Productive Data Locality Optimizations in Distributed Memory

Engin Kayraklioglu
engin@gwu.edu
Advisor: Tarek El-Ghazawi

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Abstract

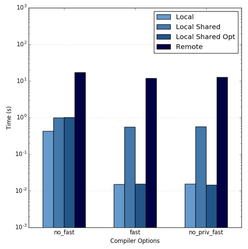
With deepening memory hierarchies in HPC systems, the challenge of managing data locality gains more importance. Coincidentally, increasing ubiquity of HPC systems and wider range of disciplines utilizing HPC introduce more programmers to the HPC community. Given these two trends, it is imperative to have scalable and productive ways to manage data locality.

In this research, we address the problem in multiple ways. We propose a novel language feature that programmers can use to transform shared memory applications to distributed memory applications easily. We introduce a high-level profiling tool to help understand how distributed arrays are used in an application. As next steps, we are designing a model to describe the implementation of data locality optimizations as an engineering process, which can lend itself to combinatorial optimization. We are also implementing a profile-based automatic optimization framework that utilizes AI to replace the programmer completely in implementing optimizations for distributed memory.

Background

Problem Characterization

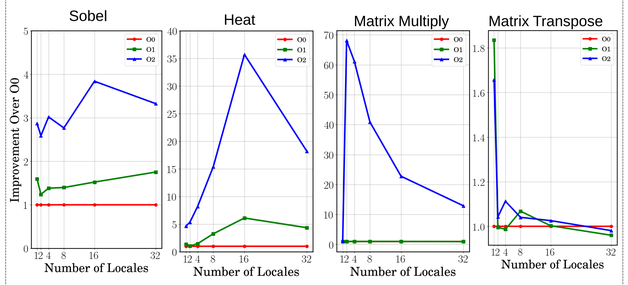
- Quantifying the latency of different access types in PGAS memory
- Orders of magnitude overhead for remote access



Manual Optimization Techniques

- Different levels of manual optimizations
  - O0: No Optimization, O1: Privatization, O2: Aggregation
- Manual optimization helps significantly
- Up to 68x improvements in real-world applications
- Implementing locality optimization is very burdensome
- Significant increases in several productivity metrics
  - Lines of code, arithmetic operations, function calls and loops

Global memory view helps, however, the language stack should be exploited more to improve programmer productivity.



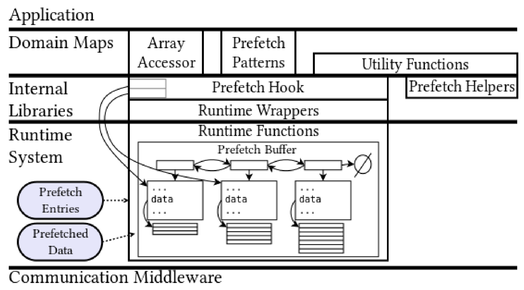
Programming Cost Metrics

	Sobel			Matrix Multiply			Matrix Transpose			Heat Diffusion		
Metric	*O0*	*O1*	*O2*	*O0*	*O1*	*O2*	*O0*	*O1*	*O2*	*O0*	*O1*	*O2*
LOC	1	13	4	4	15	9	1	26	11	8	43	78
Arithmetic	0	0	0	2	17	9	0	16	2	6	6	19
Func Call	2	17	3	0	0	0	0	7	0	4	32	38
Loops	1	5	2	2	6	1	1	2	1	1	4	15
Imp	1.0	1.8	3.8	1.0	1.1	68.1	1.0	1.8	1.7	1.0	6.1	35.7

Locality-Aware Productive Prefetching Support

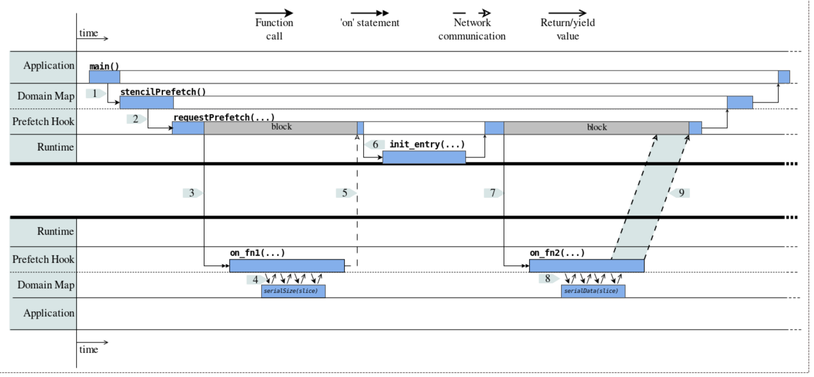
Design

- A novel programming language feature
- Exploits different capabilities of the locality-aware software stack
- Different layers of the software stack handle different tasks of data movement
- Much less programmer effort, high performance gains

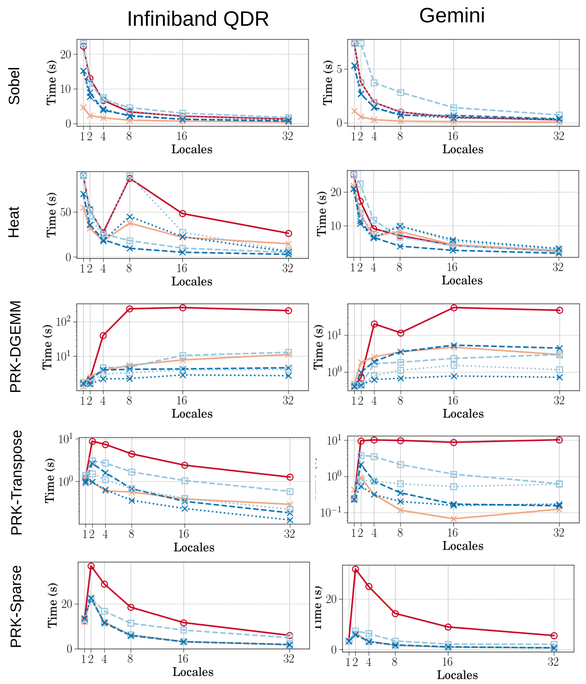


Productive
Extensible
Scalable
Efficient

- Communication protocol between compute nodes allows flexible data exchange patterns depending on the application
- Decentralized communication improves scalability
- Allows memory consistency, array reallocation
- Several small optimization in some common cases



Performance Analysis



- 2 orders of magnitude speedup over base
- Comparable or better performance than manual
- Less memory than manually optimized code
- Few lines added to the base

```
for i in 1..numIter                {
  forall (i,j) in B.domain do
    B[i,j] = A[j,i];
  forall a in A do a += 1.0; }
```

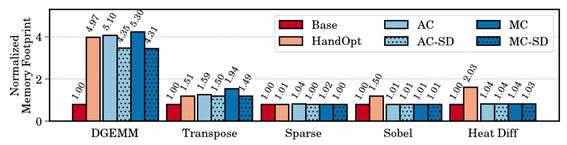
Base

```
coforall l in Locales do on l                {
  var locIdxs = B.localSubdomain();
  var tDom = {locIdxs.dim(2), locIdxs.dim(1)};
  var localA : [tDom] real ;
  for i in 1..numIter {
    localA = A[transposeDom];
    forall (i,j) in bLocSubDom do
      B[i,j] = localA[j,i];
    forall (i,j) in A.localSubdomain() do
      A [i,j] += 1.0;                        }}
```

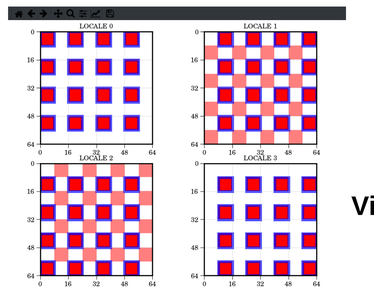
Manual

```
A.transposePrefetch();
for i in 1..numIter                {
  forall (i,j) in B.domain do
    B[i,j] = A[j,i];
  forall a in A do a += 1.0;  }
```

LAPPS

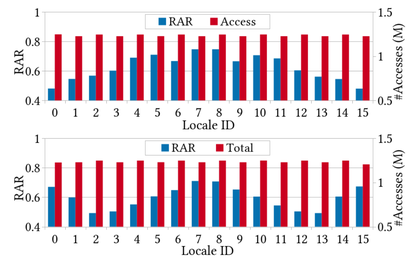


Access Pattern Analysis Tool for Distributed Arrays



Logging
Analysis
Visualization

- Assist the programmer to understand the access patterns to distributed arrays
- Helps implement data locality optimizations without thorough code inspection
- Two optimization opportunities in LULESH leading to %34 and %5 improvements



Ongoing Research

- A conceptual model for understanding distribution of labor between the programmer and the programming paradigm
  - Locality optimizations as engineering process
  - Tasks, and needed capabilities to perform them
  - Pose locality optimization implementations as a combinatorial optimization problem
- A profile-based automated optimization framework using artificial intelligence models
  - LAPPS makes data locality optimizations and application logic orthogonal
  - An AI model can predict based on training with profile data and inject calls easily

REFERENCES

[1] E. Kayraklioglu, M. Ferguson and T. El-Ghazawi, "LAPPS: Locality-Aware Productive Prefetching Support for PGAS," ACM Transactions on Code and Architecture Optimization, to appear

[2] E. Kayraklioglu and T. El-Ghazawi, "Assessing Memory Access Performance of Chapel through Synthetic Benchmarks," in 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2015, pp. 1147–1150

[3] E. Kayraklioglu, O. Serres, A. Anbar, H. Elezabi, and T. El-Ghazawi, "PGAS Access Overhead Characterization in Chapel," in 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016, pp. 1568–1577

[4] E. Kayraklioglu, W.Chang, and T. El-Ghazawi, "Comparative Performance and Optimization of Chapel in Modern Manycore Architectures" in 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 1105–1114

[5] E. Kayraklioglu, T. El-Ghazawi, "APAT: Access Pattern Analysis Tool for Distributed Arrays," in 15th ACM International Conference on Computing Frontiers (CF), 2018, pp. 248-251