Caching Puts and Gets in a PGAS Language Runtime

Michael Ferguson
Cray Inc.

Daniel Buettner
Laboratory for Telecommunication Sciences

September 17, 2015
Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.
TRAIN LATENCY
(8 HOUR TRIP, 60 TON CARS, 60 SEC/CAR)
TRAIN BANDWIDTH

Number of Cars

Bandwidth tons/min

12
9
6
3
0
INFINIBAND (IB) LATENCY

* with small 10-node cluster, QDR IB
INFINIBAND (IB) BANDWIDTH

* with small 10-node cluster, QDR IB

Max BW: 5000 MB/s
AGGREGATION

OVERLAP

CACHE HELPS WITH BOTH!
BACKGROUND: MEMORY MODEL ALLOWS PREFETCH AND WRITE-BEHIND
Memory model for C11, C++11, Chapel: data race free programs are sequentially consistent

A RACY PROGRAM

Thread 1
x = 42;
notify = 1;

Thread 2
while 0 == notify { /* wait */ }
compute_with(x);
A RACY PROGRAM

Thread 1
x = 42;
notify = 1;

Thread 2
while 0 == notify { /* wait */ }
compute_with(x);

Thread 1
r1 = 42;
notify = 1; x = r1;

Thread 2
r2 = notify; while 0 == r2 { /* wait */ }
compute_with(x);
compiler or processor
Compiler *and* processor would like to start loads earlier in order to hide memory latency. We’ll call that *prefetch*.
Compiler *and* processor would like to complete stores later in order to hide memory latency. We’ll call that *write behind*.

\[ B[i] = \ldots \]

![Diagram](image)
- Overlap loads (start early)
- Reuse values from earlier load
- Aggregate loads (cache lines)

- Overlap stores (finish later)
- Aggregate many stores into a single store

prefetch

load x

store y

write behind
REMEMBER THE RACY PROGRAM?

Thread 1
x = 42;
notify = true;

Thread 2
while 0 == notify { /* wait */ }
compute_with(x);

 Thread 1
r1 = 42;
notify = 1; x = r1;

Thread 2
r2 = notify; while 0 == r2 { /* wait */ }
compute_with(x);

compiler or processor
prefetch → acquire
  
  load x
  
  store y
  
  release
  
  atomic, sync
  provide both
  acquire, release
  
  write behind
COMMUNICATION OPTIMIZATION
- Overlap GETs (start early)
- Reuse values from earlier GET
- Aggregate GETs (cache lines)

- Overlap PUTs (finish later)
- Aggregate many PUTs into a single PUT

prefetch

load x
GET x

store y
PUT y

write behind
FIXING IT WITH A CACHE
CACHE FOR REMOTE DATA

- Goal: communication aggregation and overlap
- Bonus points: avoiding repeated communication

- Software cache in Chapel's runtime
- One cache per pthread
- Write-back cache with dirty bits
CACHE COHERENCY

- Simple, local coherency
- Discard all cached data on *acquire*
- Wait for pending operations on a *release*
- Strategy used in related work with UPC
## CACHE FEATURES

<table>
<thead>
<tr>
<th>Feature</th>
<th>Overlap</th>
<th>Aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong></td>
<td>PUT</td>
<td>GET</td>
</tr>
<tr>
<td>Do PUTs in background</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Start one PUT per contiguous written region</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Round GETs up to 64-byte cache lines</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Sequential read-ahead</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Programmer-provided prefetch hints*</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
WEAK MEMORY CONSISTENCY?

1. x starts at 0;
   ...
   if someOption then
2.   x = 2;
   if someOtherOption then
3.   x = 3;
4. return x;
WEAK MEMORY CONSISTENCY?

1 \ x \text{ starts at } 0;
\ldots
\ldots
2 \ \text{PUT} \ 2 \text{ into } x;
\ldots
3 \ \text{PUT} \ 3 \text{ into } x;
4 \ \text{GET} \ x;

Chapel

result must be 3

OpenSHMEM

result could be 0, 2, or 3
COMPILER OPTIMIZATION?

for i in 1..100 {
    // PUT into B
    B[f(i)] = i;
}

Can the compiler prove these PUTs do not overlap?

PUT 1 into B[f(1)]

PUT 2 into B[f(2)]

PUT 3 into B[f(3)]
for i in 1..100 {
    // PUT into B
    B[f(i)] = i;
}

With a cache, conflicting access is handled at runtime.
OVERLAPPING GETS

```plaintext
var A:[1..n] int;
on Locales[1] {
    var sum:int;
    for i in 1..n do
        sum += A[f(i)]
}
```

We would like to overlap the GETs for A[f(i)] with each other
var A: [1..n] int;
on Locales[1] {
    var sum: int;
    var h: [0..k] handles;
    var bufs: [0..k] int;
    // Warm up loop
    for i in 1..k {
        // nonblocking GET A[f(i)] into bufs[i%k]
        h[i%k] = get_nb(bufs[i%k], A[f(i)])
    }
    for i in 1..n {
        wait (h[i%k]);
        sum += bufs[i%k];
        if i+k<=n {
            // nonblocking GET A[f(i+k)] into bufs[(i+k)%k]
            h[(i+k)%k] = get_nb(bufs[(i+k)%k], A[f(i+k)])
        }
    }
}

Explicit overlap is messy!
```plaintext
var A:[1..n] int;
on Locales[1] {
    var sum:int;
    // Optional warm up
    for i in 1..k do prefetch(A[f(i)]);
    for i in 1..n {
        if i+k <= n then prefetch(A[f(i+k)]);
        sum += A[f(i)]
    }
}
```

Much better!
for \(i\) in 1..n do
\[B[i] = \text{compute}(i);\]

\begin{align*}
\text{var localB: [1..n] int; for } & i \text{ in } 1..n \text{ do} \\
& \text{localB}[i] = \text{compute}(i); \\
& B = \text{localB};
\end{align*}

for \(i\) in 1..n do
\[\text{consume}(A[i]);\]

\begin{align*}
\text{var localA: [1..n] int = A; for } & i \text{ in } 1..n \text{ do} \\
& \text{consume}(\text{localA}[i]);
\end{align*}

Simple, cache aggregates

Manual optimization reduces portability
PERFORMANCE
TEST CONFIGURATIONS

- Cray XC30™ system with 50 nodes, Aries network
  - GASNet Aries: GASNet with the aries conduit
  - Cray uGNI: native uGNI support for Chapel
- Cray CS400™ system with 200 nodes, FDR InfiniBand
  - GASNet ibv: GASNet with the InfiniBand Verbs conduit

- v1.9+ is revision 5ba6639
- v1.11+ is revision 6c635a1
SYNTHETIC BENCHMARKS

![Graph showing speedup comparisons between different interfaces and operations (copy, rand-puts, rand-gets).]
APPLICATION BENCHMARKS

<table>
<thead>
<tr>
<th>Application</th>
<th>GASNet ibv v1.11+</th>
<th>GASNet Aries v1.11+</th>
<th>Cray uGNI v1.11+</th>
</tr>
</thead>
<tbody>
<tr>
<td>lulesh</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>miniMD</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTRANS</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSCA2.4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PREFETCH EXAMPLE

```plaintext
var A: [1..n] int;
on Locales[1] {
    var sum:int;
    // Optional warm up
    for i in 1..k do prefetch(A[f(i)]);
    for i in 1..n {
        if i+k <= n then prefetch(A[f(i+k)]);
        sum += A[f(i)]
    }
}
```
PREFETCH EXAMPLE

![Graph showing speedup vs. prefetch distance for different network technologies.]

- GASNet ibv v1.11+
- GASNet Aries v1.11+
- Cray uGNI v1.11+
- 1x
VS OPTIMIZATION

Speedup

GASNet ibv v1.9+
GASNet ibv v1.11+
GASNet Aries v1.9+
GASNet Aries v1.11+

1x

*for GASNet/Aries, lulesh improved 3.2x between v1.9+ and v1.11+
VS OPTIMIZATION

* with GASNet Aries v1.11+ configuration

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>C+cache</th>
<th>llvm</th>
<th>llvm+cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>lulesh</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>miniMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTRANS*10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time
Cache for Remote Data: providing communication overlap and aggregation since Chapel v 1.10!
Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Copyright 2014 Cray Inc.
Backup Slides
APPLICATION BENCHMARKS

**Speedup**

- **lulesh**: GASNet ibv v1.11+, GASNet Aries v1.11+, Cray uGNI v1.11+, 32 loc GASNet Aries v1.11+ (1x)
- **miniMD**: GASNet ibv v1.11+, GASNet Aries v1.11+, Cray uGNI v1.11+, 32 loc GASNet Aries v1.11+ (1x)
- **PTRANS**: GASNet ibv v1.11+, GASNet Aries v1.11+, Cray uGNI v1.11+, 32 loc GASNet Aries v1.11+ (1x)
- **SSCA2.4**: GASNet ibv v1.11+, GASNet Aries v1.11+, Cray uGNI v1.11+, 32 loc GASNet Aries v1.11+ (1x)
• **acquire** and **release** triggered by task or on statement spawn, join, start, and finish

```plaintext
release
on {
    acquire...
    release
}
acquire

sync {
    release
    begin {
        acquire
        ....
        release
    }
    acquire
```

LOOKING INSIDE
CACHE ENTRY

- node
- address
- readahead trigger
- min sequence number
- max put sequence number
- max prefetch sequence number

1024 byte cache page

64 byte cache lines

Valid Line Bits

Optional Dirty Bits
Inspired by “Two Level Tree Structure for Fast Pointer Lookup” by Hans J Boehm
CACHE DATA STRUCTURES

New Pages

Am LRU \rightarrow Ain \rightarrow Aout

Dirty LRU
Free Lists

2Q Queues
Operations Queue

per task:
last acquire sequence number
WRITE BEHIND

Write Recorded in Dirty Bits, Page added to Dirty Queue
Flushed on *release* or when there are too many dirty pages
GET with 2 earlier valid lines triggers synchronous readahead
The next GET triggers asynchronous readahead

ra skip=1 pg len = 1 pg

ra skip,len=0

ra skip=1 pg len =2 pg
GET here triggers more readahead

ra skip,len=0  ra skip=1 pg len =2 pg

ra skip=2 pg len =4 pg
INFINIBAND (IB) LATENCY

* with small 10-node cluster, QDR IB

Latency (ns)

8 16 32 64 128 256 512 1024

Request Size (bytes)

GASNET get
IB Benchmark
GASNET put
INFINIBAND (IB) BANDWIDTH

* with small 10-node cluster, QDR IB

Max BW: 5000 MB/s

Bandwidth MB/s

Request Size (bytes)
Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Copyright 2014 Cray Inc.
http://chapel.cray.com  chapel_info@cray.com  https://github.com/chapel-lang/chapel/