Programming Model Challenges for Extreme Scale Computing and Analytics

Vivek Sarkar
E.D. Butcher Chair in Engineering
Professor of Computer Science
Rice University
vsarkar@rice.edu

Acknowledgment: Supported in part by the X-Stack program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR)
Challenges for Exascale & Extreme Scale Systems

- Characteristics of Extreme Scale systems in the next decade
  - Massively multi-core (~ 100’s of cores/chip)
  - Performance driven by parallelism, constrained by energy & data movement
  - Subject to frequent faults and failures

- Many Classes of Extreme Scale Systems

  - Mobile, < 10 Watts, \(O(10^1)\) concurrency
  - Terascale Embedded, 100’s of Watts, \(O(10^3)\) concurrency
  - Petascale Departmental, 100’s of KW, \(O(10^6)\) concurrency
  - Exascale Data Center > 1 MW, \(O(10^9)\) concurrency

Data Challenges in Science

- Overall trend: most science domains will become data-intensive in the exascale timeframe (and many well before then)

What Many Programmers Do Today

Programmers spend significant amount of time rewriting code for improved performance

Source: Ian Karlin
… but rewriting large complex applications for each hardware generation is impractical.

Solutions must be portable, adaptable to the future and maintainable.

Source: Ian Karlin
How should exascale applications be programmed?

“Revolutionary” ideas
- asynchronous parallelism everywhere
- portable computation and data mappings
- distributed global address/name space
- hierarchical abstractions of locality
- built-in support for failure recovery
- freedom from deadlocks, data races, …

→

Evolutionary adoption
- leverage new standards e.g., C++
- Influence and build on future versions of MPI + OpenMP
- offer library interfaces before compiler support, …
Outline

1. DSL example
   - Habanero Data-Flow Graph Language (DFGL)

2. Metadata example
   - Data Layout Specifications in TALC

3. C++11 library example
   - HabaneroUPC++: a compiler-free PGAS library

4. Compiler example
   - Communication Optimizations in LLVM
Motivation: help Application Developers specify all the parallelism in their code in a portable manner

Current practice: use OpenMP task dependences

1. #pragma omp parallel
2. #pragma omp single
3. {
4.   for (int j = iymin; j < iymax; j++){
5.     for (int i = ixmin; i < ixmax; i++){
6.       #pragma omp task depend(in:dataptr[i][j-1]) \ 
7.       depend(in:dataptr[i-1][j]) \ 
8.       depend(out:dataptr[i][j])
9.       process_cell(i,j,nu,ncellx,ncelly,vo,vi, ...);
10.   } // for-i
11. } // for-j
12.} // omp-parallel

... but this fine-grained task parallel version won’t run efficiently on any platform!
Our Approach: use Data-Flow Graph Language (DFGL) as an eDSL that is amenable to compiler optimizations

```
// C/C++ code declaring vo, process_cell, etc

#pragma dfgl
{
    // Dependsences
    [vo:j-1,i],[vo:j,i-1] -> (process_cell:j,i) -> [vo:j,i];

    // Iteration domain
    env :: (process_cell:{iymin..iymax-1},{ixmin..ixmax-1});
}
```

Access functions

Computation step instance

Iteration domains specified as ranges

The environment starts the initial steps in the graph

"Polyhedral Optimizations for a Data-Flow Graph Language." Alina Sbirlea, Jun Shirako, Vivek Sarkar, LCPC 2015
Prototype Implementation of DFGL in D-TEC project

- Automatic DFGL $\rightarrow$ SCoP $\rightarrow$ OpenMP transformations
  - OpenMP is a useful as a portable target for higher level programming models
- SCoP transformations generate tiled OpenMP parallel code with OpenMP 4.1 doacross construct for pipeline parallelism
- Experimental results obtained for LULESH implemented with DFGL
  - Single POWER7 node: 32 cores, 3.86GHz (BlueBiou system @ Rice)
  - LULESH problem size: 50 iterations, 100x100x100 space
LULESH speedup on POWER7

![LULESH speedup on POWER7](image)
Outline

1. DSL example
   - Habanero Data-Flow Graph Language (DFGL)

2. Metadata example
   - Data Layout Specifications in TALC

3. C++11 library example
   - HabaneroUPC++: a compiler-free PGAS library

4. Compiler example
   - Communication Optimizations in LLVM
Motivation: help Application Developers specify data layouts in a portable manner

<table>
<thead>
<tr>
<th>Platform</th>
<th>Layout 1</th>
<th>Layout 2</th>
<th>Layout 3</th>
<th>Layout 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM POWER7</td>
<td>1.00</td>
<td>4.66</td>
<td>4.66</td>
<td>4.71</td>
</tr>
<tr>
<td>AMD APU</td>
<td>1.00</td>
<td>1.26</td>
<td>1.38</td>
<td>1.40</td>
</tr>
<tr>
<td>Intel Sandybridge</td>
<td>1.00</td>
<td>1.06</td>
<td>1.10</td>
<td>1.10</td>
</tr>
<tr>
<td>IBM BG/Q</td>
<td>1.00</td>
<td>1.65</td>
<td>2.14</td>
<td>2.20</td>
</tr>
</tbody>
</table>

Example: performance of IRSmk kernel relative to default 27x1 layout for different CPUs – opposite trends are seen on GPUs!
Our Approach

Separation of Concerns:

- User specifies data layouts for different architectures in "meta files"
- Automatic Layout Generator can be used to recommend best layouts for different architectures
- Code is unchanged!

## Manual vs. Automated Layout Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Power7 8 Threads</th>
<th>AMD APU 4 Threads</th>
<th>Sandybridge 8 Threads</th>
<th>BG/Q 64 Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRSmk Best Manual Layout</td>
<td>4.70x</td>
<td>1.46x</td>
<td>1.11x</td>
<td>2.20x</td>
</tr>
<tr>
<td>IRSmk Automated Layout</td>
<td>4.67x</td>
<td>1.43x</td>
<td>1.10x</td>
<td>2.08x</td>
</tr>
<tr>
<td>LULESH Best Manual Layout</td>
<td>1.43x</td>
<td>1.50x</td>
<td>1.02x</td>
<td>1.10x</td>
</tr>
<tr>
<td>LULESH Automated Layout</td>
<td>1.58x</td>
<td>1.46x</td>
<td>0.96x</td>
<td>1.07x</td>
</tr>
<tr>
<td>SRAD Best Manual Layout</td>
<td>1.35x</td>
<td>3.13x</td>
<td>1.00x</td>
<td>1.08x</td>
</tr>
<tr>
<td>SRAD Automated Layout</td>
<td>1.20x</td>
<td>2.55x</td>
<td>0.46x</td>
<td>0.98x</td>
</tr>
</tbody>
</table>

*Speedup of Best Manual and Automated Layout relative to base layout*
Outline

1. DSL example
   - Habanero Data-Flow Graph Language (DFGL)

2. Metadata example
   - Data Layout Specifications in TALC

3. C++11 library example
   - HabaneroUPC++: a compiler-free PGAS library

4. Compiler example
   - Communication Optimizations in LLVM
Motivation: help Application Developers leverage productivity benefits of new C++ standards

**C++ futures**

```cpp
// create async task w/ result
auto f = std::async(
    <lambda-expr>);

. . .

// Retrieve result
// (wait if needed)
int result = f.get();
```

**C+11 lambda expressions**

```cpp
// create lambda
auto func =
    [ capture_list ]
    (formal_params) { ... };

. . .

// execute lambda
func (argument_list);
```
Our Approach: HabaneroUPC++ Library
(Example constructs)

Remote task creation
asyncAt ( destPlace, [capture_list] ( ) {
  Statements1;
});

Asynchronous one-sided data movement
asyncCopy (src, dest, count, ddf);

Message-driven task activation
asyncAwait(ddf, capture_list] ( ) {
  Statements2;
});
Weak Scaling Result for Habanero-UPC++ version of LULESH on NERSC Edison system

![Graph showing performance (FOM z/sec) vs HabaneroUPC++ Places for different worker/place configurations: 1 worker/place, 4 worker/place, 8 worker/place, 12 worker/place. The graph includes logarithmic axes for both the x and y axes, with performance ranging from 100 to 1e+06 and places ranging from 1 to 512.](image)
LSMS example (MPI and Habanero-UPC++ versions)

**MPI version:**
// Post MPI_IRecv() calls
... 
// Post MPI_ISend() calls
... 
// Perform all MPI_Wait() calls
// Perform tasks
... 

**Habanero-UPC++ version:**
// Issue one-sided
// asyncCopy() calls
... 
// Issue data-driven tasks
// in any order
hcpp::asyncAwait(
  result1, result2,
  [=]() { task body });
... 

Source: Markus Eisenbach, Wael Elwasif
Habanero-UPC++ is enabled by tight integration of task and communication runtimes

This integration has been demonstrated for MPI and GASNet --- motivation for Open Community Runtime (OCR) project.
Outline

1. DSL example
   - Habanero Data-Flow Graph Language (DFGL)

2. Metadata example
   - Data Layout Specifications in TALC

3. C++11 library example
   - HabaneroUPC++: a compiler-free PGAS library

4. Compiler example
   - Communication Optimizations in LLVM
Our vision: Common LLVM-based Communication Optimization Framework for Multiple Languages

Chapel Programs
UPC++ Programs
X10 Programs
CAF Programs

Chapel-LLVM frontend
UPC++-LLVM frontend
X10-LLVM frontend
CAF-LLVM frontend

LLVM IR (use address space feature)

LLVM-based Communication Optimization passes
Lowering Pass

“LLVM-based Communication Optimizations for PGAS Programs,” Akihiro Hayashi, Jisheng Zhao, Michael Ferguson, Vivek Sarkar.
Performance improvement due to Communication Optimization for Jacobi example in Habanero-UPC++
Genomics Analysis: a Computational Science Example that uses Spark on JVMs (NSF InTrans project w/ UCLA)

A large genome collection of healthy population and cancer patients

Big data, compute-intensive

Enable effective searches of precision medicine for cancer treatment

Customized accelerators

Genomic analysis pipeline

Supercomputer in a rack

Gene mutations discovered in codon reading frames
Portable Mapping of Java 8 Streams on to CPUs/GPUs

Platform: Single node with two 10-core IBM POWER8 sockets (20 cores, 160 threads, 3.69GHz, 256GB RAM/node), and one NVIDIA K40m GPU (2880 CUDA cores, 876MHz, 12GB global memory PCI-Express Gen 3).

Reference: Compiling and Optimizing Java 8 Programs for GPU Execution”, Kazuaki Ishizaki, Akihiro Hayashi, Gita Koblents, Vivek Sarkar, PACT 2015.
Hadoop
Reliability, Distribution

APARAPI
Bytecode translation, buffer management

OpenCL
Multi-architecture execution in native threads

HadoopCL
- Hardware aware platform manager (CPU, GPU, JVM execution modes) and multi-device scheduler
Mahout = original version, Manual = GPU-accelerated version. Each node contains a 12-core 2.80GHz Intel X5660 CPU, 2 discrete NVIDIA M2050 GPUs each with 2.5GB of global memory, and 48GB of system RAM. All nodes are connected by Infiniband, with a peak bandwidth of 40Gb/s.
Directive from Session Chairs

“Please feel free to make bold predictions and call out underserved areas of R&D in the overall space.”
Conclusions

- Predictions
  - MPI+OpenMP will be used as infrastructure to enable higher-level programming models

- Some under-served R&D areas
  - Exploration of data-centric programming models, compilers, and runtimes for HPC and Data Analytics
  - Compiler optimizations for parallel codes
  - Integrated runtime for parallelism, heterogeneity, communication, and I/O
BACKUP SLIDES START HERE
First Innovation Transitions (InTrans) Award from Intel + NSF

Press Release 14-086
TAKEING GREAT IDEAS FROM THE LAB TO THE FAB

NSF and Intel support the development of domain-specific hardware to address health care needs

Real-time adaptive low-dose CT-scan enabled by customized computing.

Credit and Larger Version

July 17, 2014

A "valley of death" is well-known to entrepreneurs--the lull between government funding for research and industry support for prototypes and products. To confront this problem, in 2013 the National Science Foundation (NSF) created a new program called InTrans to extend the life of the most high-impact NSF-funded research and help great ideas transition from lab to practice.

Today, in partnership with Intel Corporation, NSF announced the first InTrans award of $3 million to a team of researchers who are designing customizable, domain-specific computing technologies for use in healthcare.

- We invite two kinds of submissions:
  - Full-length research papers (8-page limit)
  - Short papers (4-page limit), which can take the form of position papers, experience reports, or surveys/comparisons of runtime systems

- No formal proceedings → papers presented at the workshop will also qualify for publication in more formal venues in the future

- Important Dates
  - Paper submission: **September 11, 2015**
  - Notification: October 2, 2015
  - Final papers for distribution at workshop: October 30, 2015
  - Workshop Date: Monday, November 16, 2015