LLVM-based Communication Optimizations for Chapel

Chapel Lightning Talks BoF session at SC '14, New Orleans, LA

Akihiro Hayashi, Jisheng Zhao (Rice University)
Michael Ferguson (Laboratory for Telecommunication Sciences)
Vivek Sarkar (Rice University)
A Big Picture

Habanero-C, ...

LLVM-based Chapel compiler

- Use of address space feature of LLVM offers more opportunities for communication optimization than C generation

// Chapel
x = possibly_remoteData;

// C-Code generation
chpl_comm_get(&x, ...);

// LLVM IR Generation
%x = load i64 addrspace(100)* %xptr

Backend Compiler’s Optimizations (e.g. gcc –O3)

LLVM Optimizations (e.g. LICM, scalar replacement)

Pictures borrowed from 1) http://chapel.cray.com/logo.html
2) http://llvm.org/Logo.html
An optimization Example: Communication Optimization with the existing LLVM passes

(Pseudo-Code: Before LICM)
for i in 1..N {
   // POSSIBLY REMOTE GET
   %x = load i64 addrspace(100)* %xptr
   A(i) = %x;
}

(Pseudo-Code: After LICM)
// POSSIBLY REMOTE GET
%x = load i64 addrspace(100)* %xptr
for i in 1..N {
   A(i) = %x;
}

LICM = Loop Invariant Code Motion
An optimization Example: Bulk Transformation (Coalescing)

(Pseudo-Code: Before Bulk Transformation)

```plaintext
for i in 1..N {
    // POSSIBLY REMOTE GET
    ... = A(i);
}
```

(Pseudo-Code: After LICM)

```plaintext
var localA: [1..N] int;
localA = A; // Bulk Transfer
for i in 1..N {
    ... = localA(i);
}
```

- **Bulk Transformation**
  - Remote array access per each iteration
  - Create Local Buffer & Perform bulk transfer
  - Converted to Definitely-Local Access!
An Optimization Example: Locality Inference for avoiding runtime affinity checking

```haskell
proc habanero(ref x, ref y, ref z) {
    var p: int = 0;
    var A:[1..N] int;
    if (x == 0) {
        p = y;
    } else {
        local { p = z; }
    }
    z = A(0) + z;
}
```

- **A** is definitely-local
- **p** and **z** are definitely local

Definitely-local access! (avoid runtime affinity checking)
Results on Cray XC-30 (LLVM-unopt vs. LLVM-allopt)

4.6x performance improvement on average (6 applications, 1, 2, 4, 8, 16, 32, 64 locales)
Results on Westmere Cluster (LLVM-unopt vs. LLVM-allopt)

4.4x performance improvement on average
(6 applications, 1, 2, 4, 8, 16, 32 locales)
Conclusions

- LLVM-based Communication optimizations for Chapel
- Preliminary Evaluation with 6 applications
  - Cray-XC30 Supercomputer
    - 4.6x average performance improvement
  - Westmere Cluster
    - 4.4x average performance improvement

- Future Work
  - Extend for other languages
Backup slides
LLVM IR Generation from Chapel

C/C++ Frontend
Clang

C/C++, Fortran, Ada, Objective-C Frontend
dragonegg

UPC Compiler
chpl

LLVM Intermediate Representation (IR)

Analysis & Optimizations

x86 backend
x86 Binary

Power PC backend
PPC Binary

ARM backend
ARM Binary

PTX backend
GPU Binary
LLVM-based Communication Optimizations for Chapel

1) Wide pointer optimization (--llvm-wide-opt)
   - Utilize the existing optimization passes such as loop invariant code motion for the purpose of communication optimization (The Existing LLVM Passes)
   - Combine sequences of loads/stores on adjacent memory locations into a single memcpy (Aggregation Pass)

2) Bulk Transformation (Coalescing data accesses)
   - Create locale-local buffer
   - Insert bulktransfer and replace remote accesses with local buffer access

3) Locality optimization (Locality-Inference)
   - Transform possibly-remote access to definitely-local access at compile-time to avoid runtime affinity checking

These optimizations are built on top of address space feature of LLVM
Performance Evaluations: Platforms

- **Cray-XC30 Supercomputer @ NERSC**
  - Per Node information
    - Intel Intel Xeon E5-2695 @2.40GHz x 24 cores
    - 64GB of RAM
  - Interconnect
    - Cray Aries interconnect with Dragonfly topology

- **Westmere Cluster @ Rice**
  - Per Node information
    - Intel Xeon CPU X5660@2.80GHz x 12 cores
    - 48GB of RAM
  - Interconnect
    - Quad-data rated Infiniband
    - Mellanox FCA support
Performance Evaluations: Details of Compiler & Runtime

- Compiler:
  Chapel version 1.9.0.23154 (Apr. 2014)
  - LLVM 3.3

- Runtime:
  - GASNet-1.22.0
    - Cray-XC30: aries
    - Westmere Cluster: ibv-conduit
  - qthreads-1.10
    - Cray-XC30: 2 shepherds, 24 workers/shepherd
    - Westmere Cluster: 2 shepherds, 6 workers/shepherd
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Comm Kind</th>
<th>Cray XC-30</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LLVM-gopt</td>
<td>LLVM-allopt</td>
<td></td>
</tr>
<tr>
<td>Smith-Waterman</td>
<td>LOCAL.GET</td>
<td>63.6%</td>
<td>75.5%</td>
<td></td>
</tr>
<tr>
<td>Note: obtained with 18,560x19,200 input</td>
<td>REMOTE.GET</td>
<td>36.4%</td>
<td>36.7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOCAL.PUT</td>
<td>58.0%</td>
<td>58.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOTE.PUT</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Cholesky</td>
<td>LOCAL.GET</td>
<td>77.6%</td>
<td>87.9%</td>
<td></td>
</tr>
<tr>
<td>Note: obtained with 2,000x2,000 input</td>
<td>REMOTE.GET</td>
<td>84.7%</td>
<td>99.8%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOCAL.PUT</td>
<td>10.3%</td>
<td>10.8%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOTE.PUT</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>NPB EP</td>
<td>LOCAL.GET</td>
<td>58.6%</td>
<td>58.6%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOTE.GET</td>
<td>39.7%</td>
<td>39.7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOCAL.PUT</td>
<td>29.5%</td>
<td>58.8%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOTE.PUT</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Sobel</td>
<td>LOCAL.GET</td>
<td>74.6%</td>
<td>95.2%</td>
<td></td>
</tr>
<tr>
<td>Note: obtained with CLASS=B</td>
<td>REMOTE.GET</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOCAL.PUT</td>
<td>35.8%</td>
<td>68.3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOTE.PUT</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>SSCA2</td>
<td>LOCAL.GET</td>
<td>55.6%</td>
<td>56.2%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOTE.GET</td>
<td>60.9%</td>
<td>60.8%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOCAL.PUT</td>
<td>5.6%</td>
<td>3.8%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOTE.PUT</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Stream-EP</td>
<td>LOCAL.GET</td>
<td>70.6%</td>
<td>70.6%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOTE.GET</td>
<td>35.7%</td>
<td>35.7%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOCAL.PUT</td>
<td>17.3%</td>
<td>17.3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOTE.PUT</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. The amount of Chapel Comm APIs calls made by LLVM-gopt and LLVM-allopt relative to LLVM-unopt (Cray-XC30, 16 locales)
Future Work: A compiler that can uniformly optimize PGAS Programs

- Extend LLVM IR to support parallel programs with PGAS and explicit task parallelism
  - Two parallel intermediate representations (PIR) as extensions to LLVM IR (Runtime-Independent, Runtime-Specific)

Parallel Programs (Chapel, X10, CAF, HC, ...)

1. RI-PIR Gen
2. Analysis
3. Transformation

1. RS-PIR Gen
2. Analysis
3. Transformation

LLVM
- Runtime-Independent Optimizations
  - e.g. Task Parallel Construct

LLVM
- Runtime-Specific Optimizations
  - e.g. GASNet API

Binary