Stream Floating: Enabling Proactive and Decentralized Cache Optimizations

Zhengrong Wang¹, Jian Weng¹, Jason Lowe-Power², Jayesh Gaur³, Tony Nowatzki¹

¹UCLA, ²UC Davis, ³Intel

Feb. 2021
while (i < N) 
  sum += A[i];
  i++;

Reactive Cache System:
72% cached lines without reuse.
Up to 30% extra control NoC traffic.
Missing holistic view of the program behavior.
Streams to Bridge the Gap → Proactive Cache

while (i < N)
    sum += A[i];
    i++;

Requesting Core

Remote L3 Cache

Proactive Cache System:
Driven by streams, proactively transfer.

Pattern
Reuse?
Workset?

Pattern
Reuse?
Workset?

Improve

Prefetch
Replace
Bypass
Stream Floating → Proactive Cache System

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reactive Response</td>
<td>B[A[i]] Transfer</td>
<td></td>
</tr>
<tr>
<td>Stream Request</td>
<td>Decentralized Requests</td>
<td>Multicast Transfer</td>
</tr>
<tr>
<td>Proactive Transfer</td>
<td>Proactive Transfer</td>
<td>Stream Request</td>
</tr>
<tr>
<td>+ Less Request Traffic</td>
<td>+ Less Request (decentral)</td>
<td>+ Less Response (multicast)</td>
</tr>
<tr>
<td>+ Lower Latency</td>
<td>+ Less Response (subline)</td>
<td>+ Lower LLC Bandwidth</td>
</tr>
<tr>
<td>Benefits Floating (ours) Conventional</td>
<td>Benefits Floating (ours) Conventional</td>
<td>Benefits Floating (ours) Conventional</td>
</tr>
<tr>
<td>+ Less Request Traffic</td>
<td>+ Less Request (decentral)</td>
<td>+ Less Response (multicast)</td>
</tr>
<tr>
<td>+ Lower Latency</td>
<td>+ Less Response (subline)</td>
<td>+ Lower LLC Bandwidth</td>
</tr>
<tr>
<td>+ Lower Latency (proactive)</td>
<td>+ Less Thrashing (non-cached streams)</td>
<td>+ Less Thrashing (non-cached streams)</td>
</tr>
</tbody>
</table>
Stream Floating Implementation

Original C Code

```c
int i = 0;
while (i < N) {
    sum += B[A[i]];
    i++;
}
```

Stream Pseudo-Code

```c
stream_cfg(B[A[]]);
while (i < N) {
    sum += stream_ld();
    stream_step();
}
```

- **Static + dynamic information → floating decision.**
  - e.g. reuse distance, aliased stores, hit rate in L2.

- **Offload entire stream pattern with one message.**

- **Proactively transfer stream data.**
  - Stream data is not cached (bypass coherence).
  - Extend MESI with GetUncached request.

- **Automatically migrate to next bank.**
  - Keep streaming until no credits.
  - Released by StreamEnd message.

- **Weak consistency w. aliasing detection.**
  - Strong consistency is also possible.
  - w. stream-grain coherence.

- **Indirect requests from remote SE₁₃.**
  - Decentralized address generation.
NoC Traffic Breakdown

Baseline: No HW prefetcher or stream support.
L1BG-L2ST: L1 Bingo spatial + L2 stride prefetcher.
SF-X: Stream-floating with each optimization enabled.

1.39× speedup over hardware prefetcher.
36% NoC traffic reduction.
Conclusion: Streams Enables Proactive Cache

while (i < N)
    sum += A[i];
    i++;

Proactive Cache System:
Driven by streams, proactively transfer.
Explore rich ISA semantics in the cache system.

Remote L3 Cache

Stream A[]

Pattern
Reuse?
Workset?
...
Stream Floating: Enabling Proactive and Decentralized Cache Optimizations

Zhengrong Wang¹, Jian Weng¹, Jason Lowe-Power², Jayesh Gaur³, Tony Nowatzki¹

¹UCLA, ²UC Davis, ³Intel

Feb. 2021
while (i < N) 
  sum += A[i];
  i++;

Remote L3 Cache

Requesting Core

Reactive Cache System:
72% cached lines without reuse.
Up to 30% extra control NoC traffic.
Missing holistic view of the program behavior.
while (i < N) 
  sum += A[i];
  i++;

Stream A[]

Pattern
Reuse?
Workset?
...

Remote L3 Cache

Requesting Core

Proactive Cache System:
Driven by streams, proactively transfer.

Improve

Prefetch
Replace
Bypass
...

Pattern
Reuse?
Workset?
...

Stream A[]

Provide

Infer

Stream A[]

Infer

Stream A[]
Stream Floating → Proactive Cache

- Expose stream patterns without reuse to shared L3 banks.
- Proactive cache system that driven by streams.
  - One request for an entire stream.
  - Accurate prefetch.
  - Simplified coherence protocol.
- \(1.39\times\) speedup over hardware prefetcher.
- 36% NoC traffic reduction.
Outline

• Insights and Opportunities
• Stream Floating Implementation
• Coherence and Consistency
• Evaluation
Outline

• Insights and Opportunities
• Stream Floating Implementation
• Coherence and Consistency
• Evaluation
Overheads of Caching Data without Reuse

![Diagram showing L2 Eviction and Flits Injected for different datasets with two categories: NoReuse and NoReuse-Stream.](image)

- **L2 Eviction**
  - conv3D, mv, bfs, b-tree, cfd, hotspot, hotspot3D, nn, nw, particlefilter, pathfinder, srad, avg.
  - NoReuse and NoReuse-Stream categories.

- **Flits Injected**
  - conv3D, mv, bfs, b-tree, cfd, hotspot, hotspot3D, nn, nw, particlefilter, pathfinder, srad, avg.
  - NoReuse-Ctrl and NoReuse-Data categories.
Conventional vs. Stream Floating

**Conventional Affine Access Pattern** (eg. A[i])
- Line Requests
- Reactive Response

**Indirect Access Pattern** (eg. B[A[i]])
- A[i] Transfer
- B[A[i]] Transfer

**Confluence Pattern** (eg. A[i] on two cores)
- A[i] Transfer
- A[i] Transfer

**Benefits Floating (ours) Conventional**
- + Less Request Traffic
- + Lower Latency

**Benefits Floating (ours)**
- + Less Request Traffic
- + Lower Latency

**Decentralized Requests**
- + Less Request (decentral)
- + Less Response (subline)

**Lower LLC Bandwidth**
- + Lower Latency (proactive)
- + Less Thrashing (non-cached streams)
Outline

• Insights and Opportunities

• Stream Floating Implementation
  • What to offload?
  • How to offload?
  • When to offload?

• Coherence and Consistency

• Evaluation
What to Offload: Streams

- Stream: A decoupled sequence of values/addresses [ISCA’ 19].
- Explicitly embedded in the ISA.
- Memory order defined by the first usage of the value.

### Original C Code

```c
int i = 0;
while (i < N) {
  sum += a[b[i]];
  i++;
}
```

### Decoupled Stream Pseudo Assembly

```assembly
stream_cfg();
while (i < N) {
  sum += stream_load();
  stream_step();
}
```

Use the value.

Configure.

Advance.
Stream Engines

**SE\textsubscript{CORE}:**
Mange stream configuration and issue stream requests. Make and cancel offload decisions.

**SE\textsubscript{L2}:**
Buffer stream data and match it with requests from SE\textsubscript{CORE}. Issue flow control credits to remote L3 bank (SE\textsubscript{L3}).

**SE\textsubscript{L3}:**
Generate requests and stream back data to SE\textsubscript{L2}. Receives control messages from SE\textsubscript{L2}, e.g. flow credits.
How to Offload: Configure Affine Stream A[i]

$\text{SE}_{\text{CORE}}$ configures $\text{SE}_{\text{L2}}$ with the affine stream pattern $A[i]$.

$\text{SE}_{\text{L2}}$ allocates the stream buffer, and forward configuration to $\text{SE}_{\text{L3}}$ where $A[0]$ is.
How to Offload: Proactively Stream Data to Core

$SE_{L3}$ generates requests of $A[i]$, translates (L2 TLB) and sends to L3 cache controller.
Data responses are buffered at $SE_{L2}$ and later drained by requests from $SE_{CORE}$. 
How to Offload: Flow Control and Migration

$SE_{L2}$ sends out credits to $SE_{L3}$ at coarse-granularity, further reduce traffic overhead.

Streams migrate to the next bank, and keep streaming until no credits.

Slightly increase interleave granularity to avoid too-frequent migrations.
How to Offload: End (Sink) the Stream

\( \text{SE}_{\text{CORE}} \) terminates streams by sending out StreamEnd messages.

\( \text{SE}_{\text{L3}} \) can directly release streams with known length.

**Single Request – Multiple Data**

Minimal control traffic for floating streams.

Deep and accurate prefetching.
How to Offload: Indirect Stream B[A[i]]

Associate indirect stream B[A[i]] with A[i].

SE_{L3} can directly send out indirect requests to the target L3 cache controller.

Decentralized Address Generation
Generate indirect request at remote L3. Only transfer the required subline data.
How to Offload: Stream Confluence

Neighboring cores may be requesting the same piece of data. SE_L3 can easily perform pattern matching and multicast data to different cores.
When to Offload: Detect Floating Candidates

• Target: Streams with no reuse in the private cache, or aliasing.
• Static Information: Compiler Analysis.
  • E.g. are there writes to the address?
• Dynamic Information: Stream History Table.
• Only offload when passed both static & dynamic check.
  • SE\text{CORE} can early terminate a floating stream, e.g. found aliasing.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>Stream id</td>
<td>request</td>
<td># stream requests</td>
</tr>
<tr>
<td>reuse</td>
<td># priv. cache reuses</td>
<td>miss</td>
<td># priv. cache misses</td>
</tr>
<tr>
<td>aliased</td>
<td>Aliased with stores</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE II: Stream History Table
Outline

• Insights and Opportunities
• Stream Floating Implementation
• Coherence and Consistency
  • Support Weak Consistency
  • Support Strong Consistency
• Evaluation
Support Weak Consistency: Uncached Stream Data

• Limit streams in synchronization-free region.
• Bypass coherence protocol: stream data is not cached.
  • Extend MESI protocol with uncached requests (GetU).
• Details about aliasing detection in the paper.
• Strong consistency with stream-grain coherence.
Outline

• Insights and Opportunities
• Stream Floating Implementation
• Coherence and Consistency
• Evaluation
Configurations

• LLVM-based compiler to recognize streams and transform programs.
• Gem5 20.0 cycle-level execution-driven simulator.
• 12 data processing workloads from Rodinia and micro kernels.
  • Parallelized with OpenMP, with AVX-512 enabled.
• Configurations (see paper for details):
  • 8x8 mesh topology, 3-level MESI, 32kB L1 I/D, 256kB L2, 1MB L3.
  • Base: Baseline cores without prefetcher or stream support.
  • L1Stride-L2Stride: Stride prefetcher at both L1 and L2 cache level.
  • L1Bingo-L2Stride: Bingo spatial prefetcher at L1 and stride prefetcher at L2.
  • SS: Stream-specialized processor (stream support at core) [ISCA’ 19].
  • SF: Stream floating, (offload stream to cache) [this work].
Overall Speedup with IO4 and OOO8
LLC Request Breakdown

Requests to L3 Cache

- conv3D
- mv
- bfs
- b+tree
- cfd
- hotspot
- hotspot3D
- nn
- nw
- particlefilter
- pathfinder
- srad
- avg.

- Core-Normal
- Core-Stream
- Float-Affine
- Float-Indirect
- Float-Confluence
Energy vs. Speedup

- IO4
- OOO4
- OOO8

Relative Energy vs. Speedup

This way better
Conclusion: Streams Enables Proactive Cache

while (i < N)
    sum += A[i];
    i++;

Remote L3 Cache

Pattern
Reuse?
Workset?
...