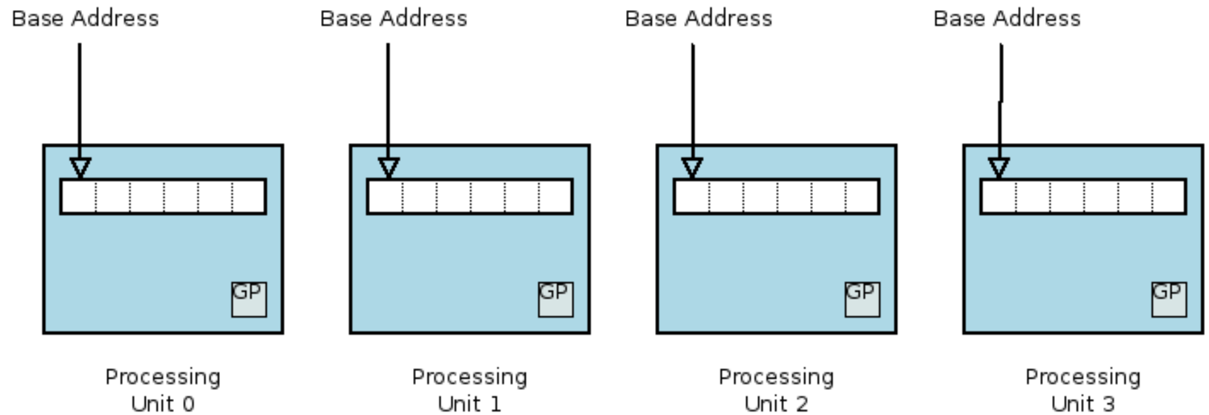
Outline

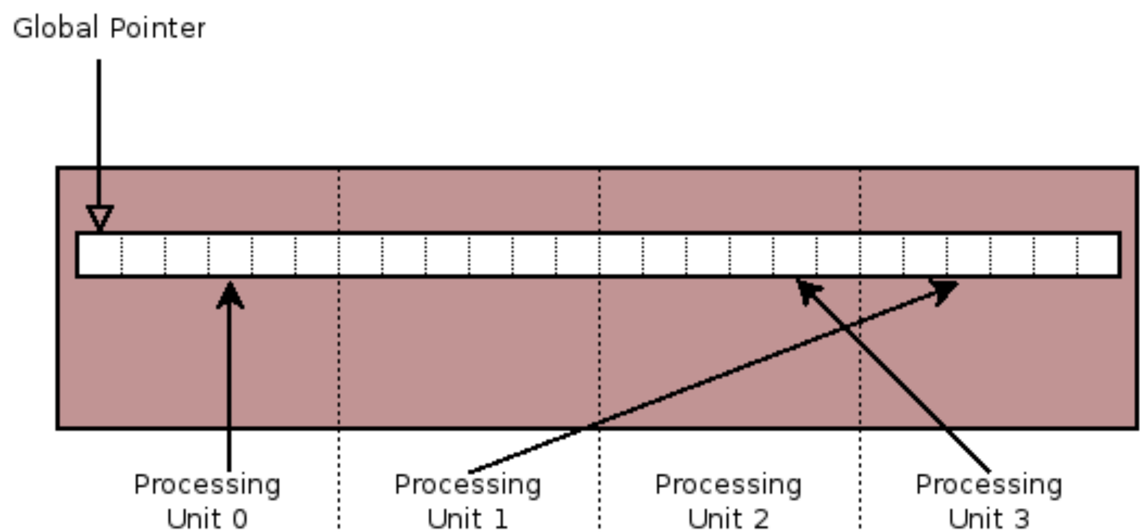
- Introduction
  - PGAS
  - Chapel
  - Motivation
- Related Studies
- Benchmarks
  - Versions
- Evaluation
- Conclusion

Introduction - PGAS

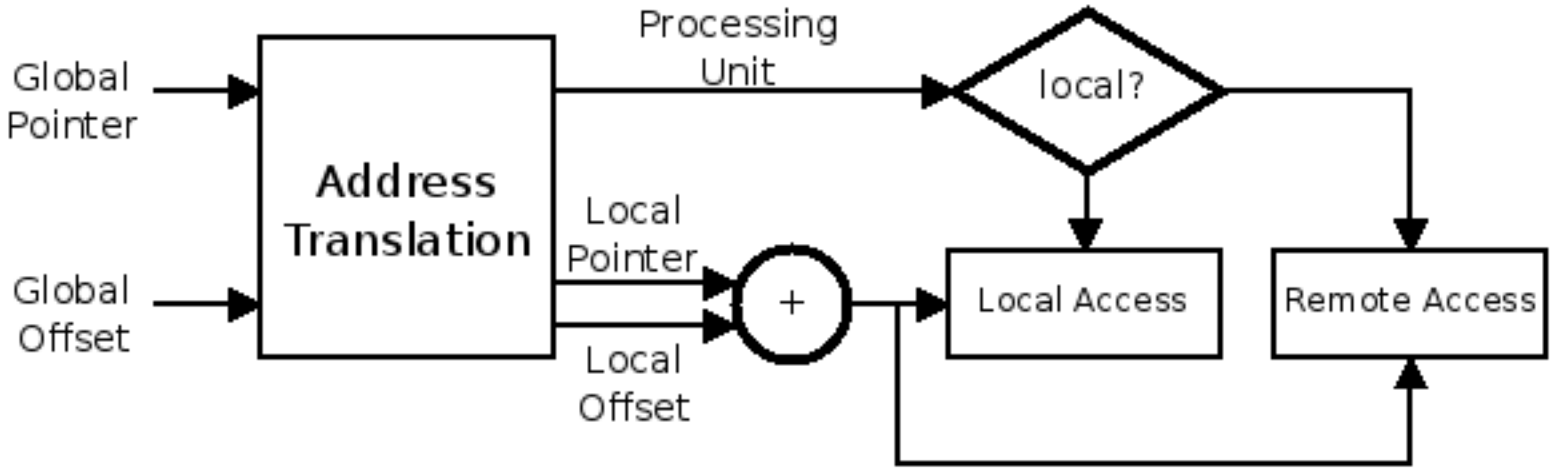
Actual



Abstraction



PGAS Access




```
const DistDom = {1..100} dmapped SomeDist();
var distArr: [DistDom] int;

writeln(distArr[14]);
```

Access Types in PGAS

	Local	Remote
Non-distributed	OK	?
distributed	Locality Check Fine Grain	Locality Check Fine grain

Chapel

- Emerging Partitioned Global Address Space language
- Carries inherent PGAS access overheads
- Programmer can mitigate overheads
- How?
- At what cost?

PGAS Access Types
in Chapel

	Local	Remote
Non-distributed	Fast	N/A
distributed	Locality Check	Fine grain

```
const ProblemSpace = {0..#N, 0..#N};
var arr : [ProblemSpace] int;
// ... some code here ...
writeln(arr[i, j]);
```

```
const DistProblemSpace = ProblemSpace dmapped Block(ProblemSpace);
var distArr: [DistProblemSpace] int;
// ... some code here ...
writeln(distArr[i, j]);
```

How to Avoid Overheads
local statement

Naive

```
forall (i,j) in distArr.domain do
    // ... find iKnowItsLocal ...
    if iKnowItsLocal then
        local writeln(distArr[i, j]);
    else
        writeln(distArr[i,j]);
```

Better

```
var localDom = {0..#SIZE/4, 0..#SIZE};
var remoteDom = {SIZE/4..SIZE, 0..#SIZE};
local forall (i,j) in localDom do
    writeln(distArr[i, j]);
forall (i,j) in remoteDom do
    writeln(distArr[i, j]);
```

How to Avoid Overheads
Bulk Copy

```
var privCopy: [ProblemSpace] int;
var copyDomain = {15..25,15..25};
privCopy[copyDomain] = distArr[copyDomain];
```

Motivation - Contribution

- Applications that have well-structured accesses to distributed data
  - Explicit domain manipulation
    - distArr.localSubdomain()
    - Other domain manipulation methods in language
  - Affine transformation;
    - Locality check avoidance
    - Bulk copy
- Performance vs productivity analysis of such transformations in application level

Relevant Related Work
PGAS

- El-Ghazawi et al., **“UPC performance and potential: A NPB experimental study”,** SC02
  - Similar study on UPC with NPB
  - Comparable performance to MPI with higher productivity
- Chen et al., **“Communication optimizations for fine-grained UPC applications”,** PACT05
  - Berkeley UPC compiler optimizations
  - Redundancy elimination, split-phase communication, message coalescing
- Alvanos et al., **“Improving performance of all-to-all communication through loop scheduling in PGAS environments”** ICS13
  - Inspector/executor logic for runtime coalescing
  - 28x speedup in UPC
- Serres et al., **“Enabling PGAS productivity with hardware support for shared address mapping: A UPC case study**”, TACO16
  - Hardware solution for wide pointer arithmetic
  - Better performance then hand optimization

Relevant Related Work
Chapel

- Hayashi et al., **''LLVM-based communication optimizations for PGAS programs'',** LLVM15
  - Language-agnostic, LLVM based optimizations
  - Remote access aggregation, locality analysis, runtime coalescing
  - Up to 3x performance
- Kayraklioglu et al., **''Assessing Memory Access Performance of Chapel through Synthetic Benchmarks'',** CCGRID15
  - Locality check avoidance gains up to 35x in random accesses
- Ferguson et al., **''Caching Puts and Gets in a PGAS Language Runtime'',** PGAS15
  - Software cache for remote data
  - Spatial and temporal locality
  - 2x improvement

Benchmarks

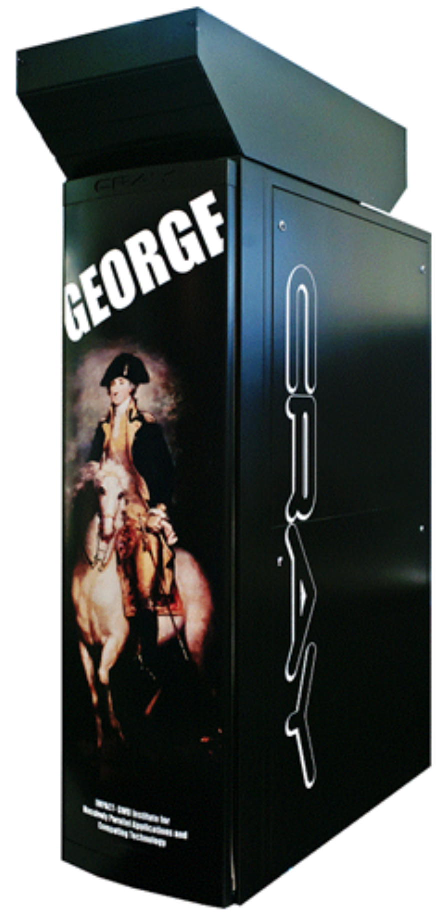
- Sobel
  - $2^{13} \times 2^{13}$
- MM
  - $C = A \times B^{T}$, $2^9 \times 2^9$
- MT
  - $2^{11} \times 2^{11}$
- 3D Heat diffusion
  - 3D, repetitive stencil
  - $2^8 \times 2^8 \times 2^8$
- STREAM
  - Full set: copy, scale, sum, triad
  - Bandwidth perspective

Versions

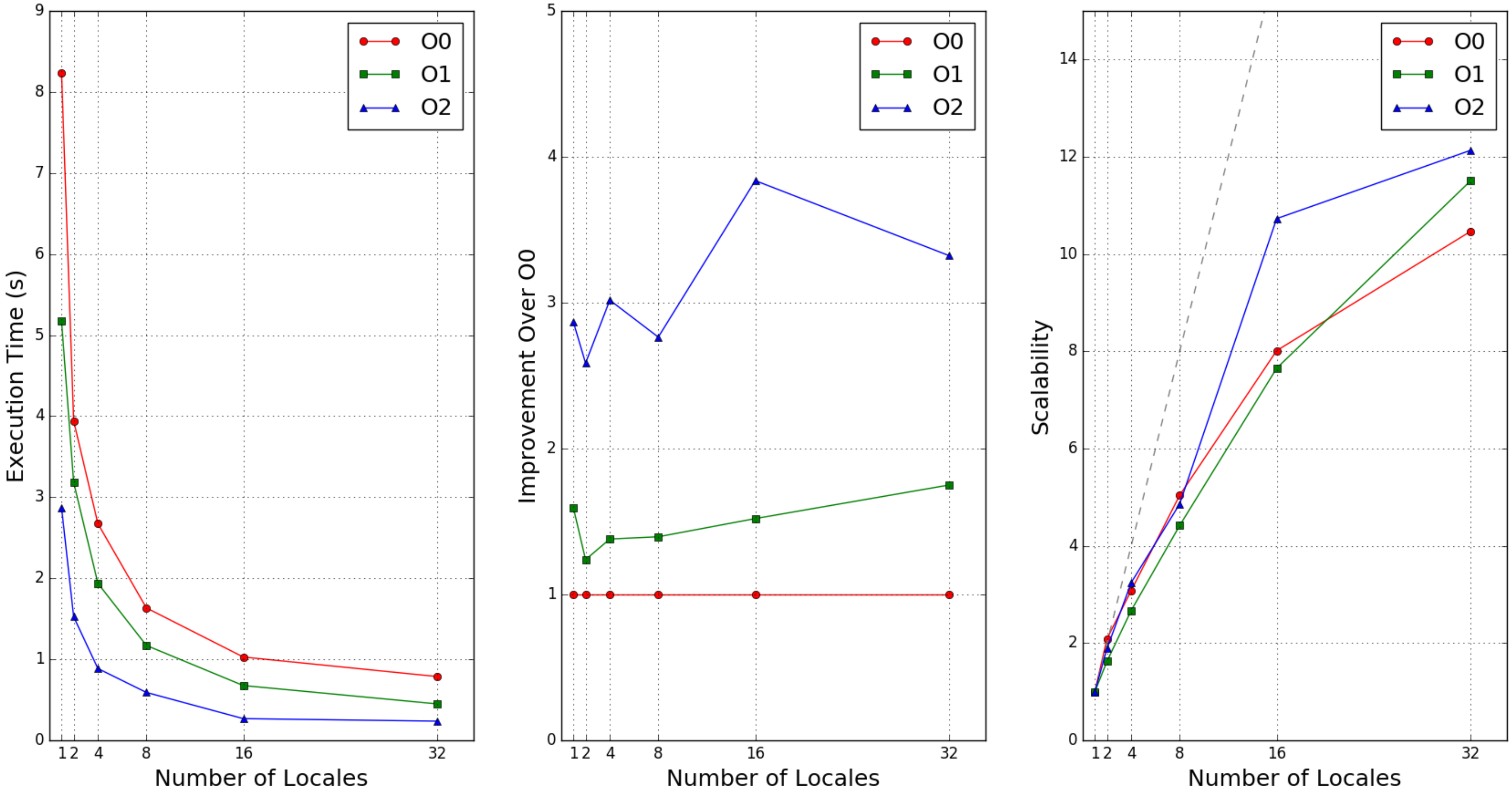
- O0
  - Simplest implementation
  - Highest programmer productivity
  - Very intuitive
- O1
  - Locality check avoidance for local accesses
  - Added programming complexity
- O2
  - Bulk copy
  - Added programming complexity(generally)

Performance Evaluation

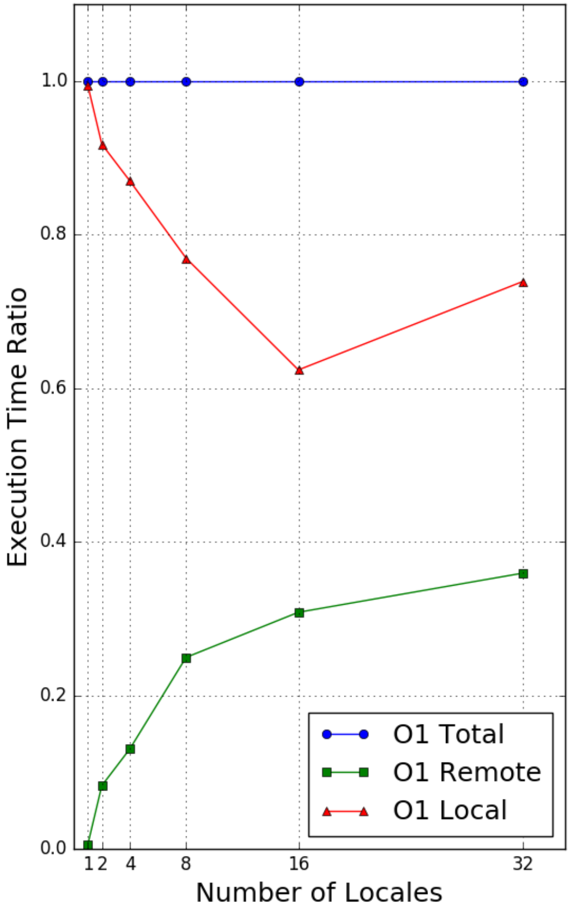
- George - Cray XE6/XK7
  - 56 nodes, dual Magny Cours with 12 hw threads each
  - Chapel version 1.12.0
  - qthreads, GasNET
  - 1-32, power-of-two nodes

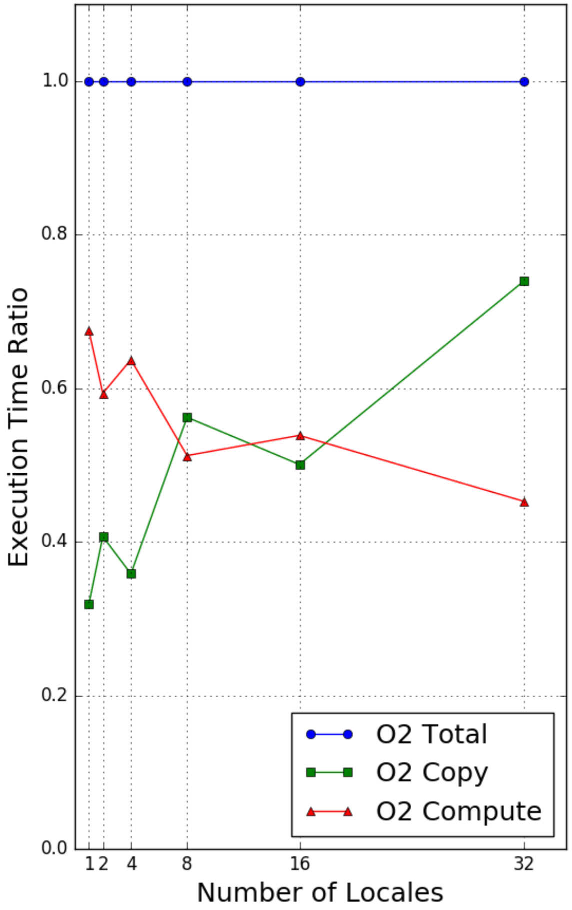


Results
Sobel

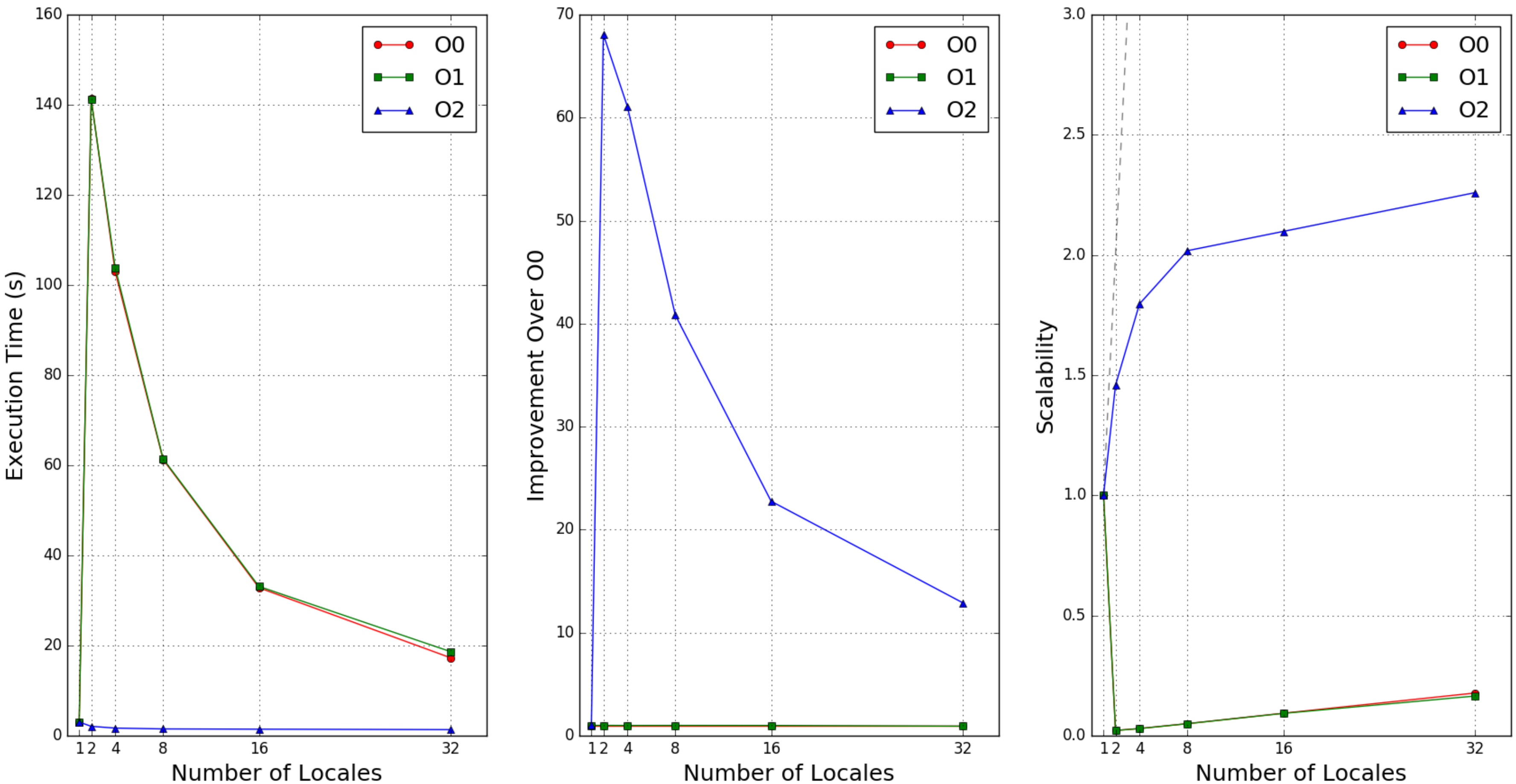


Results
Sobel - Detail



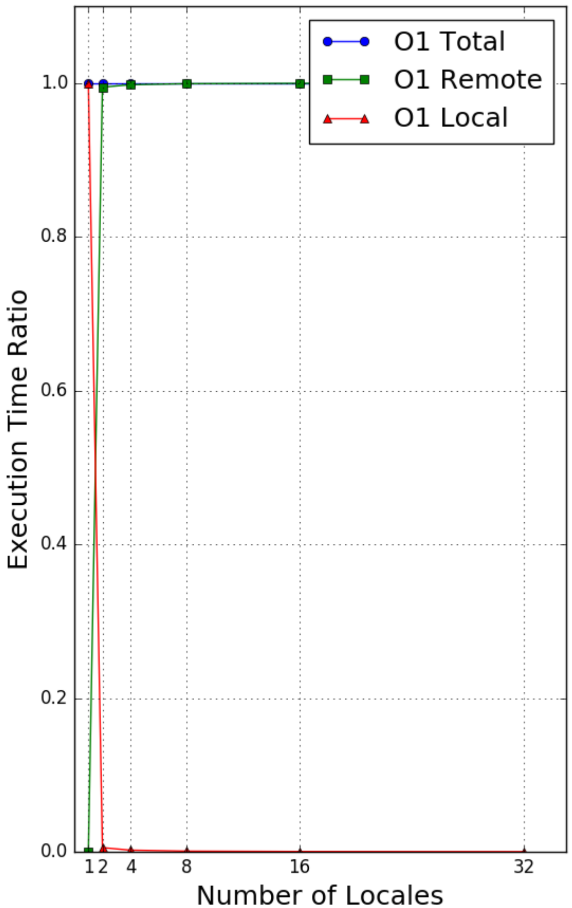


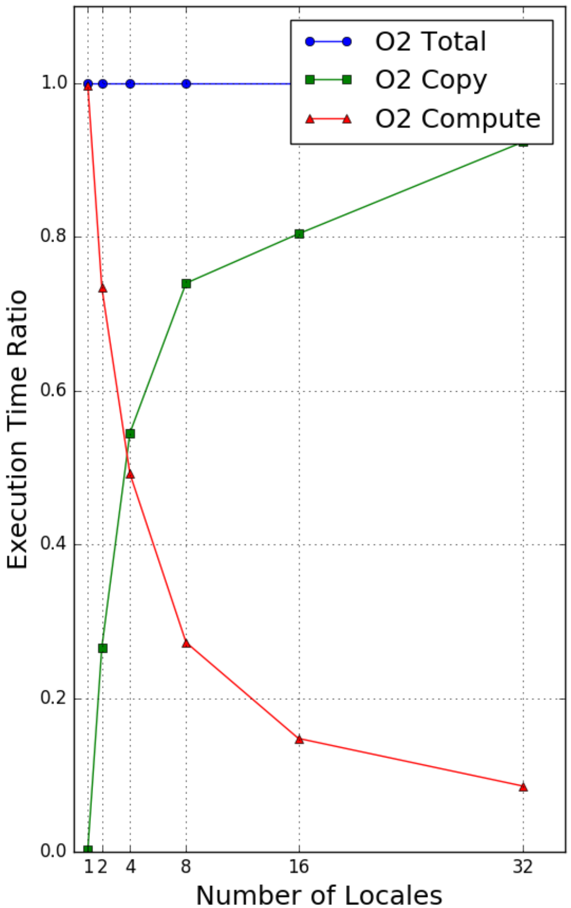
Results
MM



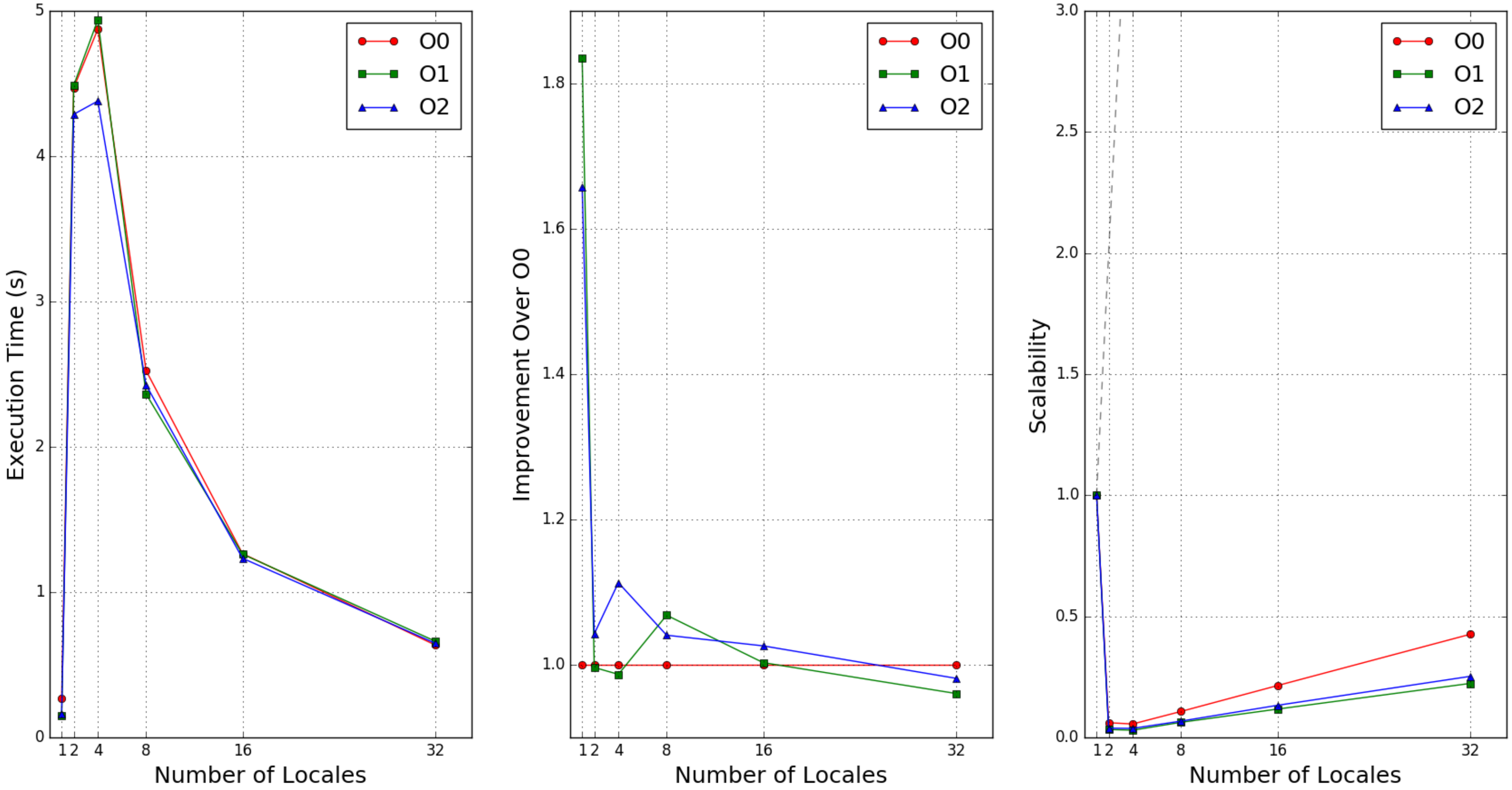
Execution Time (s)
Improvement Over O0
Scalability
Number of Locales
Number of Locales
Number of Locales
O0
O1
O2
160
140
120
100
80
60
40
20
0
70
60
50
40
30
20
10
0
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1 2 4 8 16 32

Results
MM - Detail



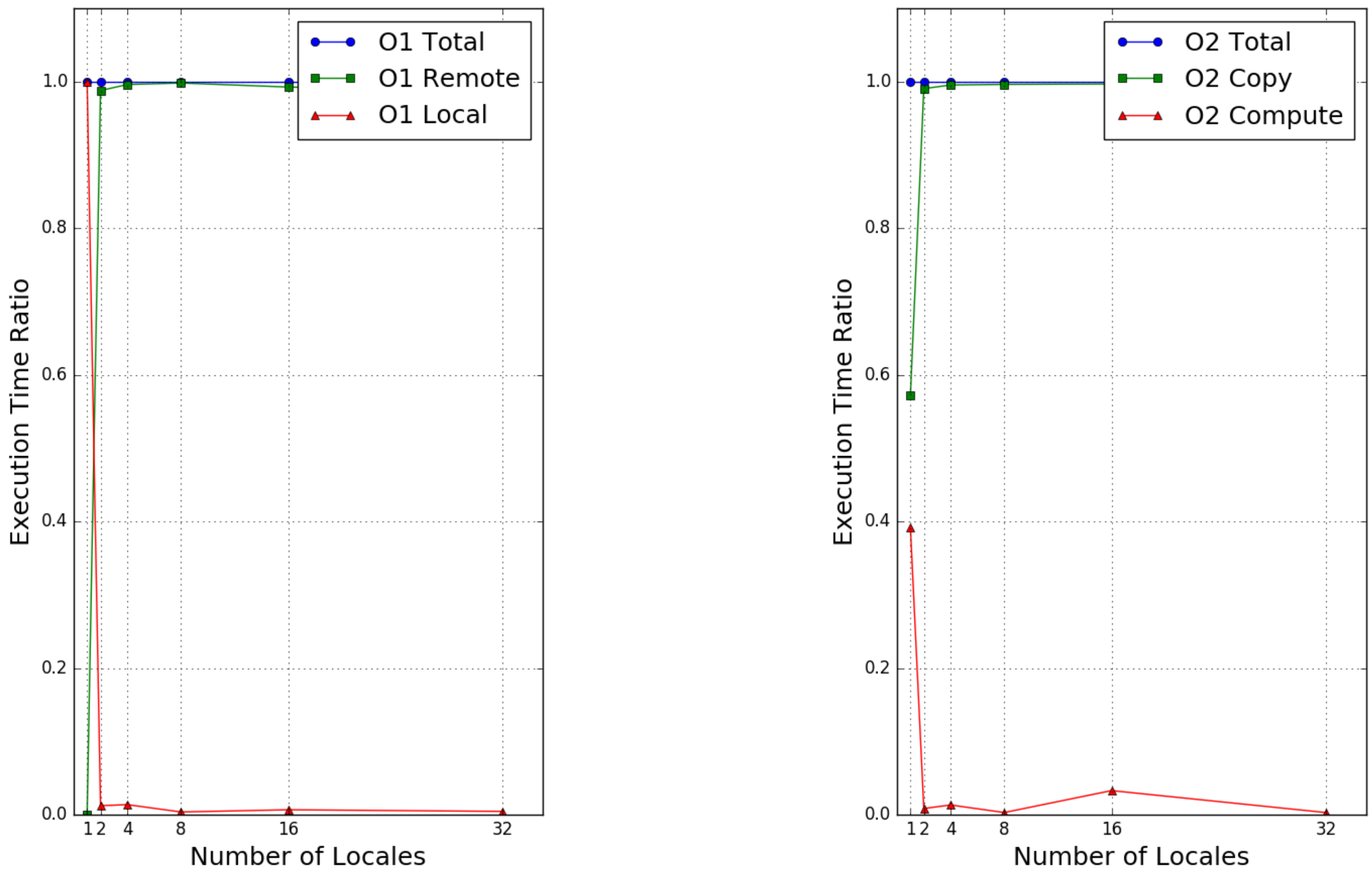


Results
MT

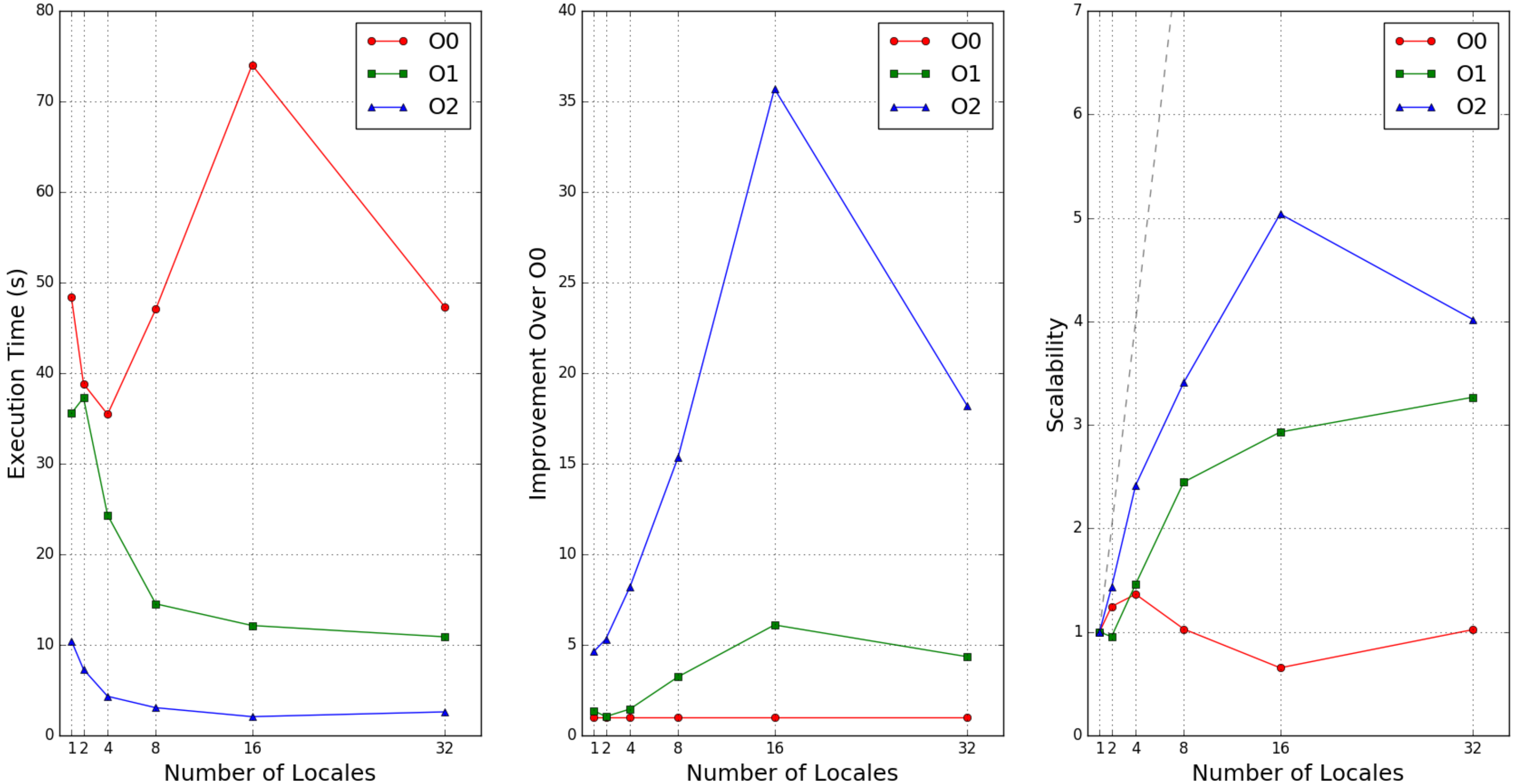


Execution Time (s)
Number of Locales
Improvement Over O0
Number of Locales
Scalability
Number of Locales
O0
O1
O2

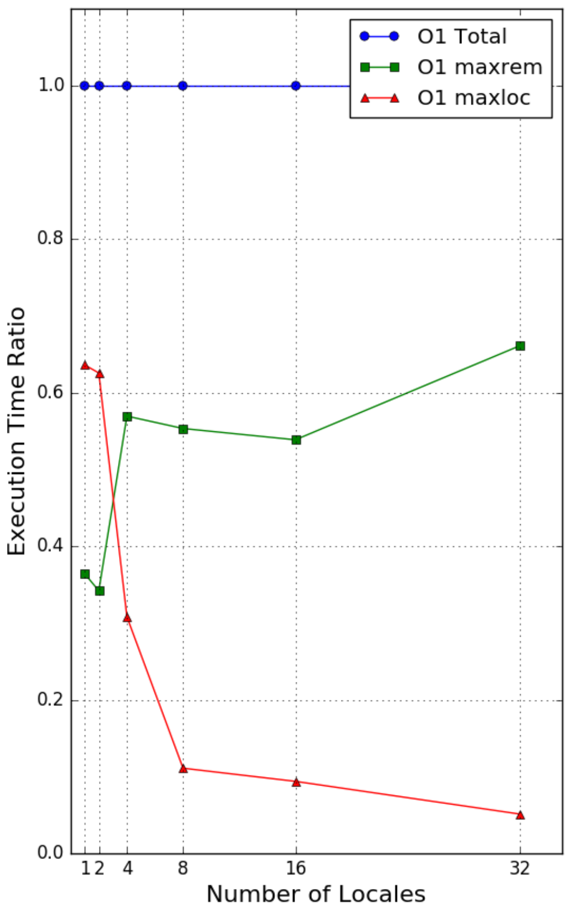
Results
MT - Detail

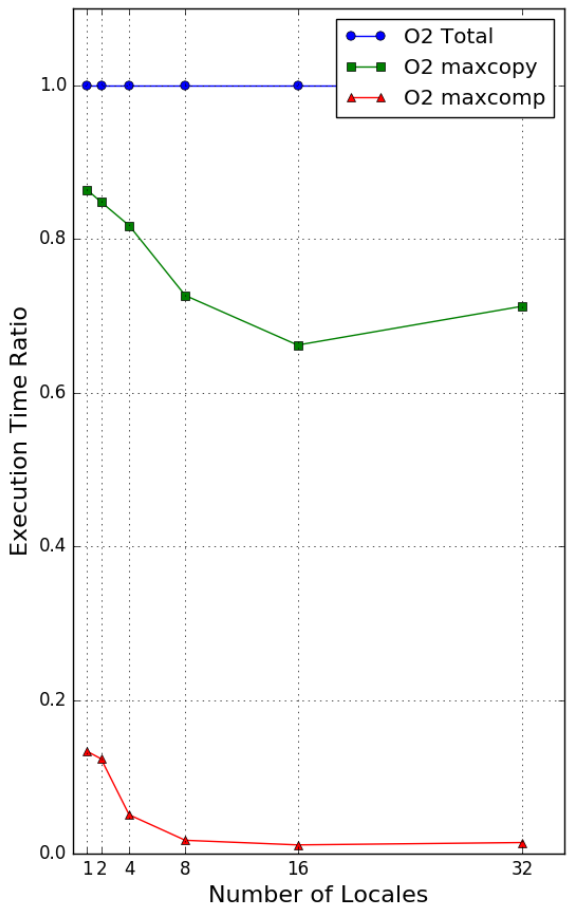


Results
3D Heat Diffusion

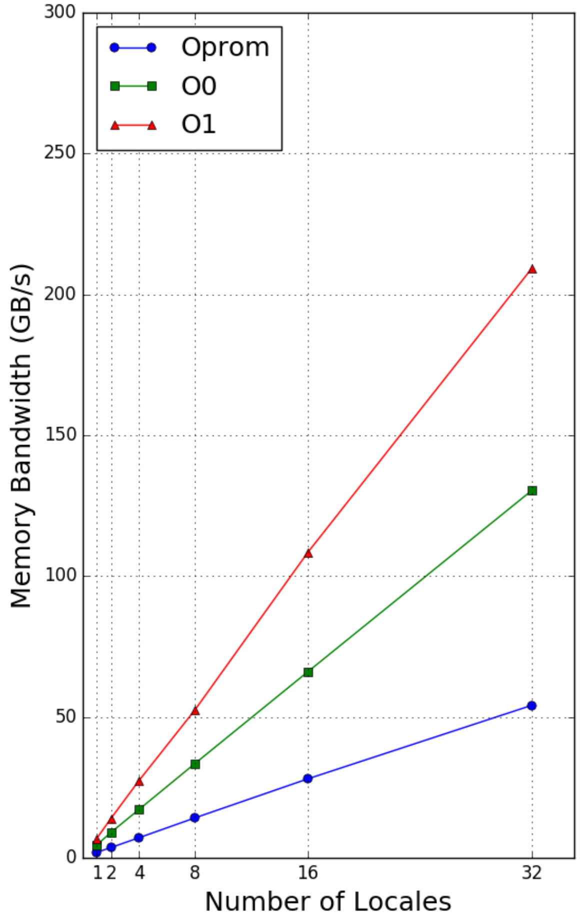


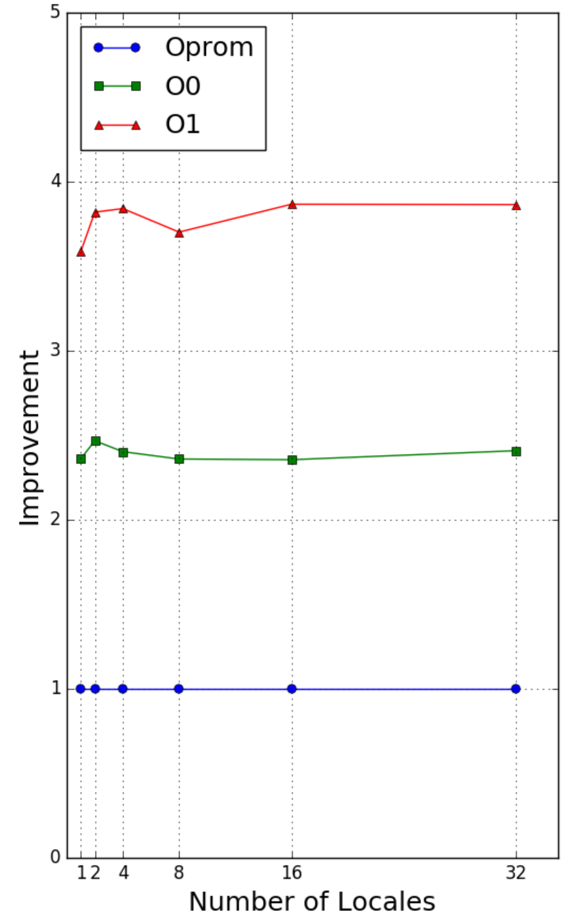
Results
3D Heat Diffusion- Detail



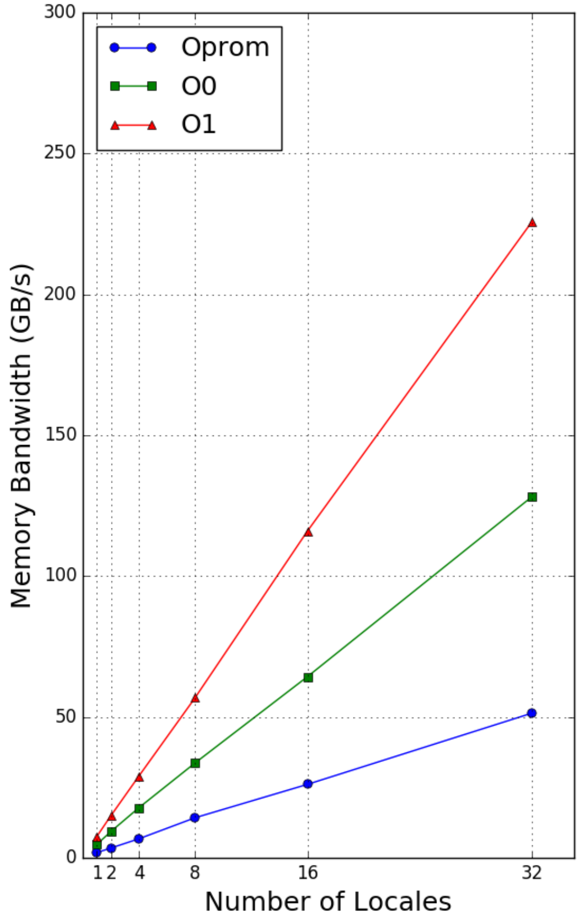


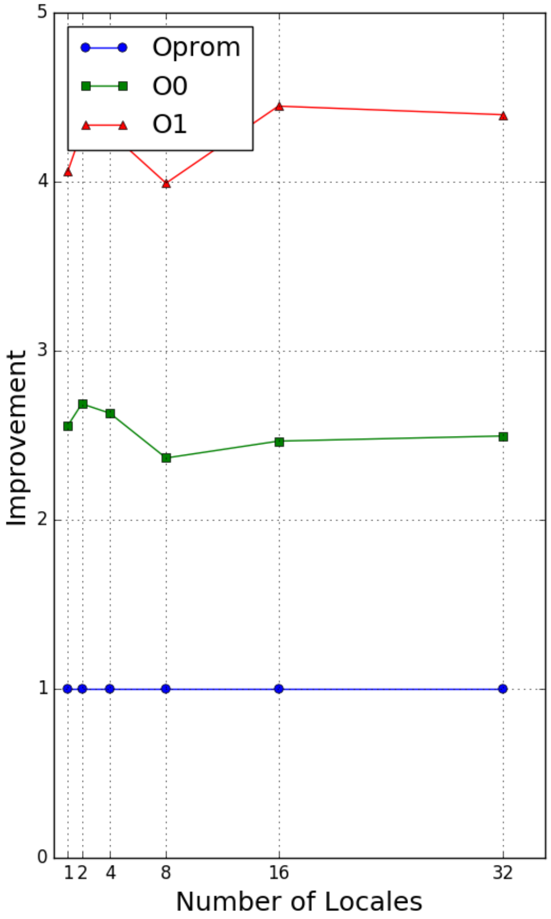
Results
Stream Scale





Results
Stream Triad





Productivity Evaluation

- What comprises “productivity”
  - How fast you learn?
  - How fast you implement?
  - How maintainable?
  - How correct?
- Qualitative, **very** subjective
- List of measures covered;
  - # lines of code,
  - # arithmetic/logic operations
  - # function calls
  - # loops

Productivity Evaluation

	Sobel			MM			MT			Heat Diff		
	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
A/L	0	0	0	2	17	9	0	16	2	6	6	19
Func	2	17	3	0	0	0	0	7	0	4	32	38
Loop	1	5	2	2	6	1	1	2	1	1	4	15
X	*1.0*	*1.8*	*3.8*	*1.0*	*1.1*	*68.1*	*1.0*	*1.8*	*1.7*	*1.0*	*6.1*	*35.7*

- O0 is highly productive
  - <10 LOC for all
- O2 seems more productive compared to O1
  - Memory footprint of O2 is not studied

Possible Directions

- More breadth
  - Sparse arrays
  - Task parallelism
  - Different applications
- More depth
  - Low-level routines, extern C functions
  - A productivity model
  - … vs Memory vs power

Recap

- PGAS access characteristics
- Application-level optimizations
- Performance vs Productivity

- Compile time affine transforms
- Runtime prefetching

Thank you

engin@gwu.edu

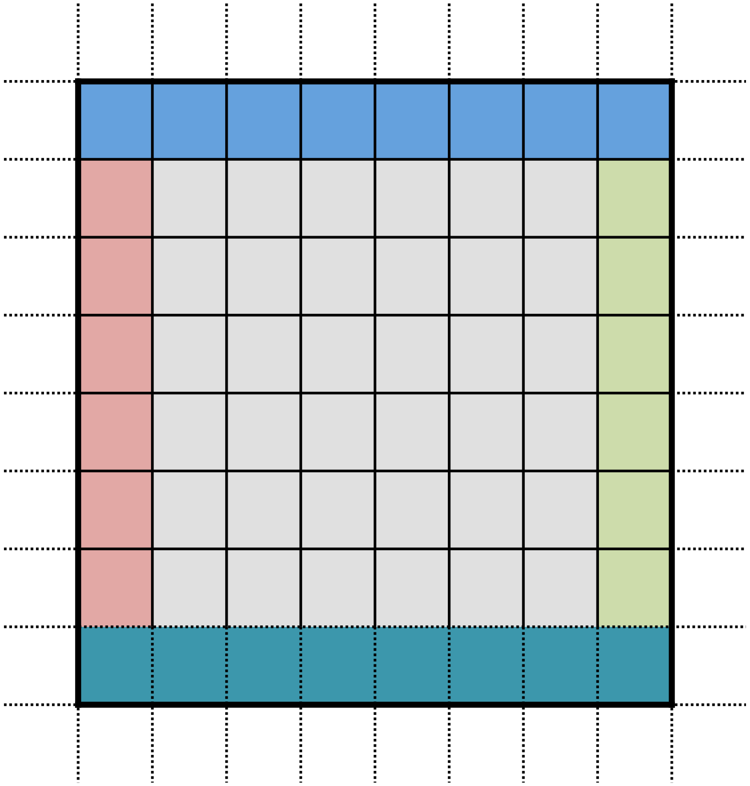
Backups

Productivity Evaluation
Sobel

- O1
  - Local subdomain queries
  - Rectangular domain methods

	Sobel		
	O0	**O1**	**O2**
LOC	1	13	4
A/L	0	0	0
Func	2	17	3
Loop	1	5	2
X	*1.0*	*1.8*	*3.8*

- O2
  - bulk copy of local subdomain expanded by 1



Productivity Evaluation
MM

- O1
  - Subdomains are calculated arithmetically
- O2
  - Manual replication

	MM		
	O0	**O1**	**O2**
LOC	4	15	9
A/L	2	17	9
Func	0	0	0
Loop	2	6	1
X	*1.0*	*1.1*	*68.1*