RICE

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)

TM

A Big Picture

CRAY CHAPEL

BERKELEY UPC UNIFIED PARALLEL C

Habanero-C, ...

LLVM

COMPILER INFRASTRUCTURE

©Oak Ridge National Lab.

© Argonne National Lab.

© RIKEN AICS

Pictures borrowed from http://chapel.cray.com/logo.html, http://llvm.org/Logo.html, http://upc.lbl.gov/, http://commons.wikimedia.org/, https://www.olcf.ornl.gov/titan/

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, …);
```

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

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

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;
}
```

Remote data access per each iteration

LICM by LLVM

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

Hoisted out of the loop!

LICM = Loop Invariant Code Motion

An optimization Example : Bulk Transformation (Coalescing)

```
(Pseudo-Code: Before Bulk Transformation)
for i in 1..N {
    // POSSIBLY REMOTE GET
    … = A(i);
}
```

Remote array access per each iteration

Bulk Transformation

```
(Pseudo-Code: After LICM)
var localA: [1..N] int;
localA = A; // Bulk Transfer
for i in 1..N {
    … = localA(i);
}
```

Create Local Buffer & Perform bulk transfer

Converted to Definitely-Local Access!

RICE

An Optimization Example : Locality Inference for avoiding runtime affinity checking

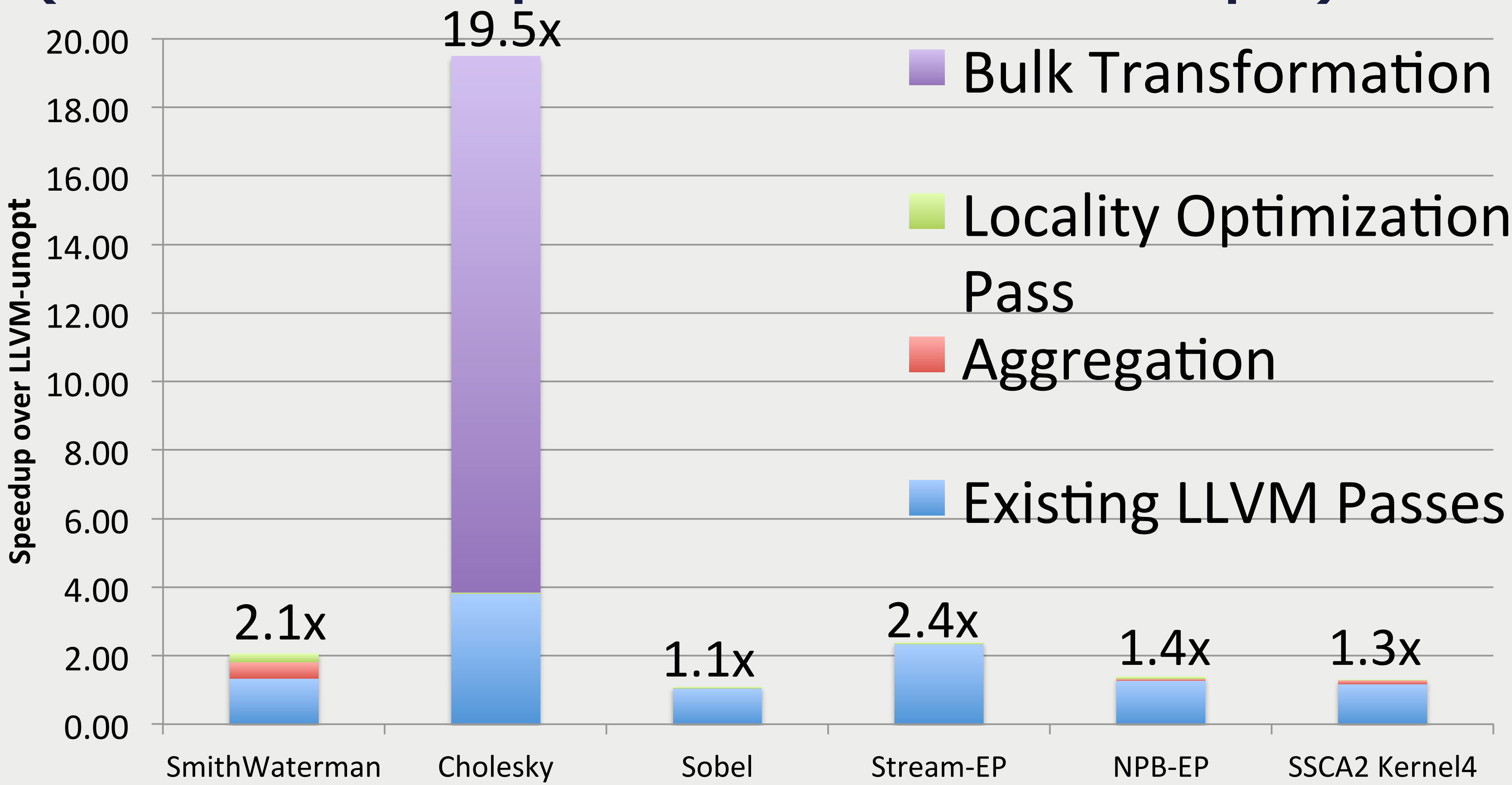
```
proc habanero(ref x, ref y, ref z) {
    var p: int = 0;
    var A:[1..N] int;
    if (x == 0) {
        p = y;
    } else {
        local { p = z; }
    }
    z = A(0) + z;
}
```

A is definitely-local

p and z are definitely local

Definitely-local access!
(avoid runtime affinity checking)

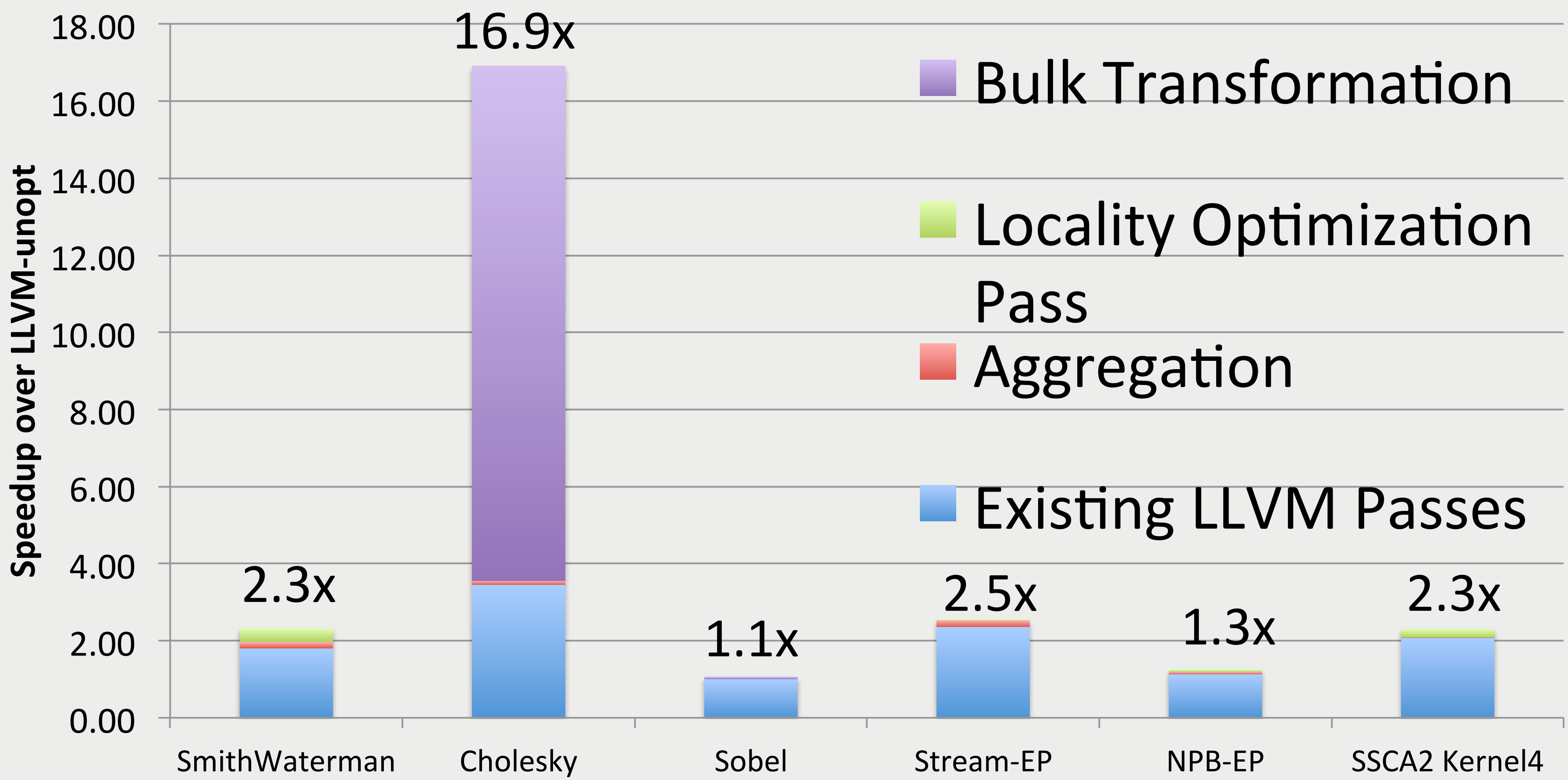
Results on Cray XC-30 (LLVM-unopt vs. LLVM-allopt)



4.6x performance improvement on average
(6 applications, 1, 2, 4, 8, 16, 32, 64 locales)

RICE

Results on Westmere Cluster (LLVM-unopt vs. LLVM-allopt)



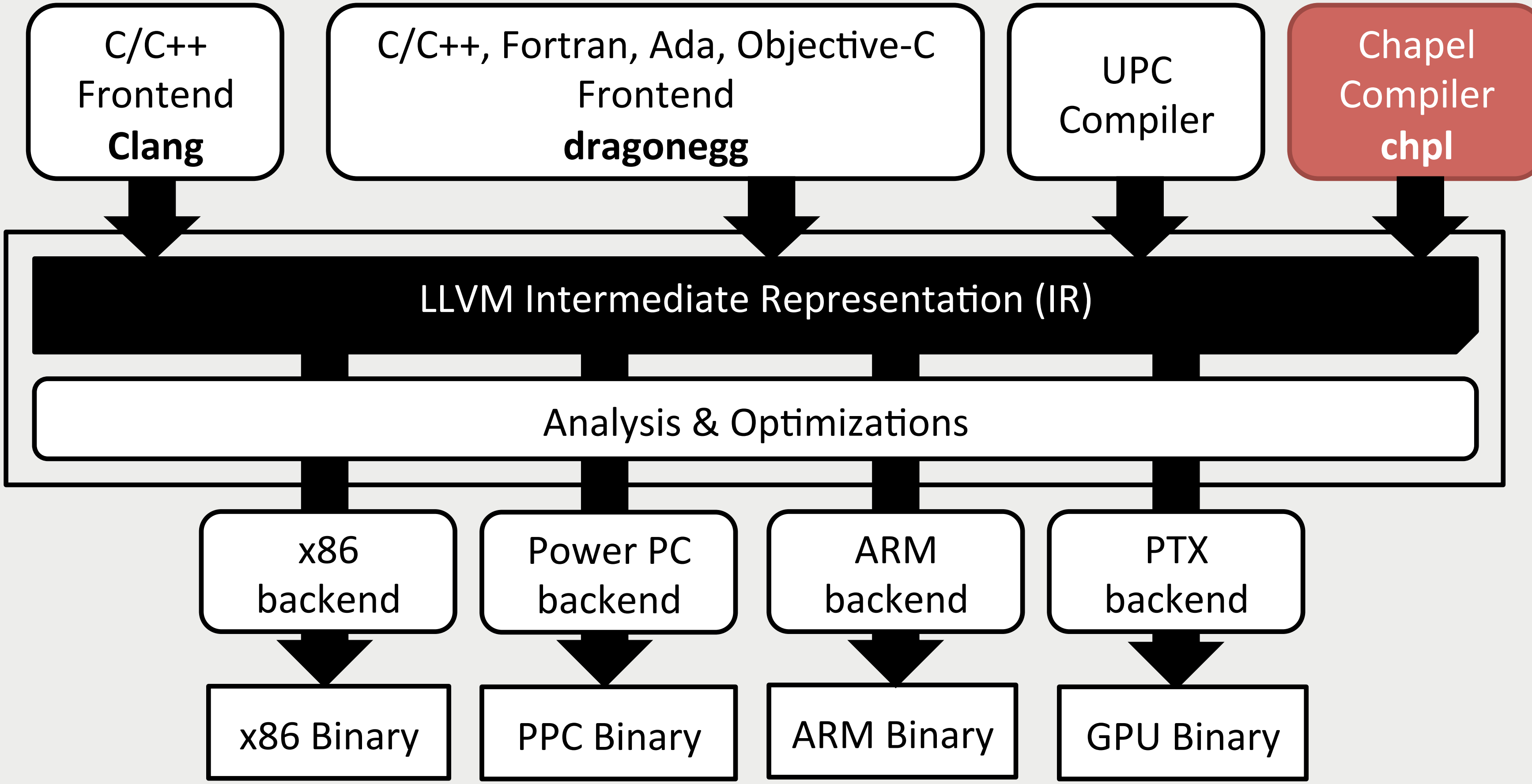
4.4x performance improvement on average
(6 applications, 1, 2, 4, 8, 16, 32 locales)

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



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

RICE

Benchmark	Comm Kind	**Cray XC-30**	
		`LLVM-gopt`	`LLVM-allopt`
Smith-Waterman Note : obtained with 18,560x19,200 input	LOCAL_GET	63.6%	75.5%
	REMOTE_GET	36.4%	36.7%
	LOCAL_PUT	58.0%	58.0%
	REMOTE_PUT	0.0%	0.0%
Cholesky Note : obtained with 2,000x2,000 input	LOCAL_GET	77.6%	87.9%
	REMOTE_GET	84.7%	99.8%
	LOCAL_PUT	10.3%	10.8%
	REMOTE_PUT	0.0%	0.0%
NPB EP	LOCAL_GET	58.6%	58.6%
	REMOTE_GET	39.7%	39.7%
	LOCAL_PUT	29.5%	58.8%
	REMOTE_PUT	0.0%	0.0%
Sobel Note : obtained with CLASS=B	LOCAL_GET	74.6%	95.2%
	REMOTE_GET	0.0%	0.0%
	LOCAL_PUT	35.8%	68.3%
	REMOTE_PUT	0.0%	0.0%
SSCA2	LOCAL_GET	55.6%	56.2%
	REMOTE_GET	60.9%	60.8%
	LOCAL_PUT	5.6%	3.8%
	REMOTE_PUT	0.0%	0.0%
Stream-EP	LOCAL_GET	70.6%	70.6%
	REMOTE_GET	35.7%	35.7%
	LOCAL_PUT	17.3%	17.3%
	REMOTE_PUT	0.0%	0.0%

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 → Binary

LLVM Runtime-Independent Optimizations	LLVM Runtime-Specific Optimizations
e.g. Task Parallel Construct	e.g. GASNet API