Highly Scalable Large-Scale Asynchronous Graph Processing using Actors

Youssef Elmougy*, Akihiro Hayashi, and Vivek Sarkar
Georgia Institute of Technology, Atlanta GA USA

* Corresponding Author and Presenting Author
The Growth of Big Data

Graph algorithms have become increasingly important for solving problems in many computational domains!

The scale of these graphs present difficulties to their processing and analysis!
The Scalability Problem

- There is a need for parallel computing resources to meet the computational and memory requirements.
- Existing algorithms and software that perform well for mainstream parallel scientific applications are not necessarily efficient for large-scale graph applications.

It is critical to develop lightweight and scalable systems to efficiently process large-scale graphs!
The 4 Overarching Challenges in Parallel Graph Processing

1. Size and scale of the problem
   - Increasing graph size and complexity
2. Synchronization overhead
   - Synchronization overhead
3. Poor locality of memory access
4. Slow convergence

This paper studies the scalability of our novel actor-based programming system to overcome the inherent challenges of large-scale graph processing!
Actor-based Scalable Architecture for Graph Processing

- Presents a lightweight, asynchronous computation model
- Utilizes fine-grained asynchronous actor messages to express point-to-point remote operations
- Treats actors as primitives of computation, where actors are inherently isolated and share no mutable state
- Actors process messages sequentially within its mailbox, thereby avoiding data races and synchronization

NOTE: “Actor” and “Selector” will be used interchangeably
The 4 overarching challenges of parallel graph processing:

1. Size and scale of the problem
   - Increasing graph size and complexity

2. Synchronization overhead
   - Synchronization overhead

3. Poor locality of memory access
   - Slow convergence

4. Slow convergence
   - Slow convergence

SOLUTION

- Large-scale graph can be distributed across multiple PEs, where the local partition per PE is small enough to fit in its local memory.

Distributed Actor Runtime

Sample Graph
Actor-based Scalable Architecture for Graph Processing

The 4 overarching challenges of parallel graph processing:

1. Size and scale of the problem
   - Increasing graph size and complexity
   - Poor locality of memory access
   - Synchronization overhead
   - Slow convergence

SOLUTION

Execution Model

- Fine-grained Asynchronous Bulk-Synchronous (FABS) Parallelism model
- Reduces barriers and time spent idling at barriers, further reducing stall cycles
Actor-based Scalable Architecture for Graph Processing

The 4 overarching challenges of parallel graph processing:

1. Size and scale of the problem
2. Synchronization overhead
3. Poor locality of memory access
4. Slow convergence

- Well suited for irregular applications due to the Partitioned Global Address Space (PGAS)
- Expresses point-to-point remote operations as short, one-sided fine-grained async messages
- Message aggregation allows for low overhead and high network utilization
- Computation is migrated to where the data is located (moving compute to data)
Actor-based Scalable Architecture for Graph Processing

The 4 overarching challenges of parallel graph processing:

1. Size and scale of the problem
2. Synchronization overhead
3. Poor locality of memory access
4. Slow convergence

- As actor messages are executed, the local state of the actor is updated, allowing the next received actor message to utilize the updated state within the same super-step.
- Updates the neighboring vertices with most recent values within the same iteration.
We focus on PageRank and Jaccard Index due to two reasons:

1. They show iterative vs non-iterative application types
2. They have been applied to many real-world problems with social impact
Experimental Setup and Architecture

• **Metrics**: execution time (in seconds) and traversed edges per second (TEPS)
• Experiments conducted on the CPU nodes of the **Perlmutter supercomputer** at the National Energy Research Scientific Computing Center (NERSC)
  • 2x AMD EPYC 7763 (Milan) CPUs
  • 64 physical cores per CPU
  • 512 GB memory
  • 1x HPE Cray Slingshot Interconnect
• Results for **different dimensions of scalability** are presented

Picture borrowed from: [https://docs.nersc.gov/systems/perlmutter/architecture/](https://docs.nersc.gov/systems/perlmutter/architecture/)
Dimensions of Scalability

Scaling performance is shown using three experiment types:

(1) SCALE1
- Increasing graph size (WEAK SCALING, 5000 vertices per core)
- Increasing #nodes (SCALE-OUT, from 256 to 65,536 cores, from 1 to 1024 nodes)

(2) SCALE2
- Constant graph size (STRONG SCALING, 10.2M vertices & 696.4M edges)
- Increasing #nodes (SCALE-OUT, from 256 to 65,536 cores, from 1 to 1024 nodes)

(3) SCALE3
- Constant graph size (STRONG SCALING, 10.2M vertices & 696.4M edges)
- Increasing #cores per node (SCALE-UP, from 1 core/node to 128 cores/node)
Performance Results: SCALE1

**PageRank**

13B TEPS

**Jaccard Index**

85.7% parallel efficiency

10B TEPS
Performance Results: **SCALE2**

- **PageRank**
  - 70.6% parallel efficiency

- **Jaccard Index**
  - #CORES: 16x, SPEEDUP: 13x
  - #CORES: 128x, SPEEDUP: 90.4x
Performance Results: SCALE3

PageRank

20.1x ↑

Jaccard Index

81.5x ↑
Contrasting to Related Approaches

- We contrast with respect to **remote atomics performance** and **graph application performance**
- Related approaches: OpenSHMEM, UPC, MPI3-RMA, YGM

<table>
<thead>
<tr>
<th>Communication System</th>
<th>Non-Blocking</th>
<th>Read / Get</th>
<th>Update / Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSHMEM (cray-shmem 7.7.19)</td>
<td>N</td>
<td>40.06</td>
<td>N.A.</td>
</tr>
<tr>
<td>OpenSHMEM NBI (cray-shmem 7.7.19)</td>
<td>Y</td>
<td>4.79</td>
<td>4.99</td>
</tr>
<tr>
<td>UPC (Berkley-UPC 2022.5.0)</td>
<td>N</td>
<td>30.37</td>
<td>30.03</td>
</tr>
<tr>
<td>UPC NBI (Berkley-UPC 2022.5.0)</td>
<td>Y</td>
<td>20.58</td>
<td>N.A.</td>
</tr>
<tr>
<td>MPI3-RMA (OpenMPI 4.1.2)</td>
<td></td>
<td>25.44</td>
<td>142.04</td>
</tr>
<tr>
<td>MPI3-RMA (cray-mpich 7.7.19)</td>
<td>Y</td>
<td>9.67</td>
<td>59.47</td>
</tr>
<tr>
<td>YGM</td>
<td>Y</td>
<td>N.A.</td>
<td>&gt; 600</td>
</tr>
<tr>
<td>Actors (cray-shmem 7.7.19)</td>
<td>Y</td>
<td>2.70</td>
<td>0.66</td>
</tr>
</tbody>
</table>

Remote Atomics

The performance data indicates a 1.8x and 7.6x improvement in read and set operations, respectively, compared to baseline.
Contrasting to Related Approaches

PageRank

Weak Scaling

Strong Scaling

24.5x

21.6x

13B TEPS

29.6x

2.8B TEPS

16.0x

0.5B TEPS
Impact of the Solution

- The actor-based system has shown **scalability**, **productivity**, and **performance**
- The extensibility of this system can front four impacts:

1. The system can be used on graphs of higher scale as well as systems of higher scaled-up/-out hardware resources
2. The system can be extended to other large-scale (iterative/non-iterative) graph applications
3. The system can be applied to structured, regular applications
4. The system can be expanded and compared to other related PGAS and non-PGAS approaches
You can try this at home... Just visit hclib-actor.com!
ACKNOWLEDGEMENT

This research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), through the Advanced Graphical Intelligence Logical Computing Environment (AGILE) research program, under Army Research Office (ARO) contract number W911NF22C0083. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government.
Highly Scalable Large-Scale Asynchronous Graph Processing using Actors

Youssef Elmougy, Akihiro Hayashi, and Vivek Sarkar
Georgia Institute of Technology, Atlanta GA USA

Thank you for your attention!