Towards an Exascale SDK

Costin Iancu
Lawrence Berkeley National Laboratory
Berkeley Lab Initiative in Extreme Data Science

XDSF: Extreme Data Science Facility concept

Other data-producing sources

Extreme Data Science Facility (XDSF)

MS-DESI

ALS

LCLS

APS

JGI

LHC

ESnet

Energy Sciences Network
XDSF: Will bring scientists together with data researchers and software engineers
Building the Exascale Ecosystem

- Multidisciplinary science at Exascale requires novel functionality in the software development environment

- Large system scale (DEGAS)
  Programming environment that adapts to system variability in performance and availability

- Composed execution models (Corvette)
  Automated techniques to reason about program behavior (correctness, performance, precision)

- “Data” pipelines (Hobbes)
  New system level support for application composition (resources, data)
DEGAS:
Dynamic Exascale Global Address Space

Katherine Yelick, LBNL PI
Vivek Sarkar & John Mellor-Crummey, Rice
James Demmel, Krste Asanović & Armando Fox, UC Berkeley
Mattan Erez, UT Austin
Dan Quinlan, LLNL
Surendra Byna, Paul Hargrove, Steven Hofmeyr, Costin Iancu, Khaled Ibrahim, Leonid Oliker, Eric Roman, John Shalf, David Skinner, Erich Strohmaier, Samuel Williams, Yili Zheng, LBNL
**Mission Statement:** To ensure the broad success of Exascale systems through a unified programming model that is productive, scalable, portable, and interoperable, and meets the unique Exascale demands of energy efficiency and resilience.
DEGAS Proposal: Goals and Objectives

- **Scalability:**
  - Billion-way concurrency; performance through hierarchical locality control

- **Programmability:**
  - Convenient programming through a global address space and high-level abstractions and libraries

- **Performance Portability:**
  - Ensure applications can be moved across diverse machines with domain-specific optimizations

- **Resilience:**
  - Integrated support for capturing state and recovering from faults

- **Energy Efficiency:**
  - Avoid communication, which will dominate energy costs, and adapt to performance heterogeneity due to system-level energy management

- **Interoperability:**
  - Encourage use of languages and features through incremental adoption
DEGAS: Dynamic Exascale Global Address Space

Hierarchical Programming Models
Communication-Avoiding Libraries and Compilers
Adaptive Interoperable Runtimes
Lightweight One-Sided Communication

UPC, Co-Array Fortran (CAF), Habanero-C, and libraries!
What we love about UPC

• Convenience
  – Build large shared structures (PGAS)
  – Read and write data “anywhere, anytime” (global, asynchronous, and one-sided); would like more than read/writes

• Locality and scalability (shared with MPI)
  – Explicit control over data layout

• That it’s a language rather than library
  – Syntactic elegance: *p = ... vs shem_put(p,...)
  – Optimizations from compilers
    • Communication, pointers, etc.
  – Correctness from compilers
    • Race and deadlock analysis,...
    • More in Titanium, less in UPC
Hierarchical PGAS (HPGAS) hierarchical memory & control

Beyond (Single Program Multiple Data, SPMD)

- Hierarchical locality model for network and node hierarchies
- Hierarchical control model for applications (e.g., multiphysics)

- **Option 1: Dynamic parallelism creation**
  - Recursively divide until... you run out of work (or hardware)

- **Option 2: Hierarchical SPMD with “Mix-ins”**
  - Hardware threads can be grouped into units hierarchically
  - Add dynamic parallelism with voluntary tasking on a group
  - Add data parallelism with collectives on a group

Two approaches: collecting vs spreading threads
UPC++ Programming System in DEGAS

• **Problem**
  – Need H-PGAS support for C++ applications
  – C++ compiler is very complex

• **Solution: “Compiler-Free” UPC++**
  – Template library approach reduces development and maintenance costs by leveraging C++ standards and compilers
  – Use SPMD+Aysnc execution model
  – Combine popular features from existing PGAS languages: async in Phalanx/X10, M-D domains and arrays in Titanium/Chapel
  – Interoperate with MPI, OpenMP, CUDA and etc.

• **Recent Progress**
  – Design of H-PGAS in the context of UPC++
  – UPC++ prototype development
  – IPDPS14 paper with results on Cray XC30 and IBM BG/Q

• **Impact**
  – Provided programming productivity similar to UPC and Titanium for C/C++ apps
  – Demonstrated competitive performance

UPC++ software: [https://bitbucket.org/upcxx/upcxx/](https://bitbucket.org/upcxx/upcxx/)
Extending Remote Access

• **We can make the global address space more powerful**
  - Remote read and write
  - Remote atomic invocation
  - Active messages (small functions that execute at high priority)
  - Remote function invocation
  - Remote invocation with multiple dependences (DAG)
  - Run anywhere in region (e.g., on-node task queue)
  - Run anywhere (global task queue)

• **Retain the SPMD model for locality: 1 main thread per core**

• **Key questions:**
  - How quickly do things run vs runtime aggregates communication
  - Resource management: avoid timing-dependence deadlock
# Initial (highly subjective) Analysis of Base Languages

<table>
<thead>
<tr>
<th></th>
<th>C++</th>
<th>Python</th>
<th>Scala</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base performance</strong></td>
<td>Good</td>
<td>Poor</td>
<td>Medium (likely to improve)</td>
</tr>
<tr>
<td><strong>Multicore performance</strong></td>
<td>Good</td>
<td>Very poor</td>
<td>OK</td>
</tr>
<tr>
<td><strong>Cross-language support</strong></td>
<td>Good</td>
<td>Good</td>
<td>Poor (expensive)</td>
</tr>
<tr>
<td><strong>Existing libraries</strong></td>
<td>Good</td>
<td>Good</td>
<td>Poor</td>
</tr>
<tr>
<td><strong>Extensibility</strong></td>
<td>Medium (cannot overload .)</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Popularity / ecosystem</strong></td>
<td>Medium</td>
<td>Good</td>
<td>Poor</td>
</tr>
<tr>
<td><strong>Language safety</strong></td>
<td>Poor</td>
<td>Poor</td>
<td>Good</td>
</tr>
</tbody>
</table>

Current focus of DEGAS for DOE: DEGAS Proposed (cut)
Communication-avoiding algorithms generalized to compilers, and communication optimizations in PGAS
Generalizing Communication Lower Bounds and Optimal Algorithms

- For serial matmul, we know \( \#\text{words\_moved} = \Omega \left( \frac{n^3}{M^{1/2}} \right) \), attained by tile sizes \( M^{1/2} \times M^{1/2} \)
  - Where do all the \( \frac{1}{2} \)'s come from?

- Thm (Christ, Demmel, Knight, Scanlon, Yelick): For any program that “smells like” nested loops, accessing arrays with subscripts that are linear functions of the loop indices, \( \#\text{words\_moved} = \Omega \left( \frac{\#\text{iterations}}{M^e} \right) \), for some \( e \) we can determine

- Thm (C/D/K/S/Y): Under some assumptions, we can determine the optimal tiles sizes

- Long term goal: All compilers should generate communication optimal code from nested loops
Communication Avoidance in DEGAS

**Problem**
- Communication dominates time and energy
- This will be worse in the Exascale era

**Solution**
- Optimize latency by overlapping with computation and other communication
- Use faster one-sided communication
- Use new Communication-Avoiding Algorithms (provably optimal communication)
- Automatic compiler optimizations

**IMPACT**
- Dense linear algebra study shows 2X speedups from both overlap and avoidance
- New “HBL” theory generalizes optimality to arbitrary loops with array expressions
- First step in automating communication-optimal compiler transformations

DEGAS: Dynamic Exascale Global Address Space

Hierarchical Programming Models
Communication-Avoiding Compilers
Adaptive Interoperable Runtimes
Lightweight One-Sided Communication

Energy / Performance Feedback
Resilience

LITHE, JUGGLE: adaptive and efficient runtime
Management of critical resources is increasingly important:

- **Memory and network bandwidth limited** by cost and energy
- **Capacity limited at many levels**: network buffers at interfaces, internal network congestion are real and growing problems

**DEGAS Overview**

**InfiniBand - 8 byte Msg Throughput**

- **Processes (8UPC)**
  - 3X
- **Hybrid**
  - 2X
- **Pthreads**
  - 5X

*Having more than 4 submitting processes can negatively impact performance by up to 4x*
DEGAS: Dynamic Exascale Global Address Space

Next Generation GASNet
GASNet-EX plans:
• **Congestion management:** for 1-sided communication with ARTS
• **Hierarchical:** communication management for H-PGAS
• **Resilience:** globally consistent states and fine-grained fault recovery
• **Progress:** new models for scalability and interoperability

Leverage GASNet (redesigned)
• Major changes for on-chip interconnects
• Each network has unique opportunities

• **Interface under design:** “Speak now or....”
  – [https://sites.google.com/a/lbl.gov/gasnet-ex-collaboration/](https://sites.google.com/a/lbl.gov/gasnet-ex-collaboration/)
DEGAS: Dynamic Exascale Global Address Space

Hierarchical Programming Models
Communication-Avoiding Compilers
Adaptive Interoperable Runtimes
Lightweight One-Sided Communication

Containment Domains and BLCR (Berkeley Lab Checkpoint Restart)
Resilience Approaches

• **Containment Domains (CDs) for trees**
  – Flexible resilience techniques (mechanism not policy)
  – Each CD provides own recovery mechanism
  – Analytical model: 90%+ efficiency at 2 EF vs. 0% for conventional checkpointing

• **Berkeley Lab Checkpoint Restart**
  – BLCR is a system-level Checkpoint/Restart
    • Job state written to filesystem or memory; works on most HPC apps
  – Checkpoint/Restart can be used for roll-back recovery
    • a course-grained approach to resilience
    • BLCR also enables use for job migration among compute nodes
  – Requires support from the MPI implementation
  – Impact: part of standard Linux release

(DEGAS Overview)
PGAS Applications (Ongoing)

• Meraculous (Evangelos Georganas)
• Convergent Matrix (Scott French)
  https://github.com/swfrench/convergent-matrix
• Communication-Avoiding Matrix (Penporn Koanantakool)
• Embree graphics (Michael Driscoll)
  https://github.com/mbdriscoll/embree/tree/upcxx
• NPB CG, MG and FT (Amir Kamil)
• Fan-both Sparse Cholesky (Mathias Jacquelin)
• mini-GMG (Hongzhang Shan)
• MILC (Hongzhang Shan)
• Global Arrays and NWChem (Eric Hoffman, Hongzhang Shan)
Planned:
• Stochastic Gradient
Corectness, Verification and Testing of Exascale Applications (Corvette)

Koushik Sen (PI)
James Demmel
UC Berkeley

Costin Iancu
Lawrence Berkeley National Laboratory
Goals

Develop correctness tools for different programming models: PGAS, MPI, dynamic parallelism

I. Testing and Verification

- Identify sources of non-determinism in executions
- Data races, atomicity violations, non-reproducible floating point results
- Explore state-of-the-art techniques that use dynamic analysis
- Develop precise and scalable tools: < 2X overhead

II. Debugging

- Use minimal amount of concurrency to reproduce bug
- Support two-level debugging of high-level abstractions
- Detect causes of floating-point anomalies and determine the minimum precision needed to fix them
PRECIMONIOUS: Tuning Assistant for Floating-Point Precision

Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen

University of California, Berkeley

David H. Bailey, Costin Iancu

Lawrence Berkeley National Laboratory

David Hough

Oracle

SC’13
Motivation

- Floating-point arithmetic used in wide variety of domains
- Reasoning about these programs is difficult
  - Large variety of numerical problems
  - Most programmers not experts in floating point or numerical analysis
- Common practice
  - Use the highest available floating-point precision
  - Disadvantages: more expensive in terms of running time, storage, and energy consumption
- Goal: develop automated technique to assist in tuning floating-point precision
Example (D.H. Bailey)

- Consider the problem of finding the arc length of the function

\[ g(x) = x + \sum_{0 \leq k \leq 5} 2^{-k} \sin(2^k x) \]

- Summing for \( x_k \in (0, \pi) \) into \( n \) subintervals

\[ \sqrt{h^2 + (g(x_{k+1}) - g(x_k))^2} \]

with \( h = \pi / n \) and \( x_k = kh \)

<table>
<thead>
<tr>
<th>Precision</th>
<th>Slowdown</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>double-double</td>
<td>20X</td>
<td>5.795776322412856 ✔</td>
</tr>
<tr>
<td>double</td>
<td>1X</td>
<td>5.795776322413031 ✗</td>
</tr>
<tr>
<td>Summation variable is double-double</td>
<td>&lt; 2X</td>
<td>5.795776322412856 ✔</td>
</tr>
</tbody>
</table>
Example (D.H. Bailey)

Original Program

```c
long double fun(long double x) {
    int k, n = 5;
    long double t1 = x;
    long double d1 = 1.0L;
    for(k = 1; k <= n; k++) {
        ...
    }
    return t1;
}

int main() {
    int i, n = 1000000;
    long double h, t1, t2, dppi;
    long double s1;
    ...
    for(i = 1; i <= n; i++) {
        t2 = fun(i * h);
        s1 = s1 + sqrt(h*h + (t2 - t1)*(t2 - t1));
        t1 = t2;
    }
    // final answer stored in variable s1
    return 0;
}
```

Tuned Program

```c
double fun(double x) {
    int k, n = 5;
    double t1 = x;
    float d1 = 1.0f;
    for(k = 1; k <= n; k++) {
        ...
    }
    return t1;
}

int main() {
    int i, n = 1000000;
    double h, t1, t2, dppi;
    long double s1;
    ...
    for(i = 1; i <= n; i++) {
        t2 = fun(i * h);
        s1 = s1 + sqrtf(h*h + (t2 - t1)*(t2 - t1));
        t1 = t2;
    }
    // final answer stored in variable s1
    return 0;
}
Searching efficiently over variable types and function implementations

- Naïve approach → exponential time
  - 19,683 configurations for arc length program \(3^9\)
  - 11 hours 5 minutes
- Global minimum vs. a local minimum

Evaluating type configurations

- Less precision does not always result in performance improvement
- Run time, memory usage, energy consumption, etc.

Determining accuracy constraints

- How accurate must the final result be?
- What error threshold to use?
Speedup for Various Error Thresholds

- Speedup %
  - 10^-4
  - 10^-6
  - 10^-8
  - 10^-10

- Functions: arclength, simpsons, bessel, gaussian, roots, polyroots, roothwt, sum, fft, blas, ep, cg

- Graph shows speedup percentage for various error thresholds.
Hobbes: OS and Runtime Support for Application Composition

Ron Brightwell (PI)
Sandia National Laboratory

Costin Iancu (Programming Models)
LBNL
- **Internode composition**
  - Minimally intrusive
  - Need to connect outputs to inputs
  - Potential for inefficient resource usage

- **Intranode composition**
  - Separate enclaves
    - Minimally intrusive
    - Connections could be made using shared memory
  - Unified Runtime

- **Integration**
  - I/O operation invokes visualization
  - Minimal overhead

- **Can we use a single description to support all of these implementations?**
  - Mapping logical structures onto physical resources through virtualization
Additional mechanisms needed to manage multiple VMs. Run in kernel mode to take advantage of VM support in modern processors. AKA Palacios

Basic mechanisms needed to virtualize hardware resources like address spaces. AKA Kitten

VMs can share the resources via time sharing or space sharing. This is managed by the GOS
Yardstick for success – application performance and ease of development
- Performance metrics – time, energy
- Software engineering metrics – “composability”, “tunability”

Interact with Co-Design centers for experiments

Identify separation of concerns between adaptive runtime and OS support
- Distinguish between mechanisms and policies
- Resource management: core, memory, network, energy
- Enable user level implementations and policies
- Identify the protection and isolation requirements

MANTRA: Check first if it can be done at the user level

Approach:
- Top-down – examine runtime APIs, determine lacking OS support
- Bottom-up – propose novel OS APIs, examine runtime implementation
THANK YOU!