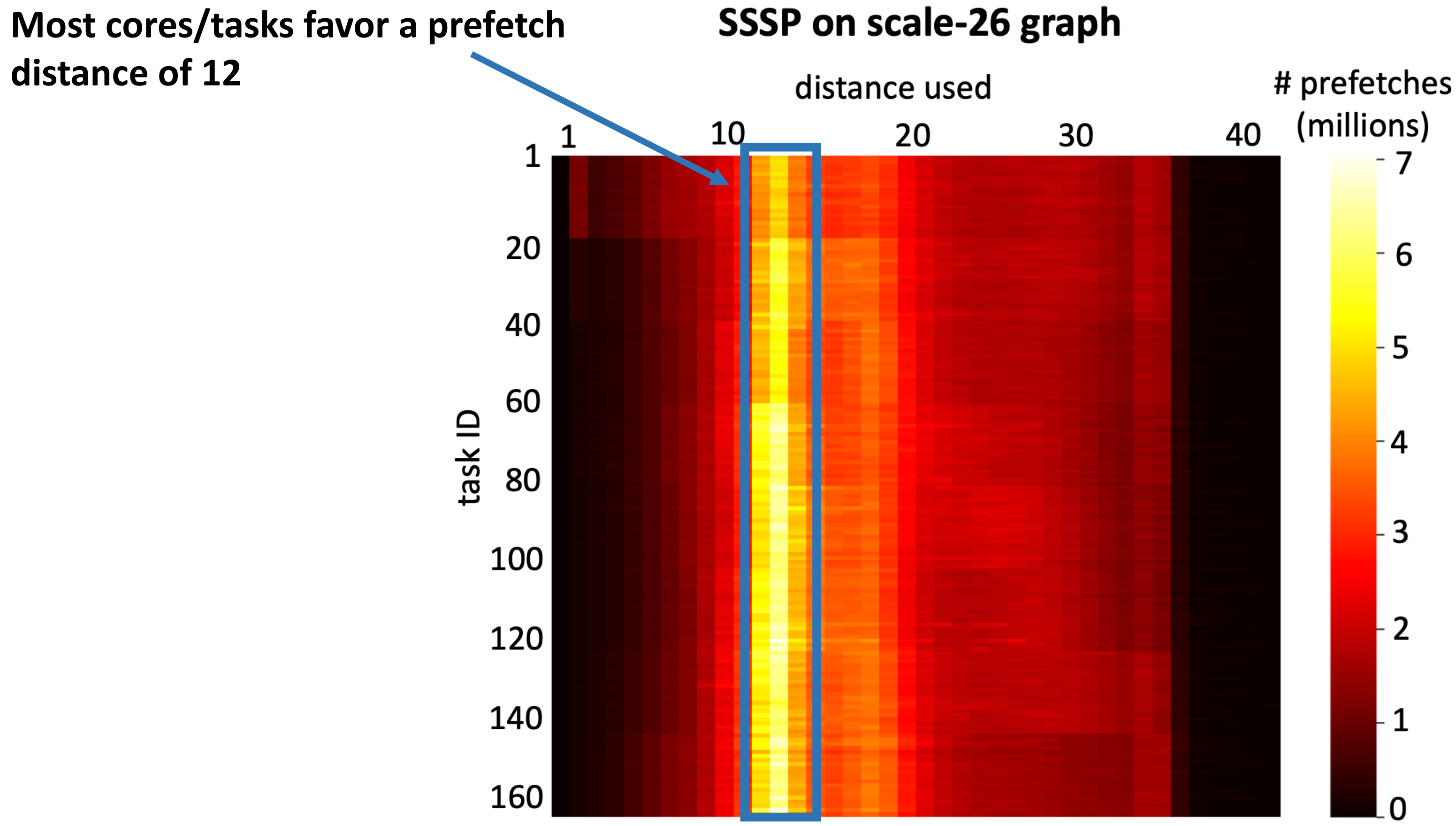


Results: Is Adapting the Distance Helping?

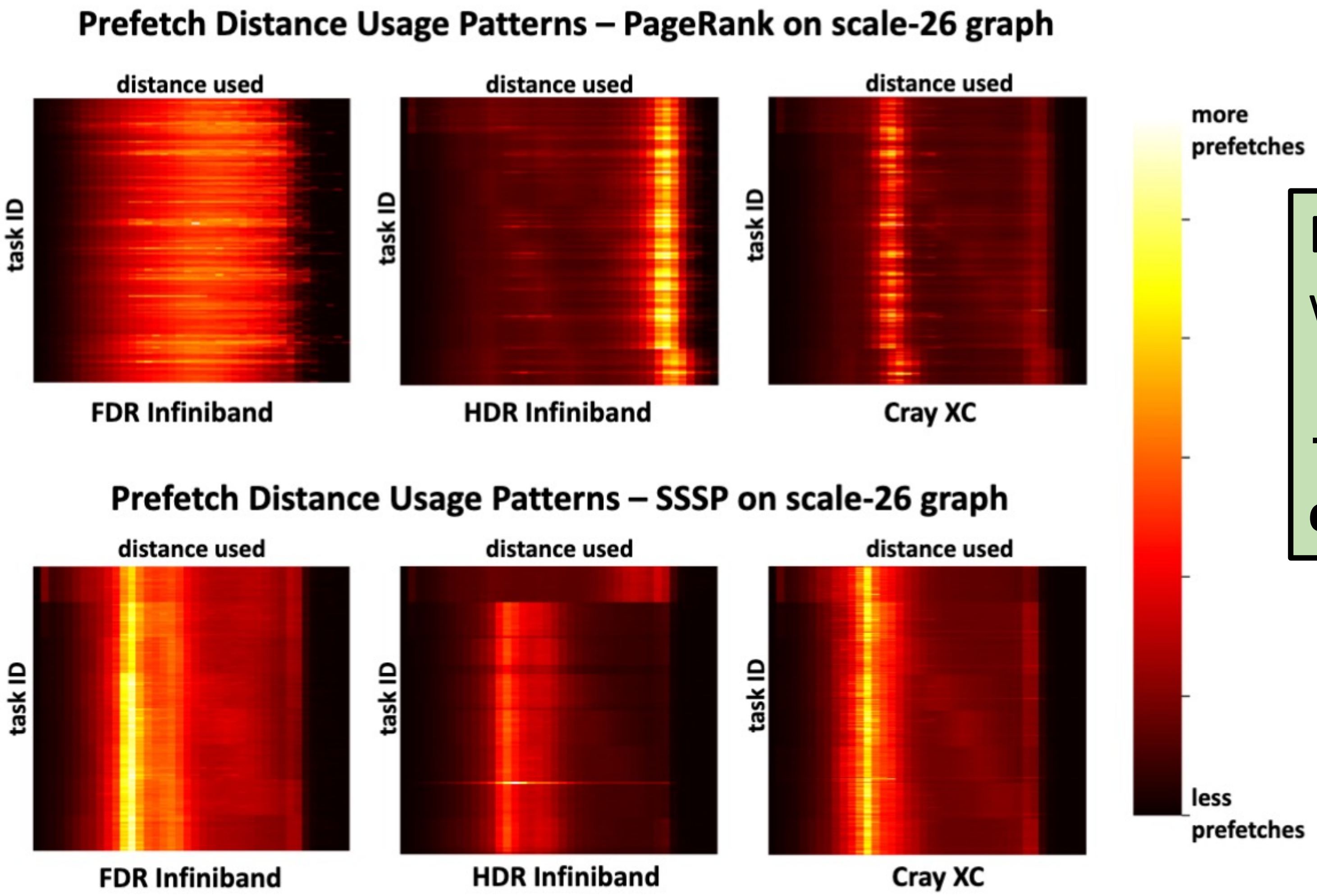
different values of the prefetch
distance used



Results: Is Adapting the Distance Helping?



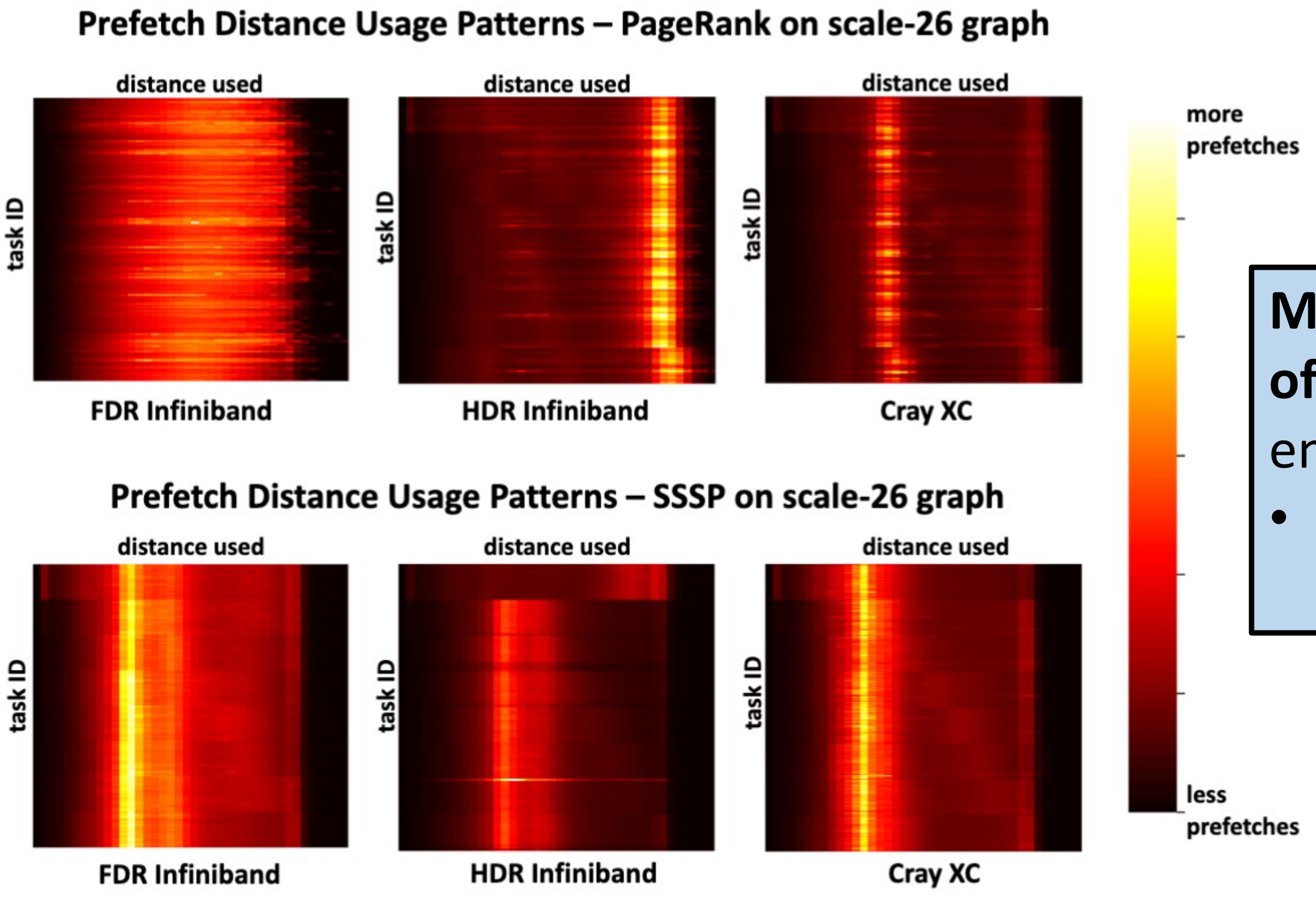
Results: Is Adapting the Distance Helping?



Different patterns across workloads and systems

→ distances are adapting to **changing environments**

Results: Is Adapting the Distance Helping?



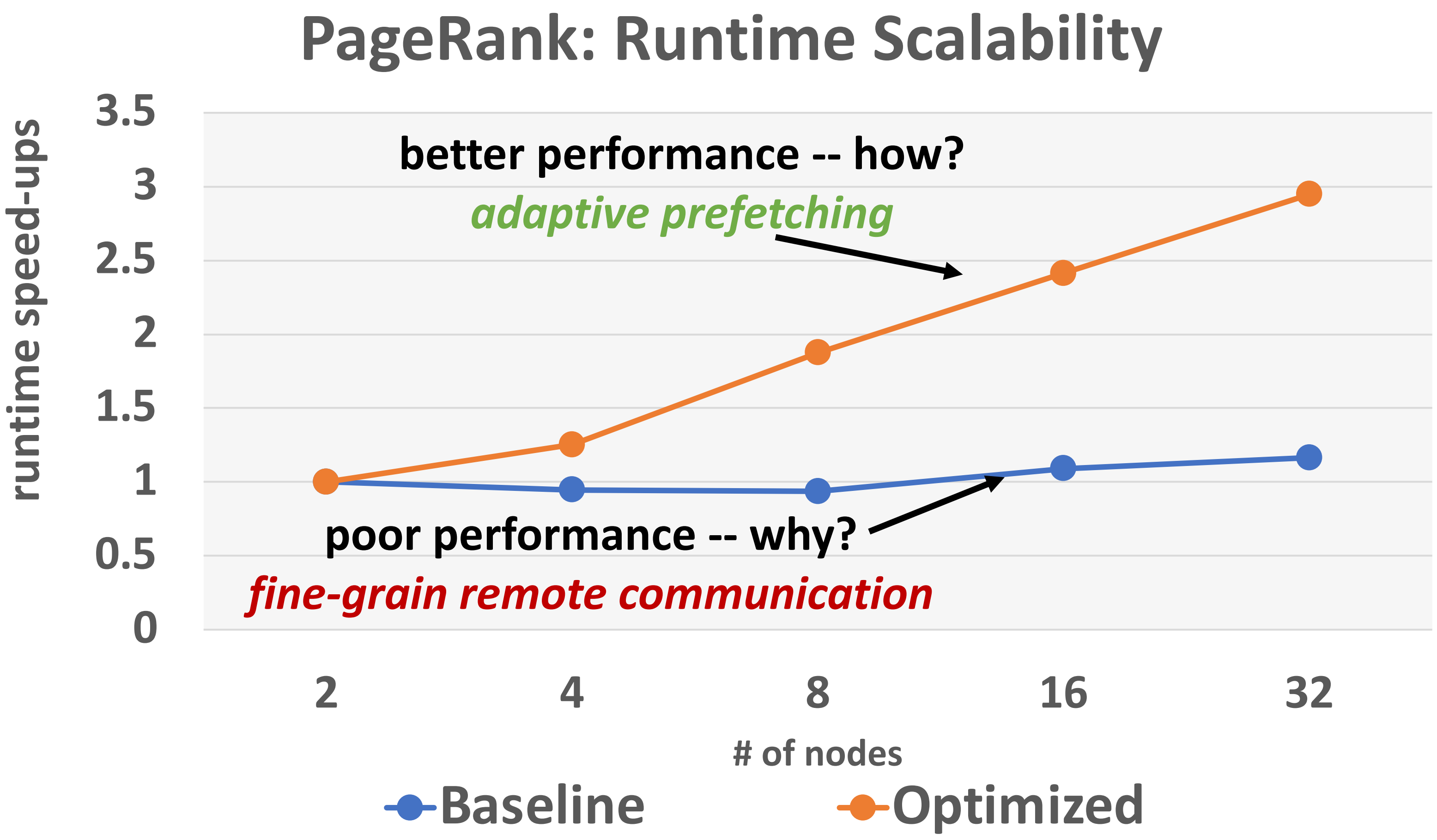
Manually picking a "good" distance **offline** and use that throughout the entire program:

- **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
runtime speed-ups
3.5
3
2.5
2
1.5
1
0.5
0
better performance -- how?
adaptive prefetching
poor performance -- why?
fine-grain remote communication
2
4
8
16
32
# of nodes
Baseline
Optimized