Adaptive Prefetching for Fine-grain Communication in PGAS Programs

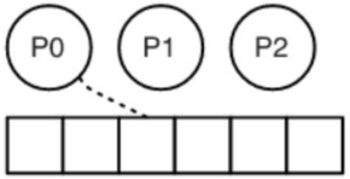
IPDPS 2024

Thomas Rolinger (UMD/NVIDIA)
Alan Sussman (UMD)
Contact: trolinger@nvidia.com

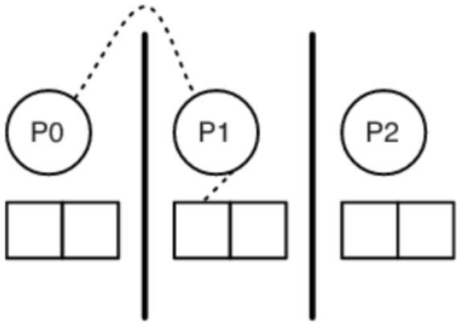
COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Background: Terminology and Motivation

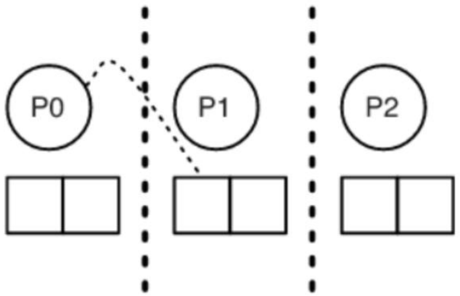
- **Partitioned Global Address Space (PGAS)** Model
  - Provides a (**logical**) shared-memory view on a **distributed-memory** system
  - **One-sided** communication (puts/gets) instead of sends/receives
    - Well-suited for applications with **irregular memory accesses**
  - Ex: **Chapel**, OpenSHMEM, UPC



Shared-memory (e.g., OpenMP)
just “**get**” the data



Message-passing (e.g., MPI)
matching sends/receives



PGAS
just “**get**” the data

Problem: Productivity vs. Performance

SpMV in PGAS (Chapel)

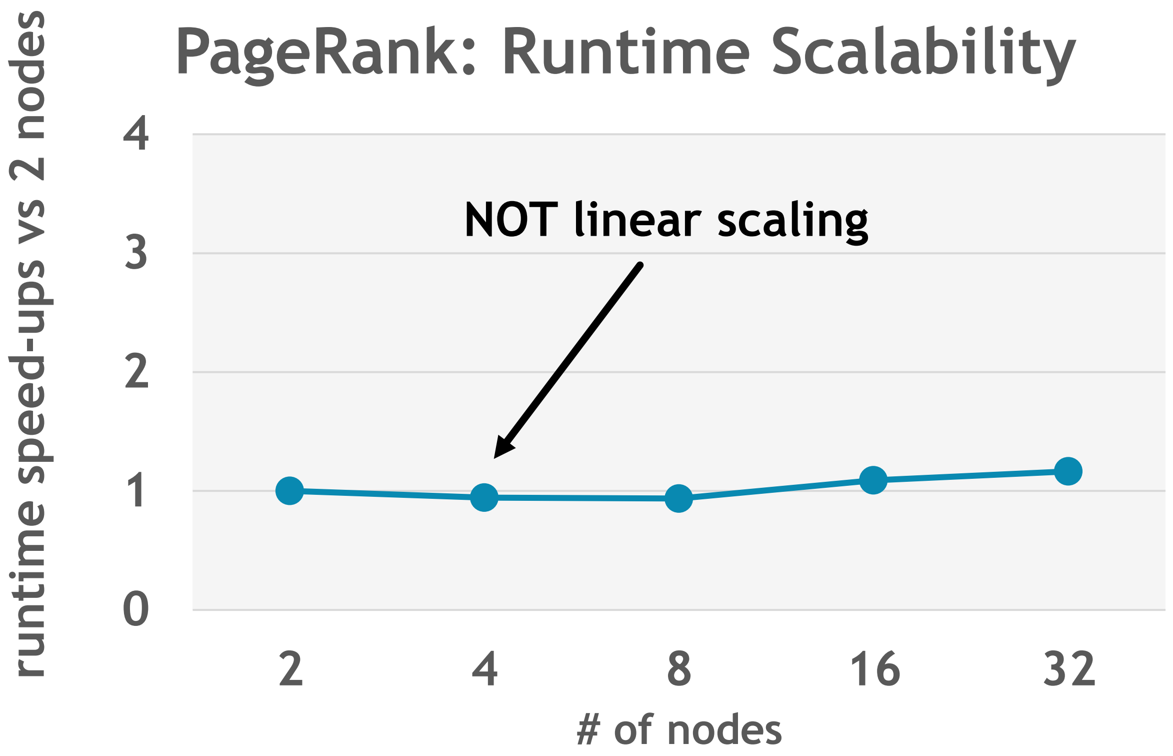
```
forall row in Rows {
  const id = row.id;
  var accum : real = 0;
  for k in row.columnOffset {
    accum += values[k] * x[col_idx[k]];
  }
  b[id] = accum;
}
```

PGAS code is very similar to shared-memory code, but is **distributed-memory** parallel

Problem: Productivity vs. Performance

SpMV in PGAS (Chapel)

```
forall row in Rows {
  const id = row.id;
  var accum : real = 0;
  for k in row.columnOffset {
    accum += values[k] * x[col_idx[k]];
  }
  b[id] = accum;
}
```

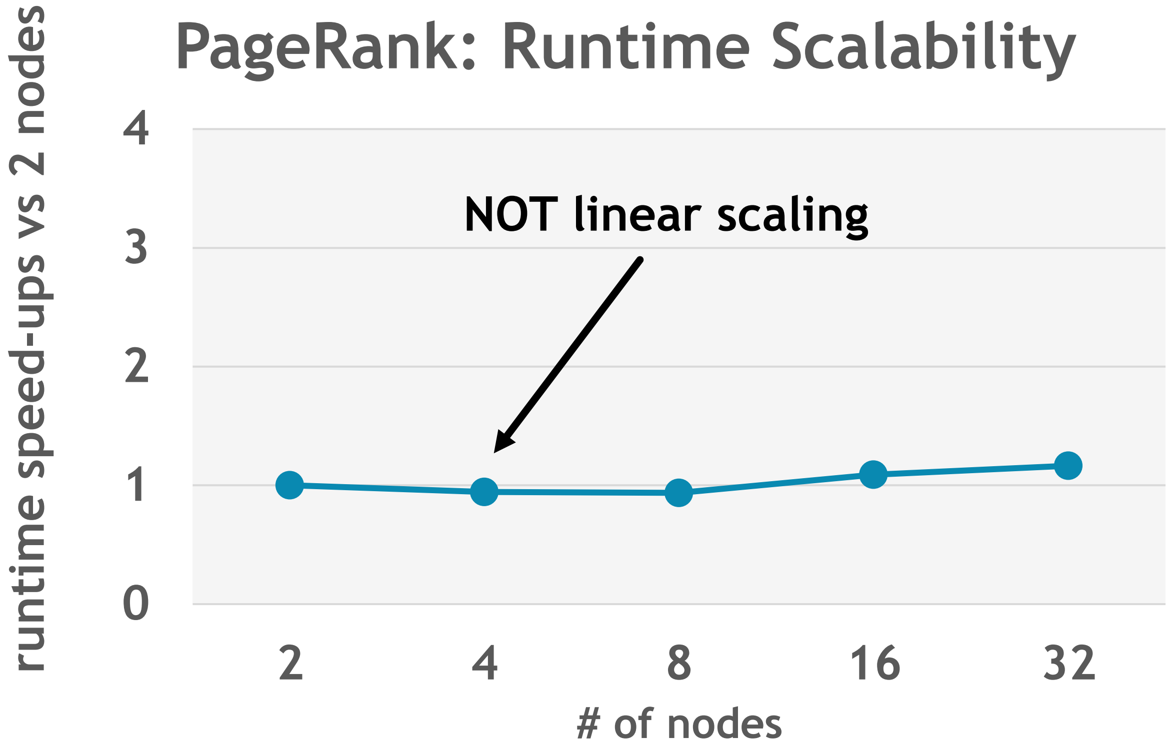


Problem: Productivity vs. Performance

SpMV in PGAS (Chapel)

```
forall row in Rows {
  const id = row.id;
  var accum : real = 0;
  for k in row.columnOffset {
    accum += values[k] * x[col_idx[k]];
  }
  b[id] = accum;
}
```

x is a distributed array



Performance issues due to ***fine-grain remote communication***

PGAS model "**encourages**" programmers to write code that exhibits fine gain communication

Problem: Productivity vs. Performance

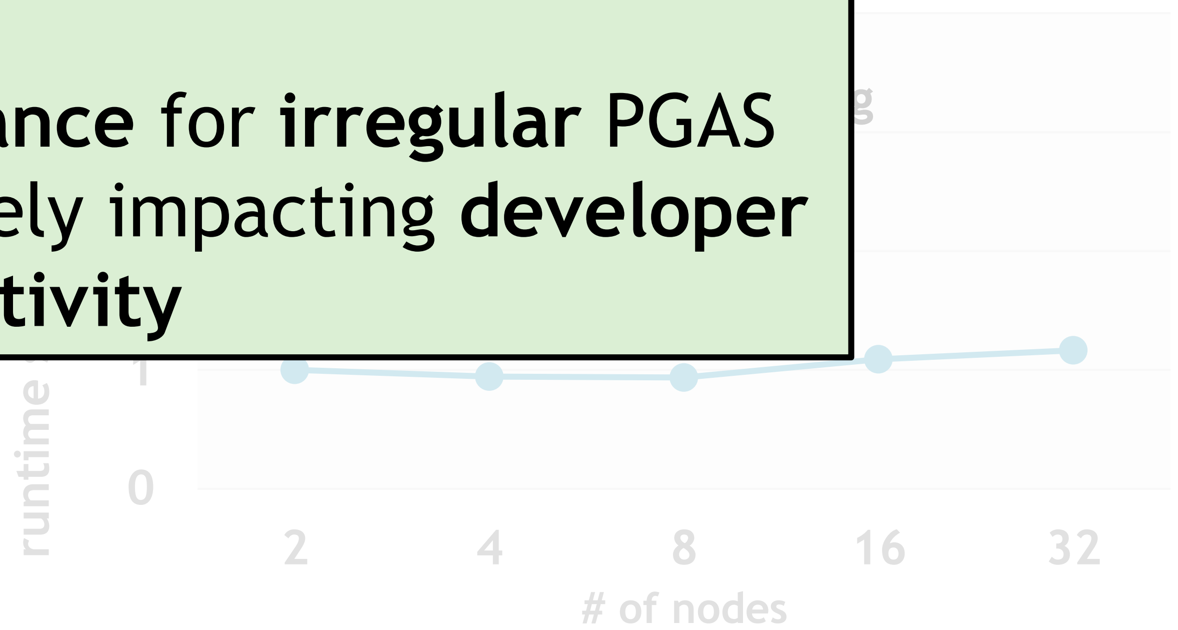
SpM

```
forall row in
  const id =
  var accum :
  for k in ro
    accum +=
  }
  b[id] = acc
}
```

x is a distributed array

Goal of this work:

Achieve **better performance** for **irregular** PGAS programs without negatively impacting **developer productivity**



Performance issues due to ***fine-grain remote communication***

PGAS model "**encourages**" programmers to write code that exhibits fine gain communication

Problem: Productivity vs. Performance

Goal of this work:

Achieve **better performance** for **irregular** PGAS programs without negatively impacting **developer productivity**

Our Approach:

Automatically improve performance via **runtime** optimization

Outline

- Optimization: **Adaptive Remote Prefetching**
- Performance Evaluation

Outline

- Optimization: **Adaptive Remote Prefetching**
- Performance Evaluation

Optimization: Adaptive Remote Prefetching

- **High level idea:**
  - Target **A[B[i]]** access patterns in **parallel loops**, where **A** is distributed and **i** is the loop index
  - Perform **non-blocking** reads for remote data that will be needed in future loop iteration
  - **Adapt** prefetching behavior as program **executes**

Optimization: Adaptive Remote Prefetching

- **High level idea:**
  - Target **A[B[i]]** access patterns in **parallel loops**, where **A** is distributed and **i** is the loop index
  - Perform **non-blocking** reads for remote data that will be needed in future loop iteration
  - **Adapt** prefetching behavior as program **executes**
- What are we prefetching into:
  - Chapel's **software/runtime-managed** cache for remote data
  - Provides mechanism to perform prefetches
  - Each core on a node has its own remote cache
    - And therefore, its **own prefetch distance/metrics**

Optimization: Adaptive Remote Prefetching (cont.)

- **Challenges:**
  1. Computing **prefetch distance**
  2. Determining when prefetching is **profitable** for performance
  3. Modifying the program to perform prefetching

Optimization: Adaptive Remote Prefetching (cont.)

- **Challenges:**
  1. Computing **prefetch distance**
  2. Determining when prefetching is **profitable** for performance
  3. Modifying the program to perform prefetching

Challenges **(1)** and **(2)** impact **performance** and are difficult because **static decisions** are not good enough.

Challenge **(3)** can impact **developer productivity** by requiring additional effort to **manually** apply the optimization.

Optimization: Adaptive Remote Prefetching (cont.)

- Challenges:
  1. Computing **prefetch distance**
  2. Determining when prefetching is **profitable** for performance
  3. Modifying the program to perform prefetching
- Our approach:
  - Use information at **runtime** to **dynamically** adjust prefetching behavior
    - Adjust prefetch distance, pause/resume prefetching
  - Develop a **compiler optimization** that **automatically** identifies candidate access patterns and then modifies the code to perform prefetching

Optimization: Adaptive Remote Prefetching (cont.)

- Challenges:
  1. Computing **prefetch distance**
  2. Determining when prefetching is **profitable** for performance
  3. Modifying the program to perform prefetching

Focus of this talk

- Our approach:
  - Use information at **runtime** to **dynamically** adjust prefetching behavior
    - Adjust prefetch distance, pause/resume prefetching
  - Develop a **compiler optimization** that **automatically** identifies candidate access patterns and then modifies the code to perform prefetching

Optimization: Prefetch Adjustment Heuristic

- Each thread/core will periodically “pause” its execution of the parallel loop independently to evaluate prefetching behavior:

Optimization: Prefetch Adjustment Heuristic

- Each thread/core will periodically “pause” its execution of the parallel loop independently to evaluate prefetching behavior:
  - **Late prefetches** → core had to wait for prefetched data to arrive
    - Prefetch distance **too small**
    - **Action:** increase prefetch distance

Optimization: Prefetch Adjustment Heuristic

- Each thread/core will periodically "pause" its execution of the parallel loop independently to evaluate prefetching behavior:
  - **Late prefetches** → core had to wait for prefetched data to arrive
    - Prefetch distance **too small**
    - **Action:** increase prefetch distance
  - **Unnecessary prefetches** → prefetched data already in remote cache
    - Memory access pattern is **sequential**/not irregular
    - **Action:** pause prefetching
      - Will **resume** prefetching if cache miss rate is too high

Outline

- Optimization: Adaptive Remote Prefetching
- Performance Evaluation

Performance Evaluation: Systems and Apps

Name	CPUs	Interconnect	Topology
FDR-56	Intel Xeon E5-2650	56 Gb/s IB	Single Switch
HDR-100	AMD EPYC 7763	100 Gb/s IB	Fat-tree
HDR-200	AMD EPYC 7713	200 Gb/s IB	Fat-tree
Cray XC	Intel Xeon E5-2699	Aries	Dragonfly
Cray EX	AMD EPYC 7662	Slingshot-11	Dragonfly

Distributed-memory Systems

Data sets

Name	Rows	Non-zeros	Density (%)
NAS_F	54M	55B	0.002
AGATHA_2015	184M	11.6B	3.4e−5
s-28	268M	8.5B	1.2e−5
MOLIERE_2016	30M	6.7B	7.3e−4
NAS_E	9M	6.6B	0.008
GAP-urand	134M	4.3B	2.4e−5
GAP-kron	134M	4.2B	2.3e−5
s-27	134M	4B	2.4e−5
com-Friendster	66M	3.6B	8.4e−5
mycielskian20	786K	2.7B	4.4e−1
s-26	67M	2B	4.8e−5
sk-2005	51M	1.9B	7.6e−5
it-2004	41M	1.2B	6.7e−5
webbase-2001	118M	1B	7.3e−6
uk-2005	40M	936M	6.0e−5
nlpkkt240	28M	803M	1.0e−4
arabic-2005	23M	640M	1.2e−4
mawi_201512020330	226M	480M	9.4e−7
kmer_V1r	214M	465M	1.0e−6
nlpkkt200	16M	465M	1.8e−4
kmer_A2a	171M	361M	1.2e−6
stokes	11M	349M	2.7e−4
Queen_4147	4M	334M	1.9e−3
mycielskian18	197K	301M	7.8e−1
uk-2002	19M	298M	8.7e−5
HV15R	2M	283M	7.0e−3
mawi_201512020130	129M	270M	1.6e−6
rgg_n_2_24_s0	17M	265M	9.4e−5
com-Orkut	3M	234M	2.5e−3
indochina-2004	7M	194M	3.5e−4
kron_g500-logn21	2M	182M	4.1e−3
mawi_201512020030	69M	143M	3.0e−6
vas_stokes_4M	4M	132M	6.9e−4
Bump_2911	3M	131M	1.5e−3
Cube_Coup_dt0	2M	129M	2.8e−3
rgg_n_2_23_s0	8M	127M	1.8e−4
Flan_1565	1.6M	119M	4.9e−3
hollywood-2009	1.1M	115M	8.9e−3
ML_Geer	1.5M	111M	4.9e−3
delaunay_n24	17M	101M	3.6e−5

Applications/kernels:

- IndexGather
- SpMV
- PageRank
- SSSP (delta-stepping)

Performance Evaluation: Systems and Apps

Name	CPUs	Interconnect	Topology
FDR-56	Intel Xeon E5-2650	56 Gb/s IB	Single Switch
HDR-100	AMD EPYC 7763	100 Gb/s IB	Fat-tree
HDR-200	AMD EPYC 7713	200 Gb/s IB	Fat-tree
Cray XC	Intel Xeon E5-2699	Aries	Dragonfly
Cray EX	AMD EPYC 7662	Slingshot-11	Dragonfly

Distributed-memory Systems

Data sets

Name	Rows	Non-zeros	Density (%)
NAS_F	54M	55B	0.002
AGATHA_2015	184M	11.6B	3.4e−5
s-28	268M	8.5B	1.2e−5
MOLIERE_2016	30M	6.7B	7.3e−4
NAS_E	9M	6.6B	0.008
GAP-urand	134M	4.3B	2.4e−5
GAP-kron	134M	4.2B	2.3e−5
s-27	134M	4B	2.4e−5
com-Friendster	66M	3.6B	8.4e−5
mycielskian20	786K	2.7B	4.4e−1
s-26	67M	2B	4.8e−5
sk-2005	51M	1.9B	7.6e−5
it-2004	41M	1.2B	6.7e−5
webbase-2001	118M	1B	7.3e−6
uk-2005	40M	936M	6.0e−5
nlpkkt240	28M	803M	1.0e−4
arabic-2005	23M	640M	1.2e−4
mawi_201512020330	226M	480M	9.4e−7
kmer_V1r	214M	465M	1.0e−6
nlpkkt200	16M	465M	1.8e−4
kmer_A2a	171M	361M	1.2e−6
stokes	11M	349M	2.7e−4
Queen_4147	4M	334M	1.9e−3
mycielskian18	197K	301M	7.8e−1
uk-2002	19M	298M	8.7e−5
HV15R	2M	283M	7.0e−3
mawi_201512020130	129M	270M	1.6e−6
rgg_n_2_24_s0	17M	265M	9.4e−5
com-Orkut	3M	234M	2.5e−3
indochina-2004	7M	194M	3.5e−4
kron_g500-logn21	2M	182M	4.1e−3
mawi_201512020030	69M	143M	3.0e−6
vas_stokes_4M	4M	132M	6.9e−4
Bump_2911	3M	131M	1.5e−3
Cube_Coup_dt0	2M	129M	2.8e−3
rgg_n_2_23_s0	8M	127M	1.8e−4
Flan_1565	1.6M	119M	4.9e−3
hollywood-2009	1.1M	115M	8.9e−3
ML_Geer	1.5M	111M	4.9e−3
delaunay_n24	17M	101M	3.6e−5

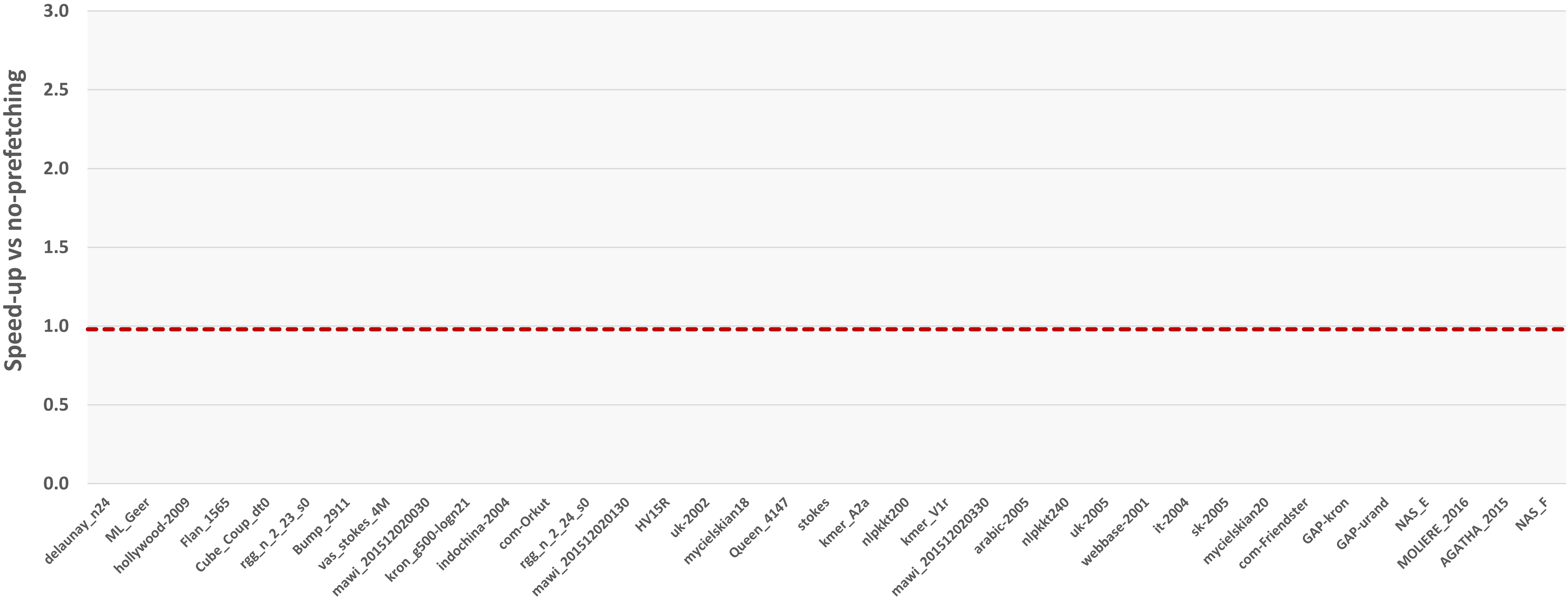
Applications/kernels:

- IndexGather
- SpMV
- PageRank
- SSSP (delta-stepping)

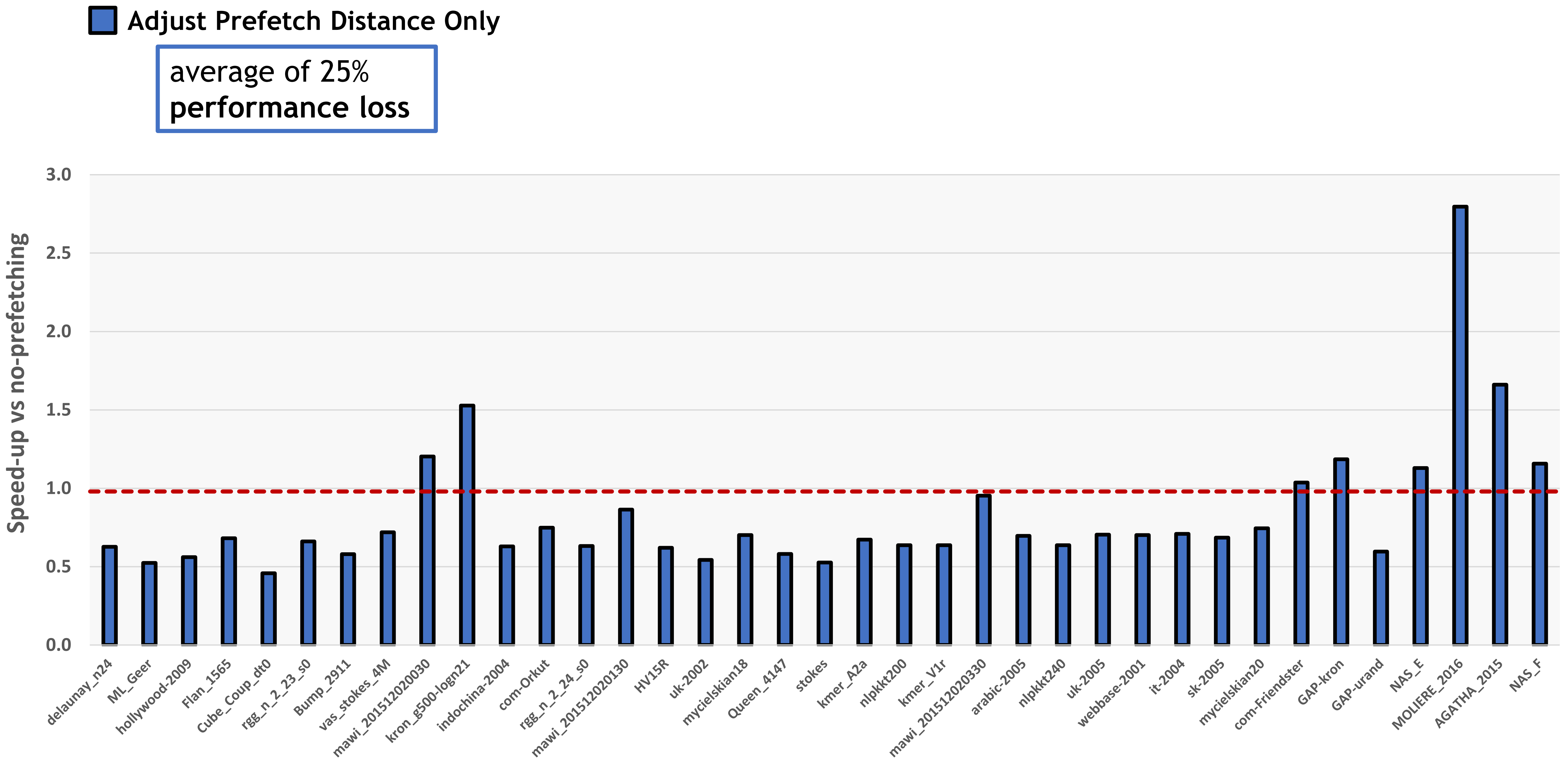
In this talk: impact of different adaptive decisions for SpMV on the HDR-200 system

See our paper for many more experiments and results

SpMV Speed-ups vs. No-prefetching – 64 nodes HDR 200



SpMV Speed-ups vs. No-prefetching - 64 nodes HDR 200

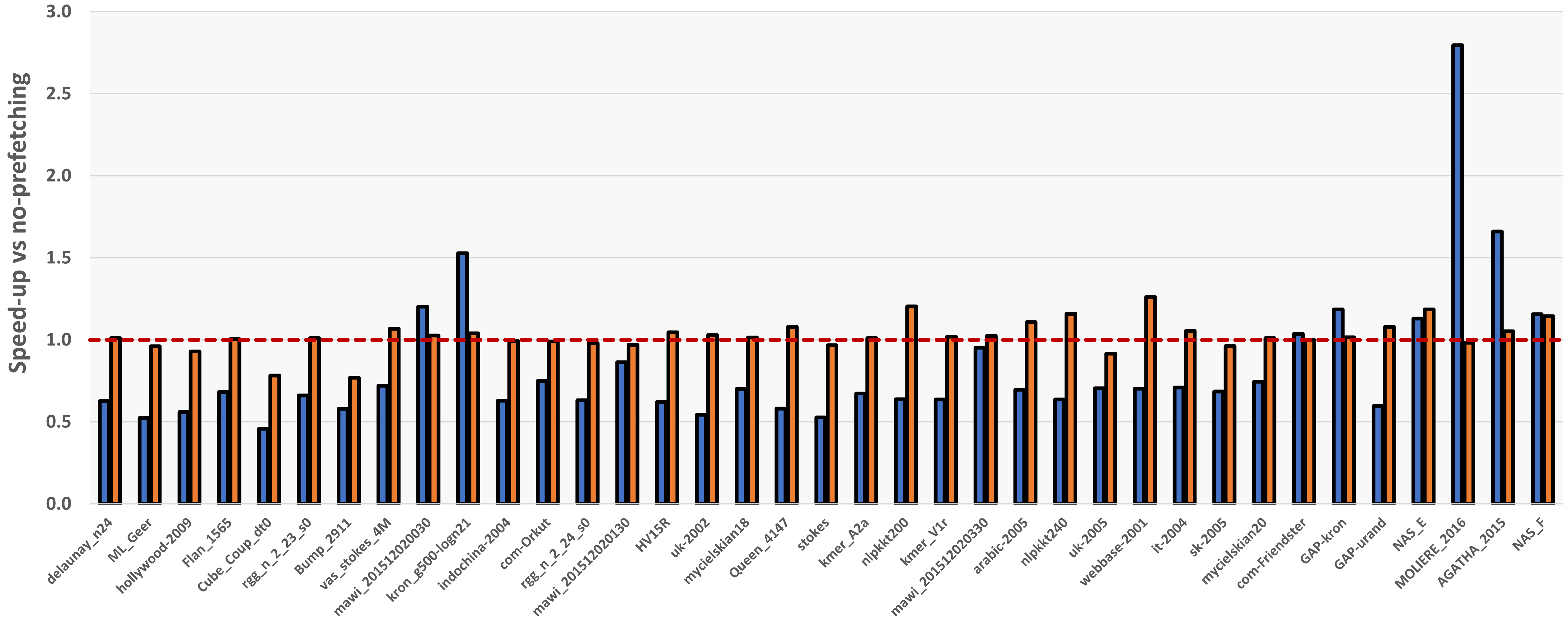


SpMV Speed-ups vs. No-prefetching - 64 nodes HDR 200

Adjust Prefetch Distance Only | Adjust + Pause Prefetching

average of 25% **performance loss**

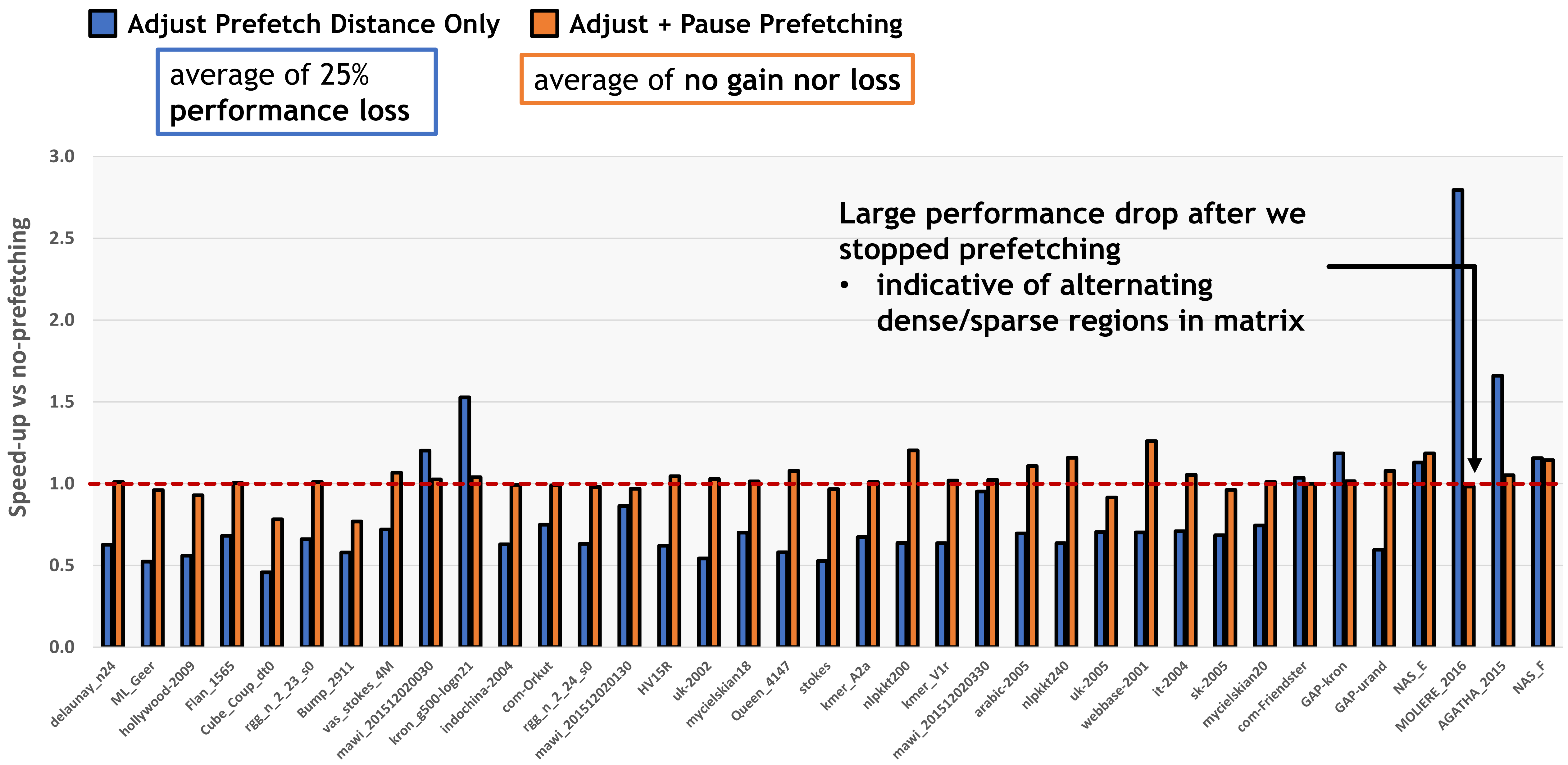
average of **no gain nor loss**



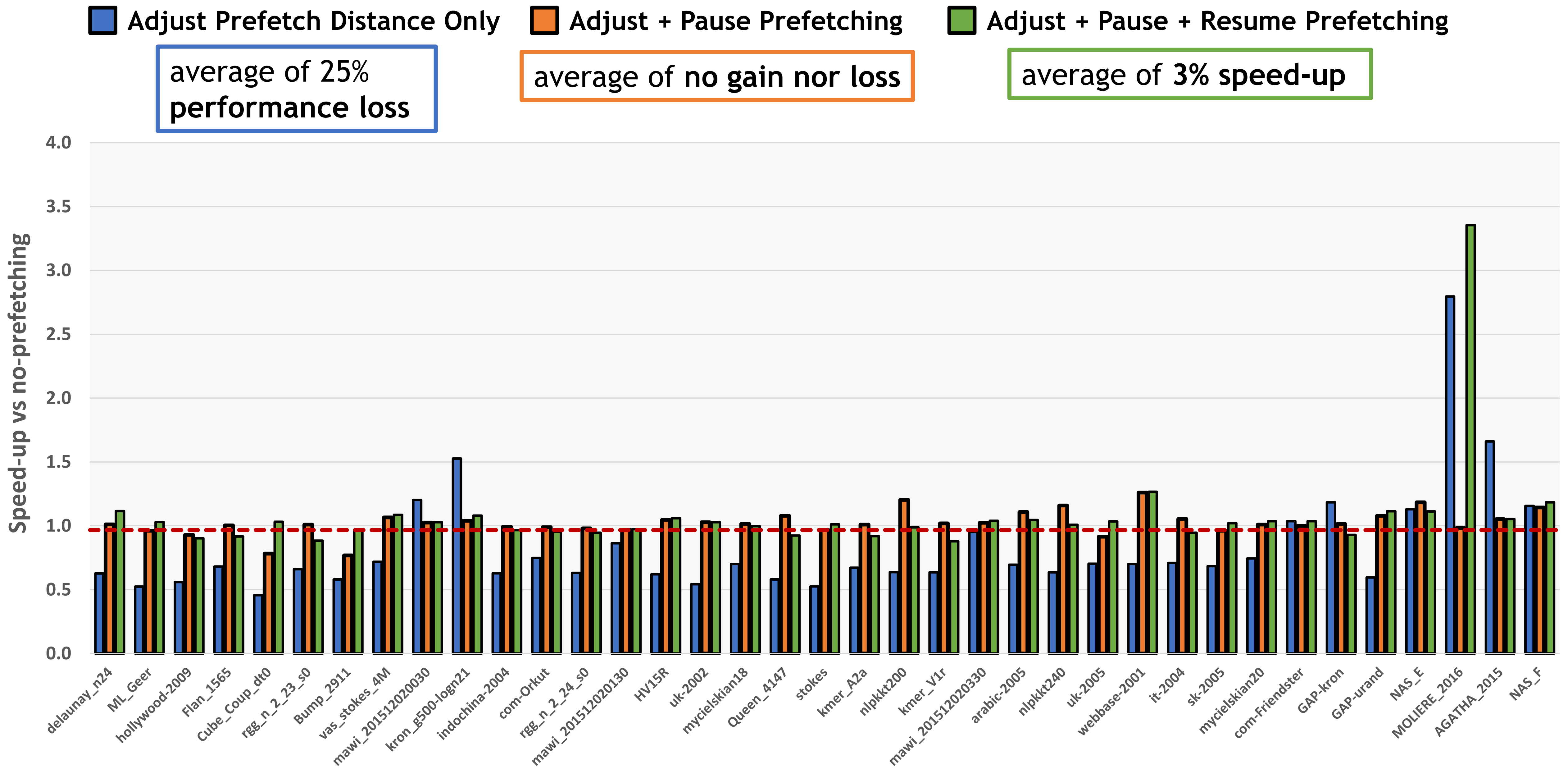
SpMV Speed-ups vs. No-prefetching - 64 nodes HDR 200

average of 25% **performance loss**

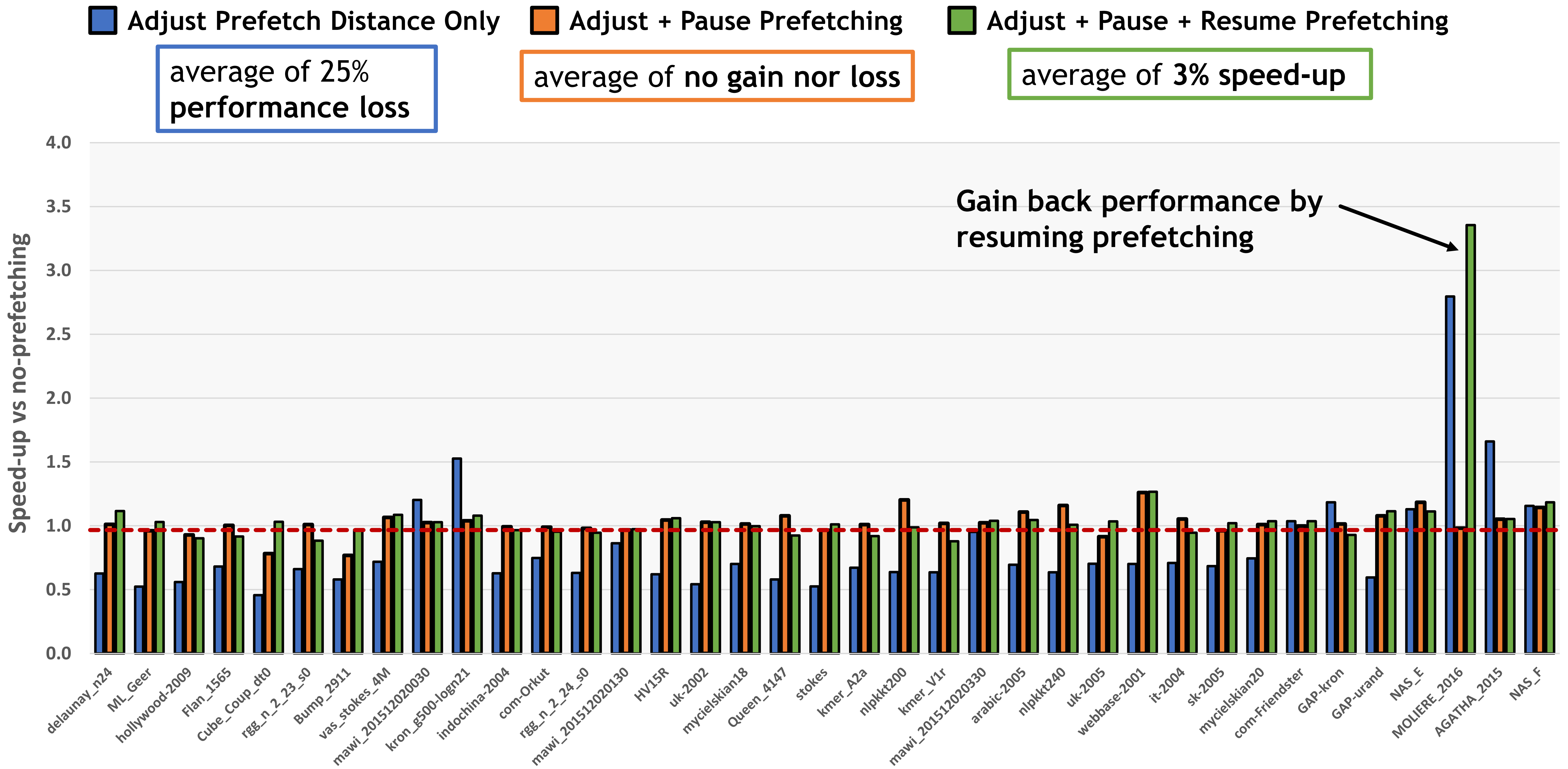
average of **no gain nor loss**



SpMV Speed-ups vs. No-prefetching - 64 nodes HDR 200



SpMV Speed-ups vs. No-prefetching - 64 nodes HDR 200



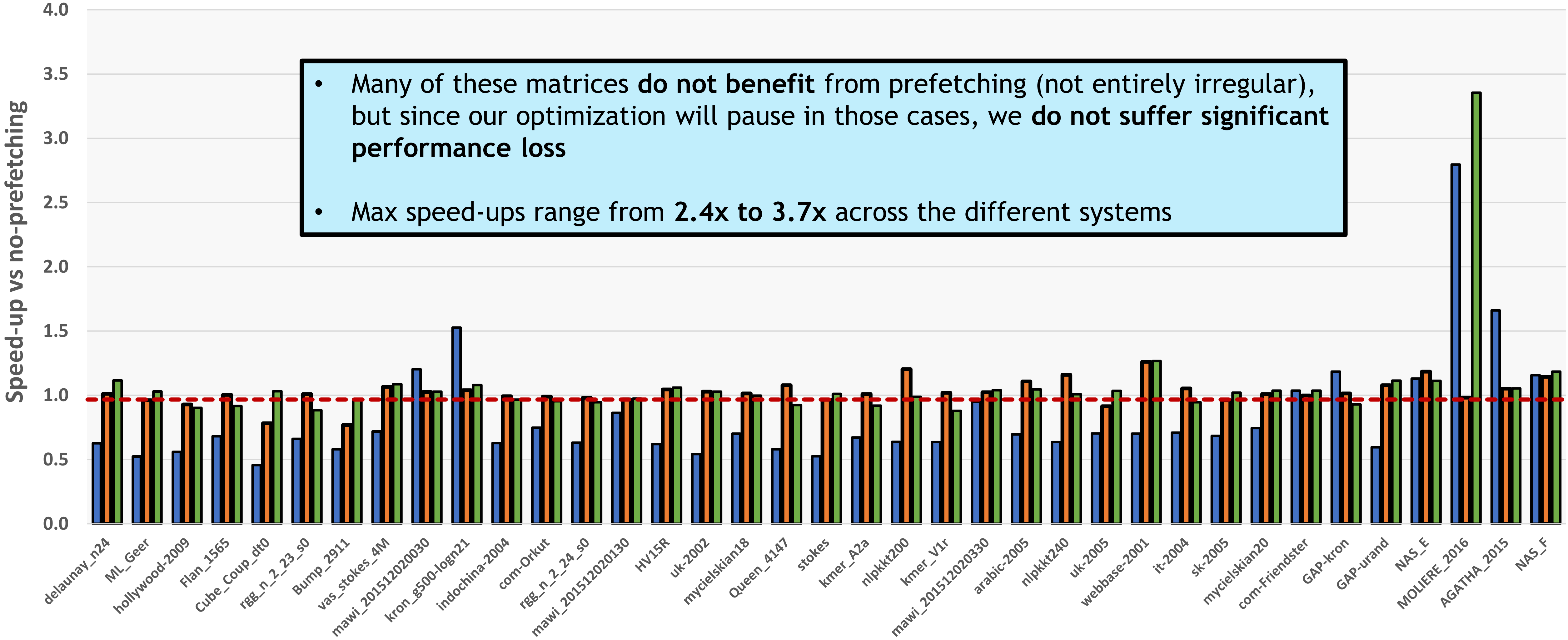
SpMV Speed-ups vs. No-prefetching - 64 nodes HDR 200

Adjust Prefetch Distance Only — average of 25% **performance loss**

Adjust + Pause Prefetching — average of **no gain nor loss**

Adjust + Pause + Resume Prefetching — average of **3% speed-up**

- Many of these matrices **do not benefit** from prefetching (not entirely irregular), but since our optimization will pause in those cases, we **do not suffer significant performance loss**
- Max speed-ups range from **2.4x to 3.7x** across the different systems



Performance Evaluation: Final Remarks

- Performance losses are **avoided** due to **adaptive** behavior
- Performance **gains** can be significant:
  - IndexGather: 292x, SpMV: 3.7x, PageRank: 1.6x, SSSP: 2x

Performance Evaluation: Final Remarks

- Performance losses are **avoided** due to **adaptive** behavior
- Performance **gains** can be significant:
  - IndexGather: 292x, SpMV: 3.7x, PageRank: 1.6x, SSSP: 2x
- Adaptive prefetching is built into larger **framework** for other compiler optimizations for irregular memory accesses
  - E.g., data replication via inspector-executor (LCPC`22)
  - Adaptive prefetching can be applied in situations where inspector-executor cannot
    - Inspector is too costly, replicated data is not read-only, etc.

See our paper for additional experiments and results