Using Cowichan Problems to Investigate Programmability of X10 Programming System

Jeeva S. Paudel, J. Nelson Amaral

Department of Computing Science

University of Alberta, Edmonton

Canada

June 4, 2011

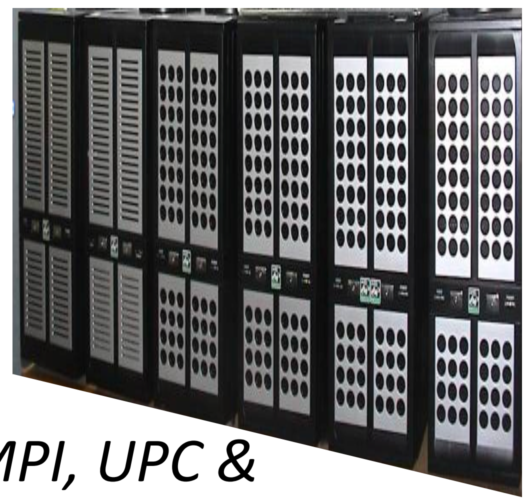
X10: Design Goals

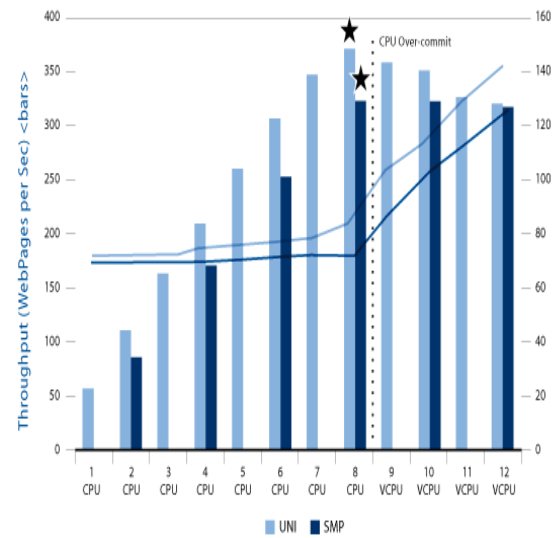
Programmer Productivity

An experiment to evaluate the programmability of three programming systems.

(Sarkar *et al.*, Workshop on Productivity & Performance in High-End Computing 2006)

Code Performance





Performance evaluation of MPI, UPC & OpenMP on multicore architectures.

(Mallon *et al.*, EuroPVM/MPI 2009)

Contributions

Performance assessment of X10

Implementation of a suite of parallel programs

Cowichan Suite

Gregory Wilson

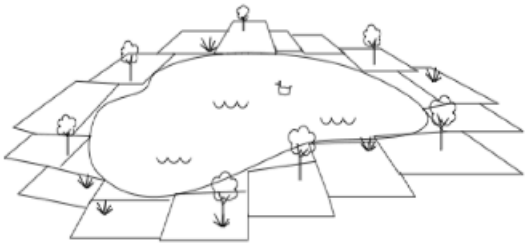
Salishan -- Cowichan

Cowichan Suite

Designed to assess the programmability of a programming system
for
writing efficient parallel programs

Cowichan Problems

Communication & Dynamic Load Balancing

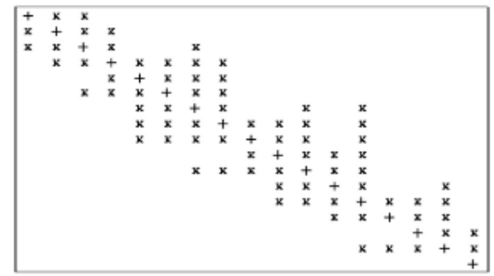


1. Turing Ring Problem

Input/Output & Intense Communication

5. Image Thinning Problem

Data Replication & Coherence



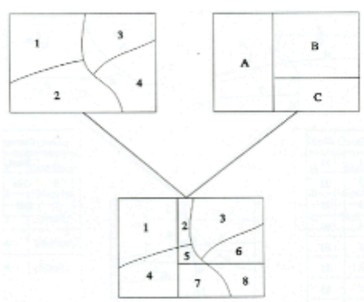
2. Skyline Matrix Problem

Data Distribution & Coherence

$((AB)C)D = (AB)(CD) = A(B(CD)) = A(BC)D$

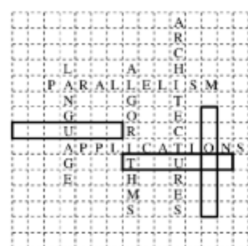
7. Matrix Chain Problem

Concurrent Input/Output



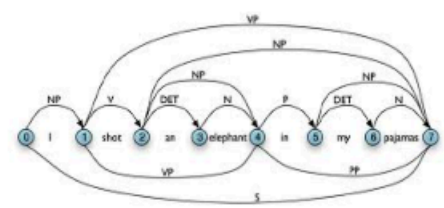
4. Polygon Overlay Problem

Task & Data Parallelism



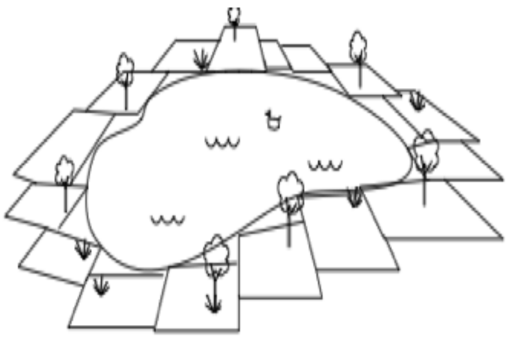
6. Kece Problem

Intense Communication



3. Active Chart Parsing Problem

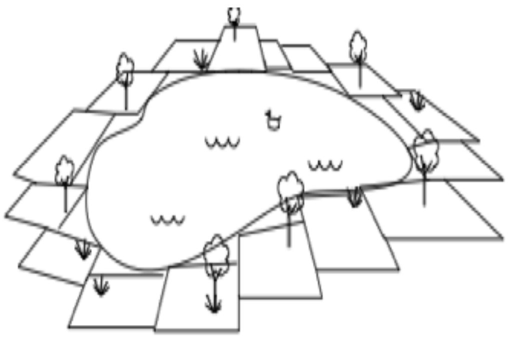
Example Implementations
Turing Ring



$$\frac{dX_i}{dt} = X_i(r_x + c_{xx}X_i + c_{xy}Y_i) + \mu_x(X_{i+1} + X_{i-1} - 2X_i)$$
$$\frac{dY_i}{dt} = Y_i(r_y + c_{yx}X_i + c_{yy}Y_i) + \mu_y(Y_{i+1} + Y_{i-1} - 2Y_i)$$

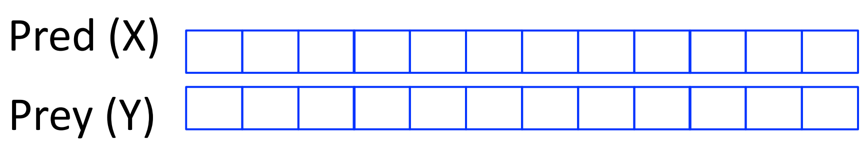
$r_{x,y}$: birth rates $c_{x,y}$: death rates $\mu_{x,y}$: migration rates

Example Implementations
Turing Ring



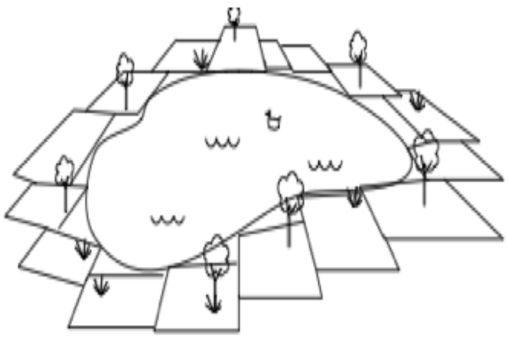
$$\frac{dX_i}{dt} = X_i(r_x + c_{xx}X_i + c_{xy}Y_i) + \mu_x(X_{i+1} + X_{i-1} - 2X_i)$$
$$\frac{dY_i}{dt} = Y_i(r_y + c_{yx}X_i + c_{yy}Y_i) + \mu_y(Y_{i+1} + Y_{i-1} - 2Y_i)$$

$r_{x,y}$: birth rates $c_{x,y}$: death rates $\mu_{x,y}$: migration rates



Sequential Implementation of Turing Ring Problem

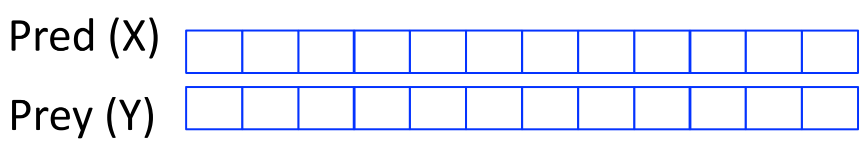
Example Implementations
Turing Ring



$$\frac{dX_i}{dt} = X_i(r_x + c_{xx}X_i + c_{xy}Y_i) + \mu_x(X_{i+1} + X_{i-1} - 2X_i)$$
$$\frac{dY_i}{dt} = Y_i(r_y + c_{yx}X_i + c_{yy}Y_i) + \mu_y(Y_{i+1} + Y_{i-1} - 2Y_i)$$

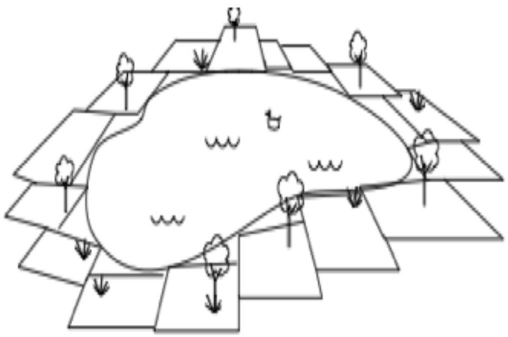
$r_{x,y}$: birth rates $c_{x,y}$: death rates $\mu_{x,y}$: migration rates

- Receive migration information
- Update local population using birth and death rates
- Update local population using migration values
- Send migration information to adjacent locations



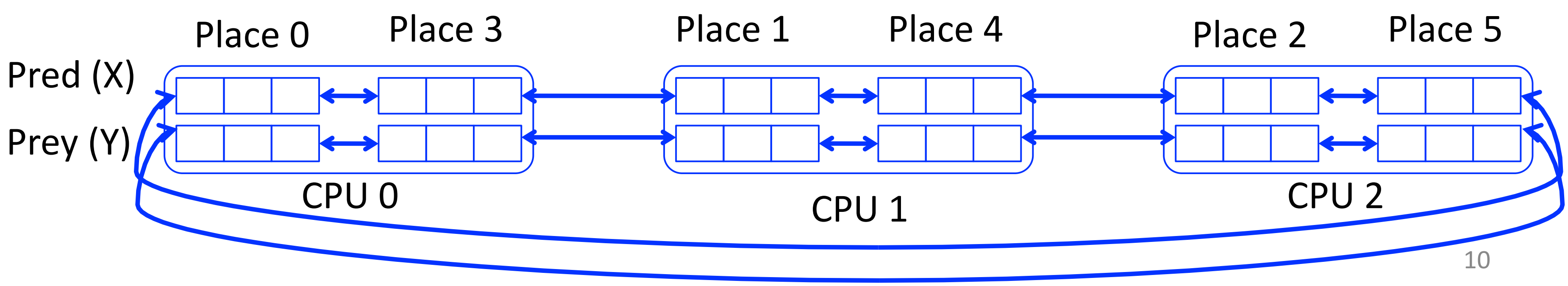
Sequential Implementation of Turing Ring Problem

Example Implementations
Turing Ring



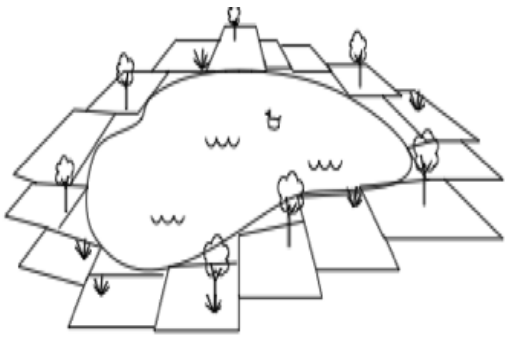
$$\frac{dX_i}{dt} = X_i(r_x + c_{xx}X_i + c_{xy}Y_i) + \mu_x(X_{i+1} + X_{i-1} - 2X_i)$$
$$\frac{dY_i}{dt} = Y_i(r_y + c_{yx}X_i + c_{yy}Y_i) + \mu_y(Y_{i+1} + Y_{i-1} - 2Y_i)$$

$r_{x,y}$: birth rates $c_{x,y}$: death rates $\mu_{x,y}$: migration rates



Parallelization of Turing Ring Problem

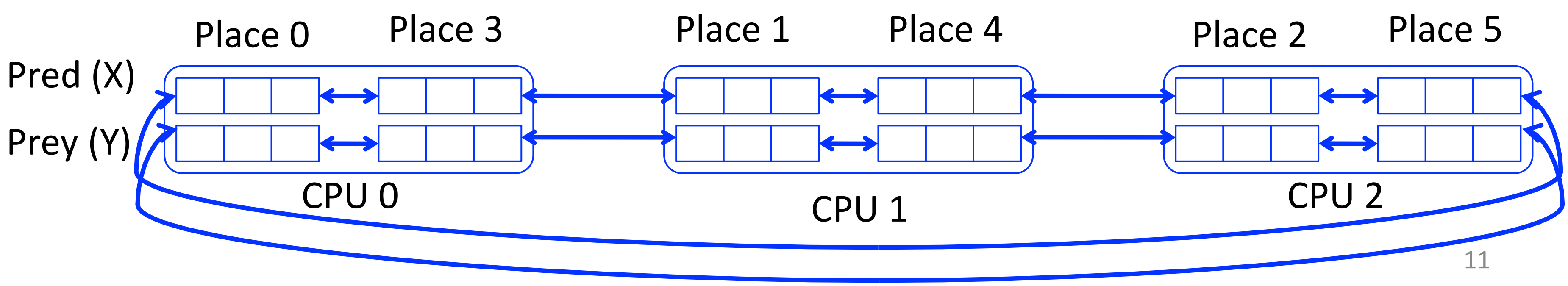
Example Implementations Turing Ring



$$\frac{dX_i}{dt} = X_i(r_x + c_{xx}X_i + c_{xy}Y_i) + \mu_x(X_{i+1} + X_{i-1} - 2X_i)$$
$$\frac{dY_i}{dt} = Y_i(r_y + c_{yx}X_i + c_{yy}Y_i) + \mu_y(Y_{i+1} + Y_{i-1} - 2Y_i)$$

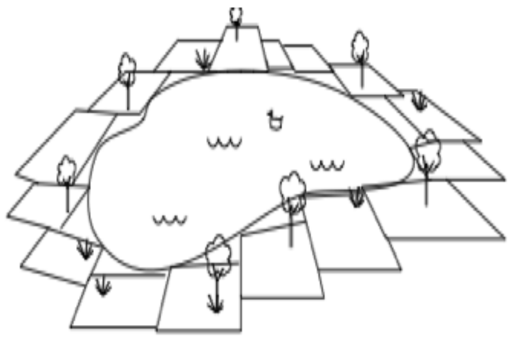
$r_{x,y}$: birth rates $c_{x,y}$: death rates $\mu_{x,y}$: migration rates

- Receive migration information
- Update local population using birth and death rates
- Update local population using migration values
- Send migration information to adjacent node
- Wait for other process to finish this iteration



Parallelization of Turing Ring Problem

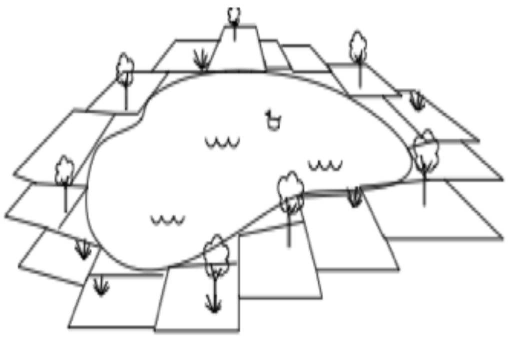
Example Implementations
Turing Ring



- Receive migration information
- Update local population using birth and death rates
- Update local population using migration values
- Send migration information to adjacent node
- Wait for other process to finish this iteration

```
for (i in per_place_region)
   clocked async {
   yk1(i) = step * eval(1,x,y,i);
   yt1(i) = y(i) + 0.5 * yk1(i);
   xk1(i) = step * eval(2, x, y, i);
   xt1(i) = x(i) + 0.5 * xk1(i);
   ...
  }
```

Example Implementations
Turing Ring

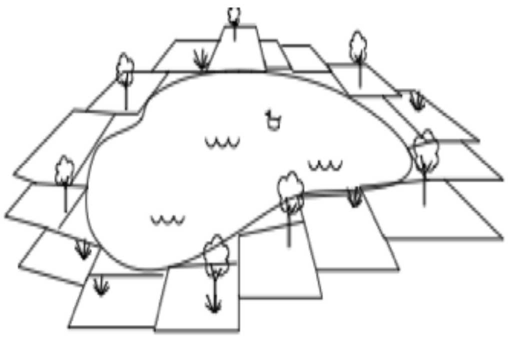


- Receive migration information
- Update local population using birth and death rates
- Update local population using migration values
- Send migration information to adjacent node
- Wait for other process to finish this iteration

```
for (i in per_place_region)
   clocked async {
   yk1(i) = step * eval(1,x,y,i);
   yt1(i) = y(i) + 0.5 * yk1(i);
   xk1(i) = step * eval(2, x, y, i);
   xt1(i) = x(i) + 0.5 * xk1(i);
   ...
  }
```

```
for (i in per_place_region)
    clocked async {
    yk2(i) = step * eval(1,xt1,yt1,i);
    yt2(i) = y(i) + 0.5 * yk2(i);
    xk2(i) = step * eval(2, xt1, yt1, i);
    xt2(i) = x(i) + 0.5 * xk2(i);
    ...
 }
```

Example Implementations
Turing Ring

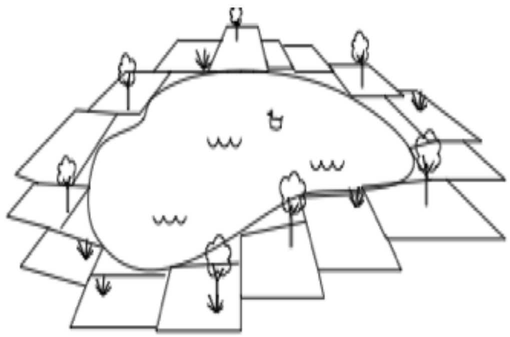


- Receive migration information
- Update local population using birth and death rates
- Update local population using migration values
- Send migration information to adjacent node
- Wait for other process to finish this iteration

```
for (z in base_dist)
   clocked async at (base_dist(z)) {
   ...
   for (i in per_place_region)
      clocked async {
      yk1(i) = step * eval(1,x,y,i);
      yt1(i) = y(i) + 0.5 * yk1(i);
      xk1(i) = step * eval(2, x, y, i);
      xt1(i) = x(i) + 0.5 * xk1(i);
      ...
     }


   for (i in per_place_region)
       clocked async {
       yk2(i) = step * eval(1,xt1,yt1,i);
       yt2(i) = y(i) + 0.5 * yk2(i);
       xk2(i) = step * eval(2, xt1, yt1, i);
       xt2(i) = x(i) + 0.5 * xk2(i);
       ...
    }


}
```

Example Implementations

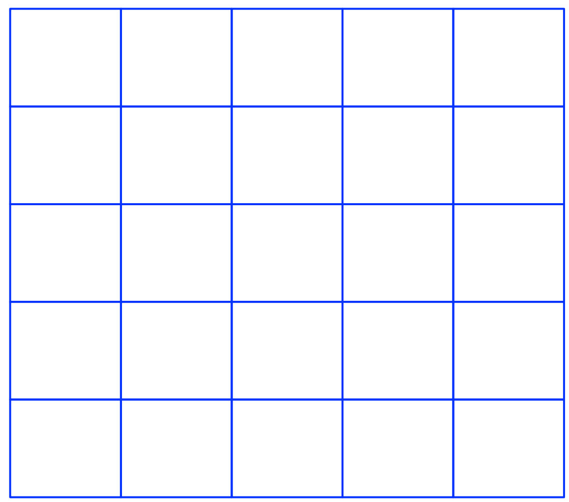
Turing Ring



- Receive migration information
- Update local population using birth and death rates
- Update local population using migration values
- Send migration information to adjacent node
- Wait for other process to finish this iteration

```
clocked finish {
  val base_dist = y.dist;
  val per_place_reg = y.region;
  for (z in base_dist)
     clocked async at (base_dist(z)) {
     ...
     for (i in per_place_region)
        clocked async {
        yk1(i) = step * eval(1,x,y,i);
        yt1(i) = y(i) + 0.5 * yk1(i);
        xk1(i) = step * eval(2, x, y, i);
        xt1(i) = x(i) + 0.5 * xk1(i);
        ...
      }
     next;
     ...
     for (i in per_place_region)
         clocked async {
         yk2(i) = step * eval(1,xt1,yt1,i);
         yt2(i) = y(i) + 0.5 * yk2(i);
         xk2(i) = step * eval(2, xt1, yt1, i);
         xt2(i) = x(i) + 0.5 * xk2(i);
         ...
      }
     next;
     ...
  }
}
```

Example Implementations
Skyline Matrix



N *X* N

Example Implementations
Skyline Matrix

29	1			
	12	2	5	
	2	8	10	
		16	11	
	22	7	3	25

N *X* N

Example Implementations
Skyline Matrix

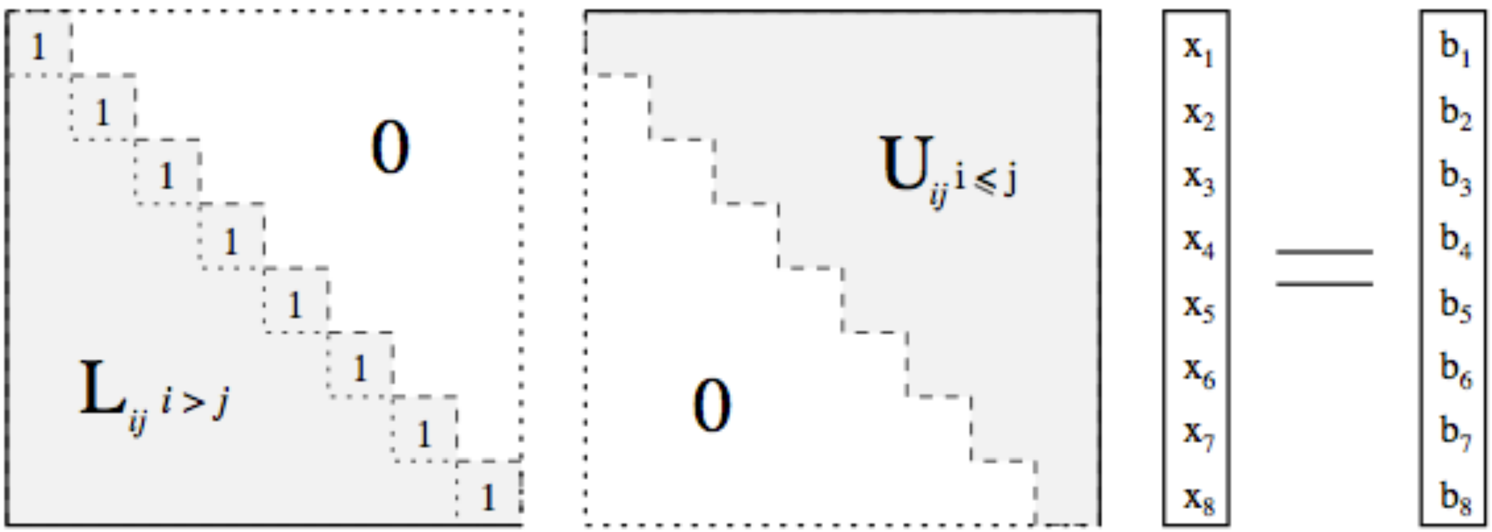
$$Ax = b$$

29	1	0	0	0
0	12	2	5	0
0	2	8	10	0
0	0	16	11	0
0	22	7	3	25

N *X* N

Example Implementations
Skyline Matrix

Ax = b

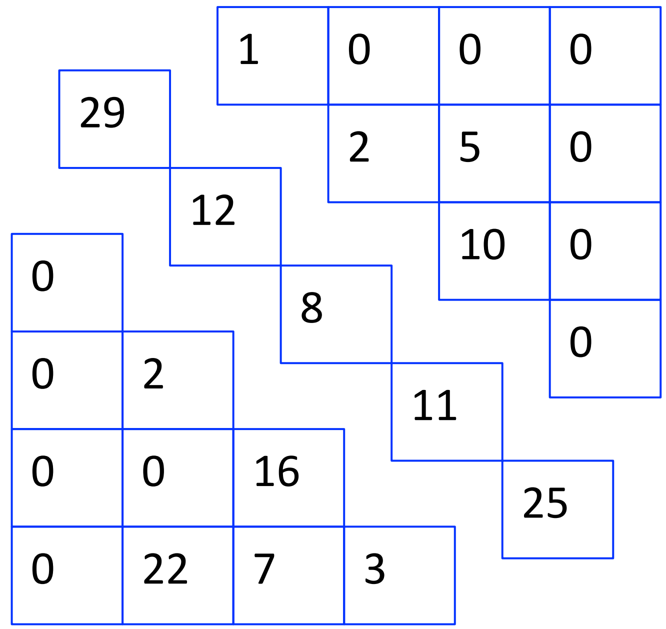


LU Decomposition

Example Implementations
Skyline Matrix

29	1	0	0	0
0	12	2	5	0
0	2	8	10	0
0	0	16	11	0
0	22	7	3	25

A



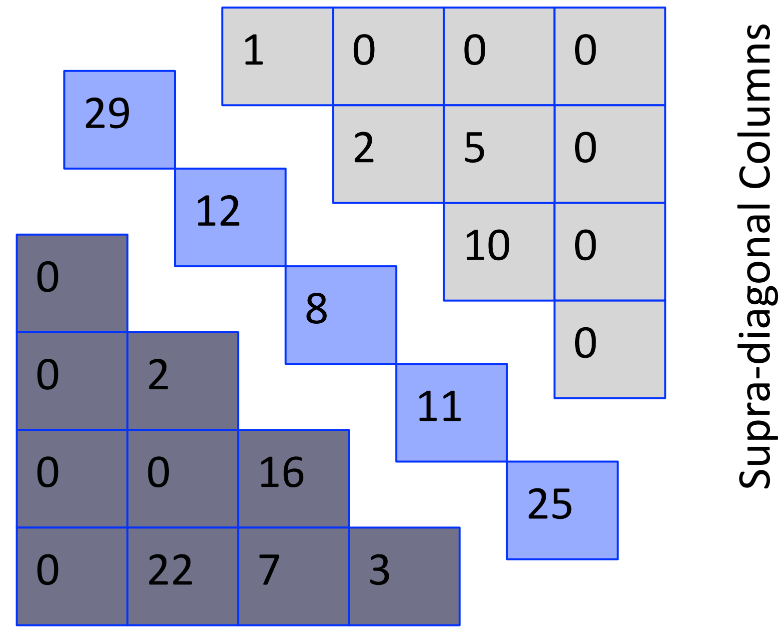
Supra-diagonal Columns

Sub-diagonal Rows

Example Implementations
Skyline Matrix

29	1	0	0	0
0	12	2	5	0
0	2	8	10	0
0	0	16	11	0
0	22	7	3	25

A

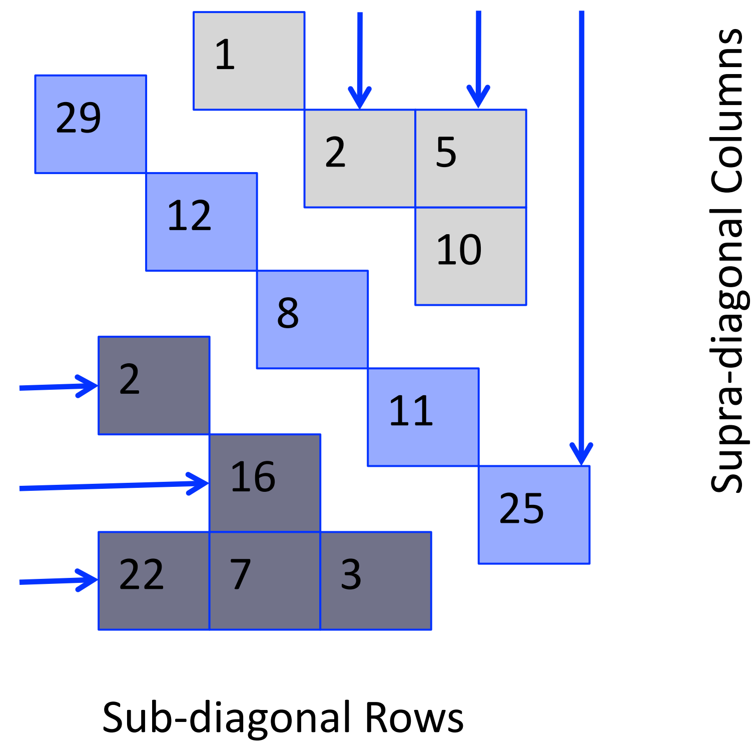


Sub-diagonal Rows

Example Implementations
Skyline Matrix

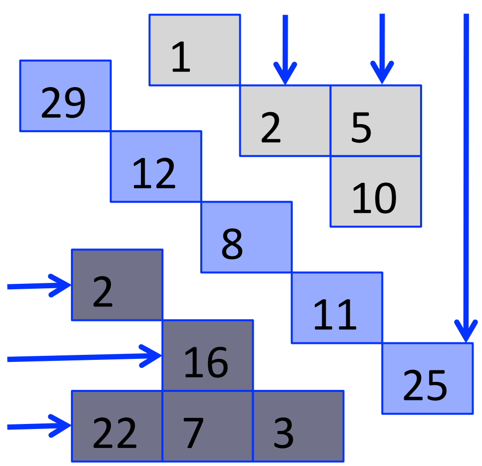
29	1	0	0	0
0	12	2	5	0
0	2	8	10	0
0	0	16	11	0
0	22	7	3	25

A



Example Implementations
Skyline Matrix

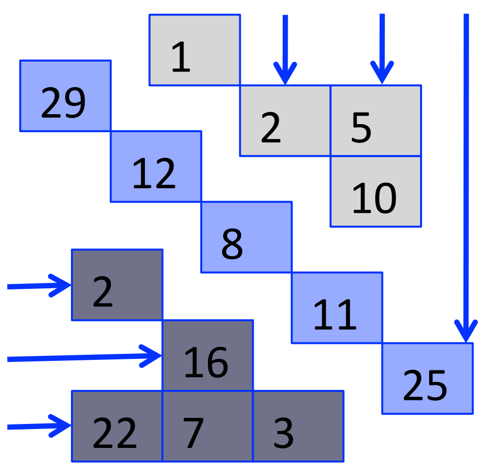
29	1	0	0	0
0	12	2	5	0
0	2	8	10	0
0	0	16	11	0
0	22	7	3	25



Example Implementations
Skyline Matrix

29	1	0	0	0
0	12	2	5	0
0	2	8	10	0
0	0	16	11	0
0	22	7	3	25

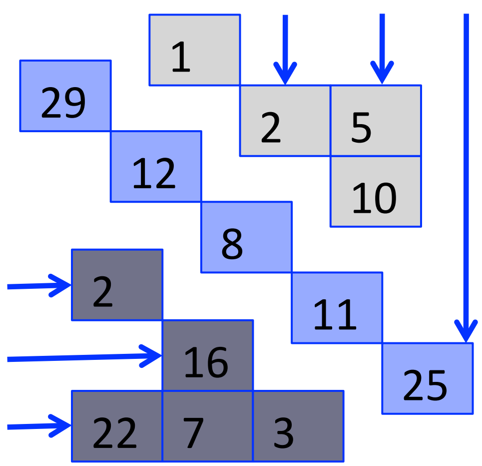
```
/* Row indices where zero-prefix ends */
lower(1..N) = {0, 2, 2, 3, 2}
```



Example Implementations
Skyline Matrix

29	1	0	0	0
0	12	2	5	0
0	2	8	10	0
0	0	16	11	0
0	22	7	3	25

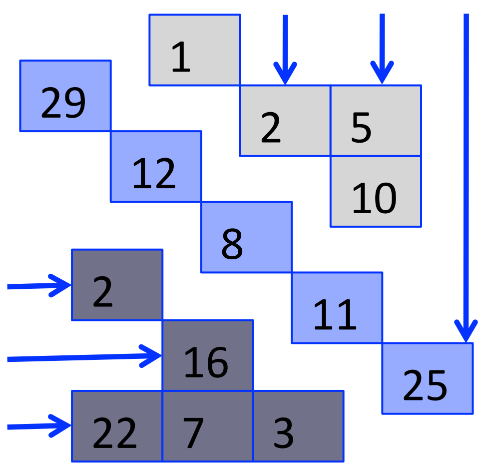
```
/* Row indices where zero-prefix ends */
lower(1..N) = {0, 2, 2, 3, 2}

/* Column indices where zero-prefix ends */
upper(1..N) = {1, 1, 2, 2, 5}
```



Example Implementations
Skyline Matrix

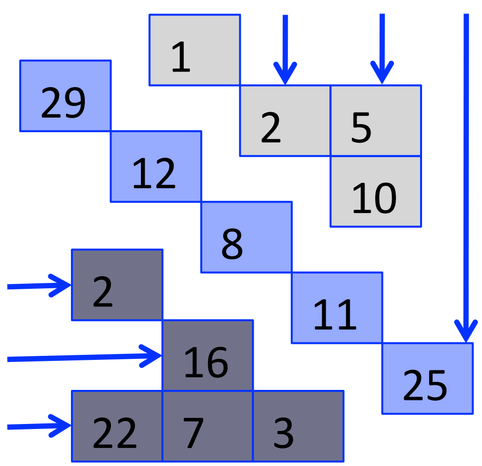
29	1	0	0	0
0	12	2	5	0
0	2	8	10	0
0	0	16	11	0
0	22	7	3	25

```
/* Row indices where zero-prefix ends */
lower(1..N) = {0, 2, 2, 3, 2}

/* Column indices where zero-prefix ends */
upper(1..N) = {1, 1, 2, 2, 5}

/* Array Region for sub-diagonals rows */
for i  in 1..N
var region_row_i <: Region = lower[i] .. (lower[i]==0)? 0: i-1

/* Array Region for supra-diagonal columns */
for i in 1 .. N
var region_column_i <: Region = upper[i] .. i
```



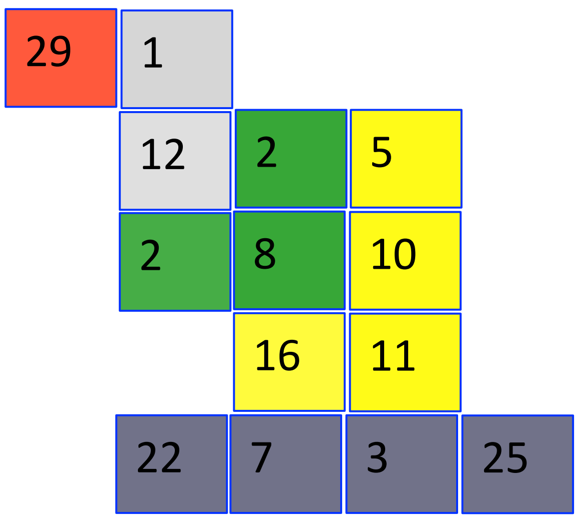
Example Implementations
Skyline Matrix

29	1	0	0	0
0	12	2	5	0
0	2	8	10	0
0	0	16	11	0
0	22	7	3	25




```
/* Row indices where zero-prefix ends */
lower(1..N) = {0, 2, 2, 3, 2}

/* Column indices where zero-prefix ends */
upper(1..N) = {1, 1, 2, 2, 5}

/* Array Region for sub-diagonals rows */
for i  in 1..N
var region_row_i <: Region = lower[i] .. (lower[i]==0)? 0: i-1

/* Array Region for supra-diagonal columns */
for i in 1 .. N
var region_column_i <: Region = upper[i] .. i

/* Define separate arrays to store each sub-diagonal row elements */
for i in 1 .. N
SubRowArray[i] (region_row_i)

/* Define separate arrays to store each supra-diagonal column elements */
for i in 1 .. N
SupraColumnArray[i] (region_column_i)
```

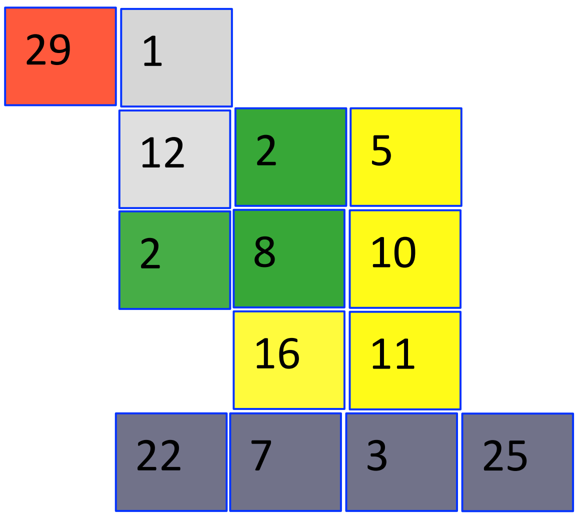
Example Implementations
Skyline Matrix



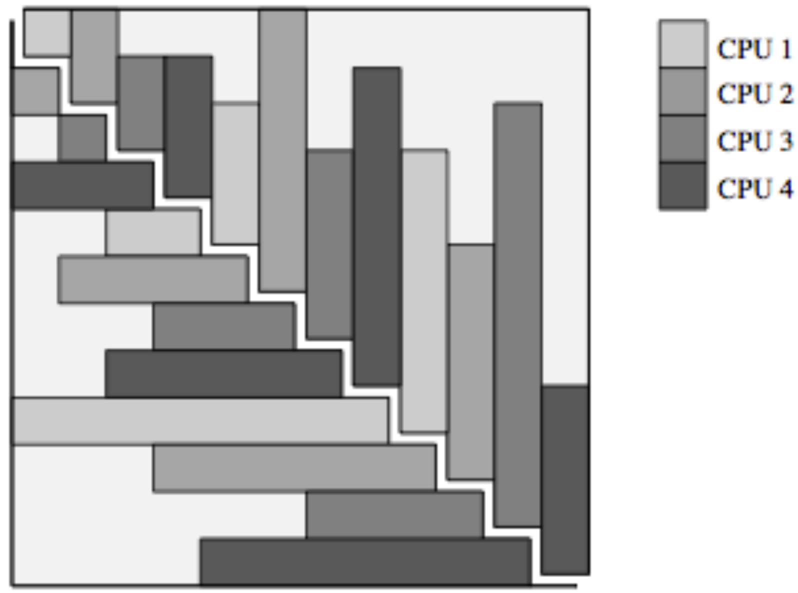
A

Data Distribution for Parallel Solution

Example Implementations
Skyline Matrix



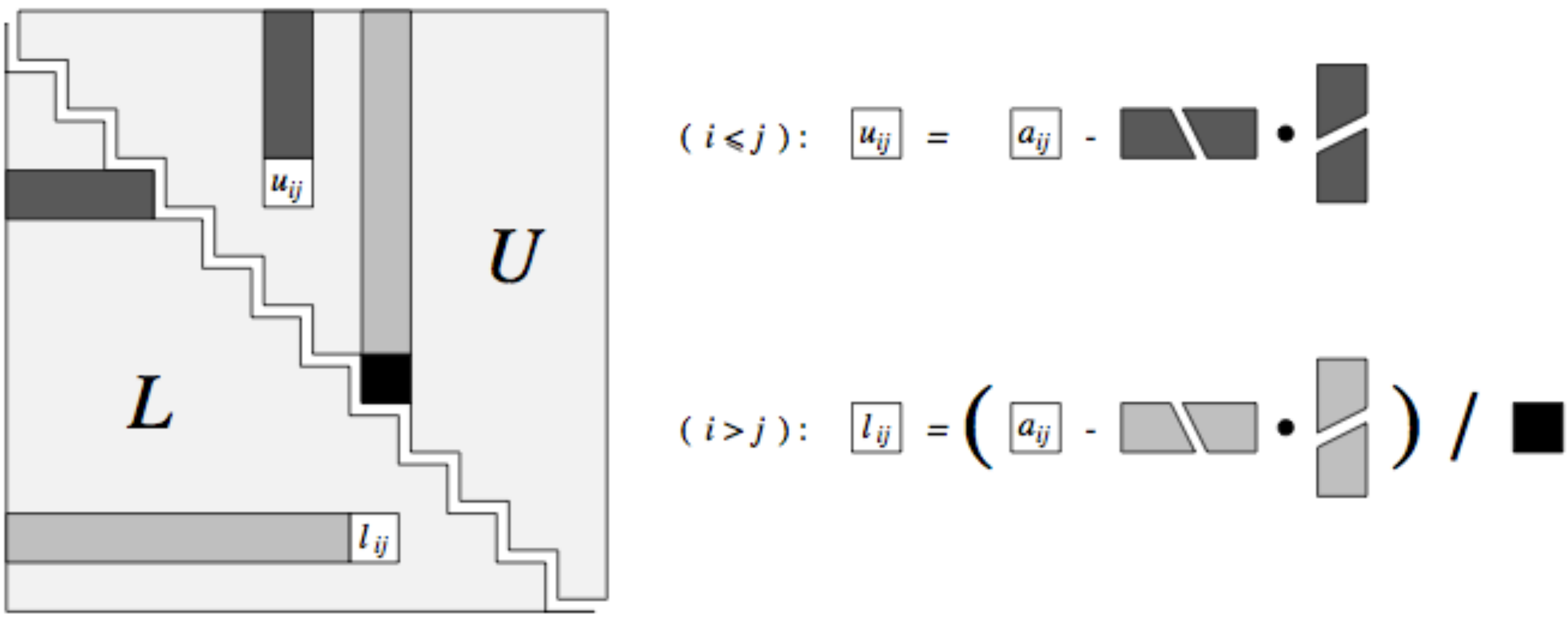
A



Data Distribution for Parallel Solution

Example Implementations
Skyline Matrix

Need for Data Replication



asyncCopy(…) and copy(…)

Experimental Setup

Platform

Server with 16 nodes, each featuring two Quad-Core AMD Opteron processors

Compiler

X10c++ backend

More mature and better optimizations

Runtime

pgas_sockets

Supports multiple places

Processors

pbs_resources

To control number of cores, and processors used

Experimental Methodology

Timing

Unix time and system wall time

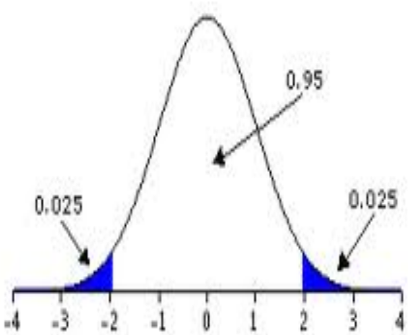
report real time

Repetitions

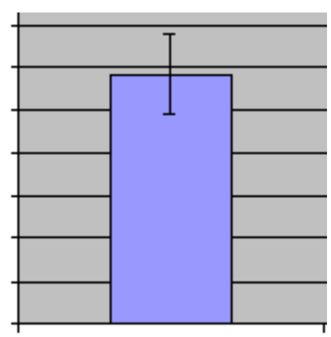
150

account for variances

Confidence Interval



95%

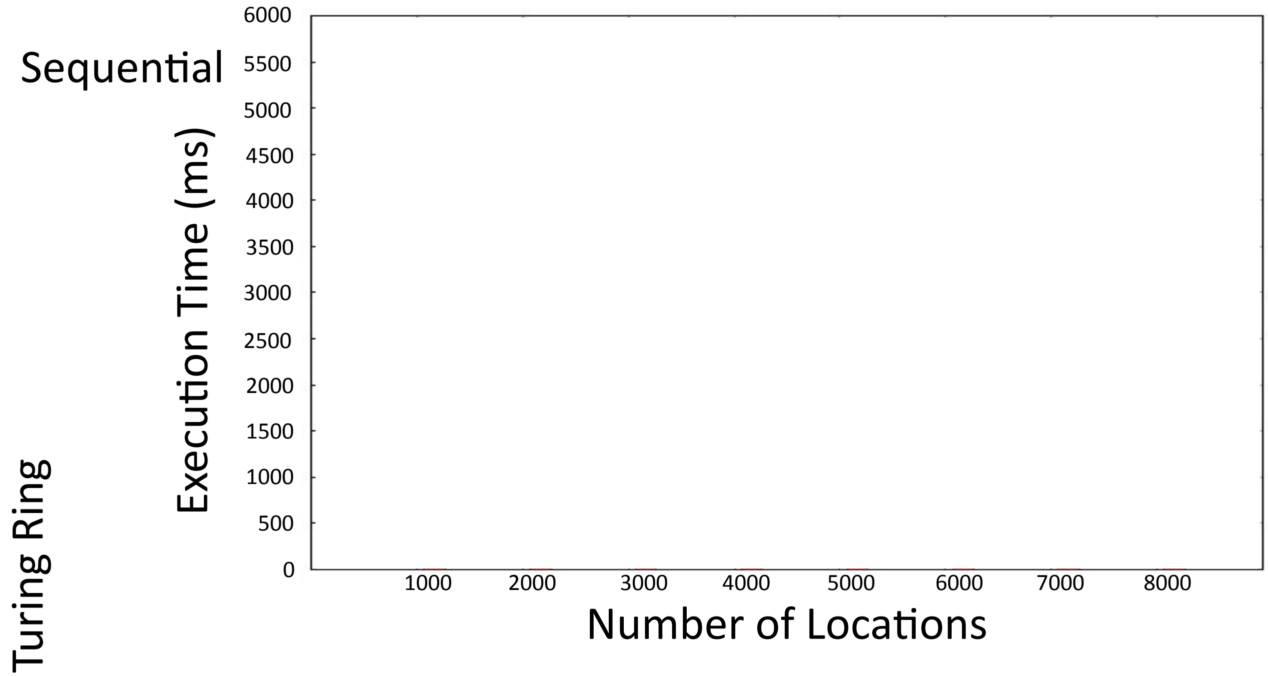


range of values that is likely to include differences from variable factors

Performance Reports

Turing Ring

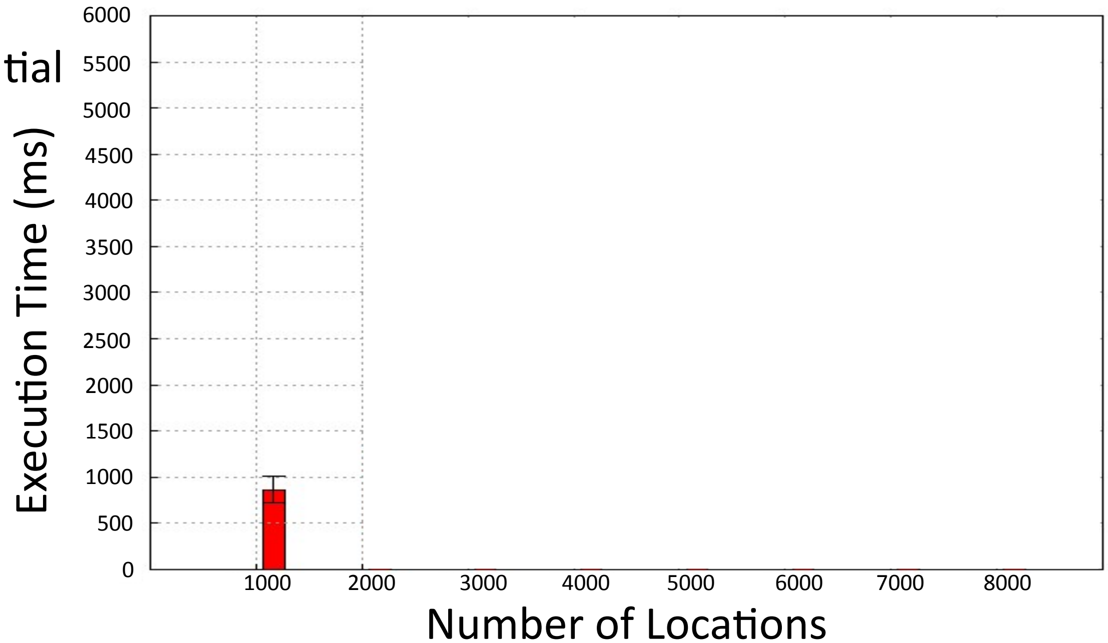
Performance Reports



Performance Reports

Sequential

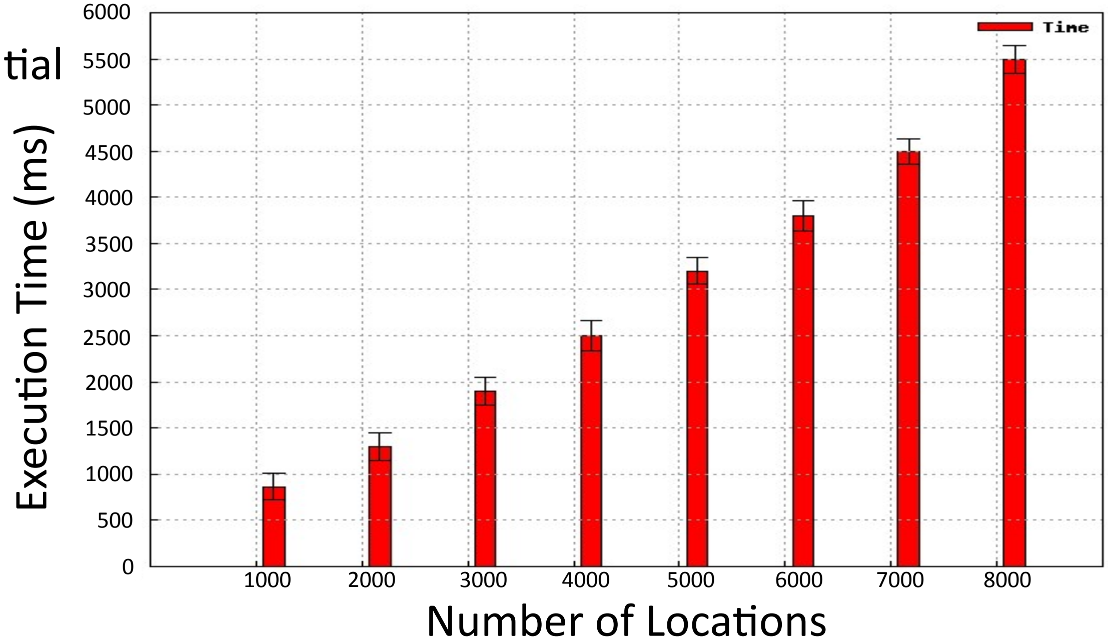
Turing Ring



Performance Reports

Sequential

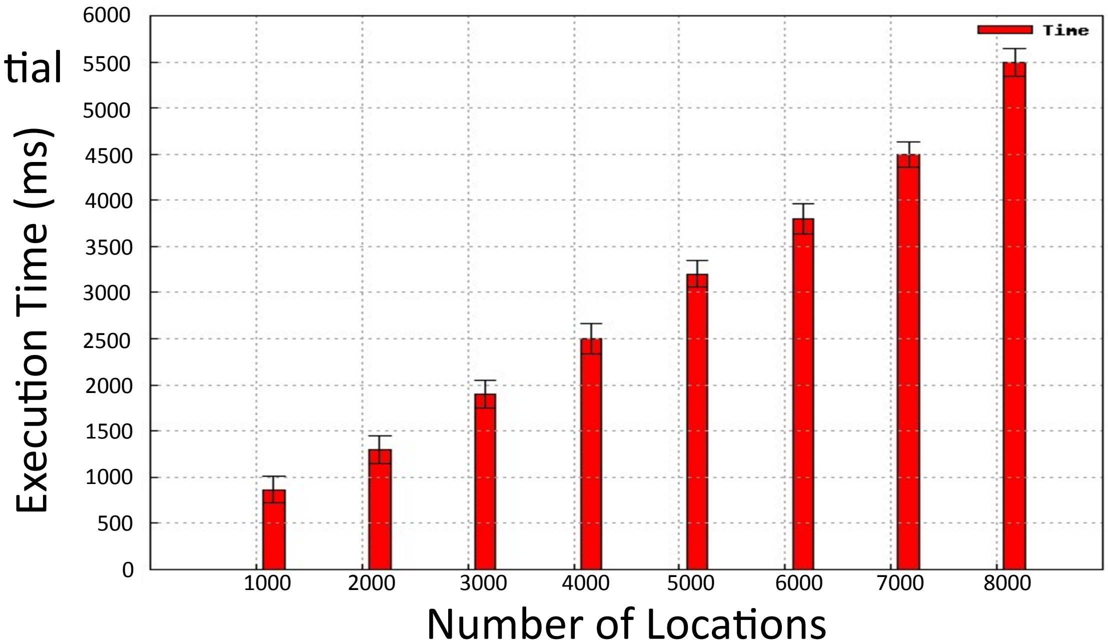
Turing Ring



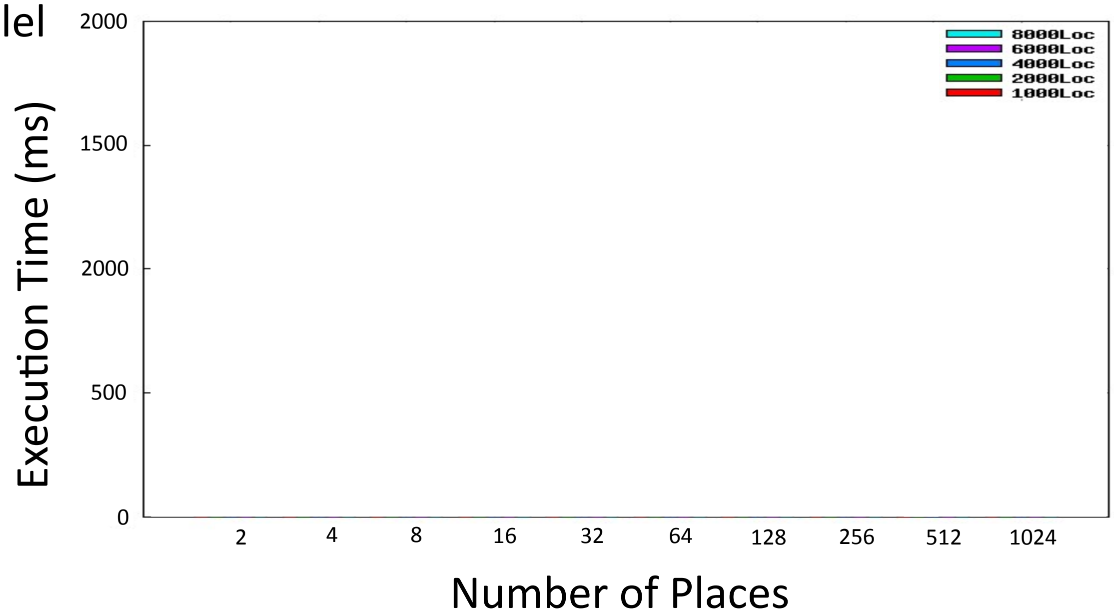
Performance Reports

Sequential

Turing Ring



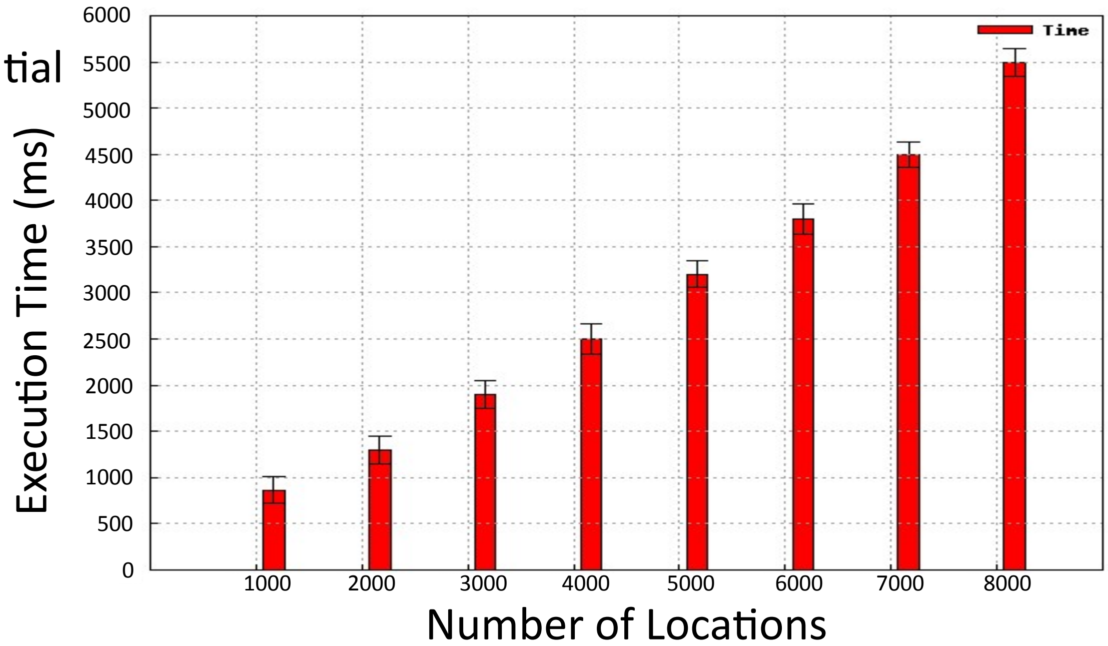
Parallel



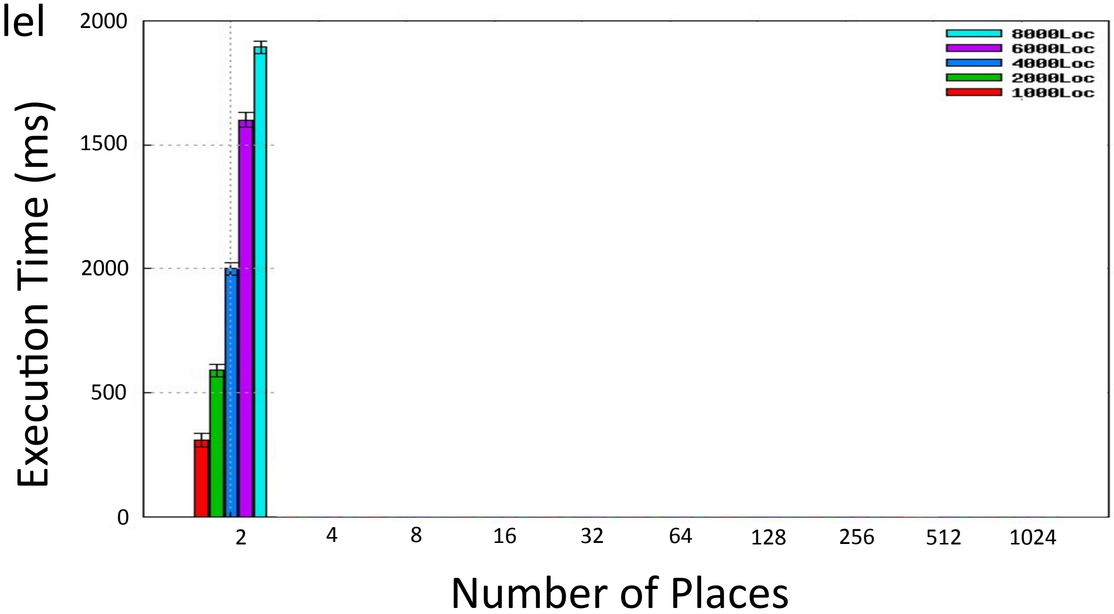
Performance Reports

Turing Ring

Sequential



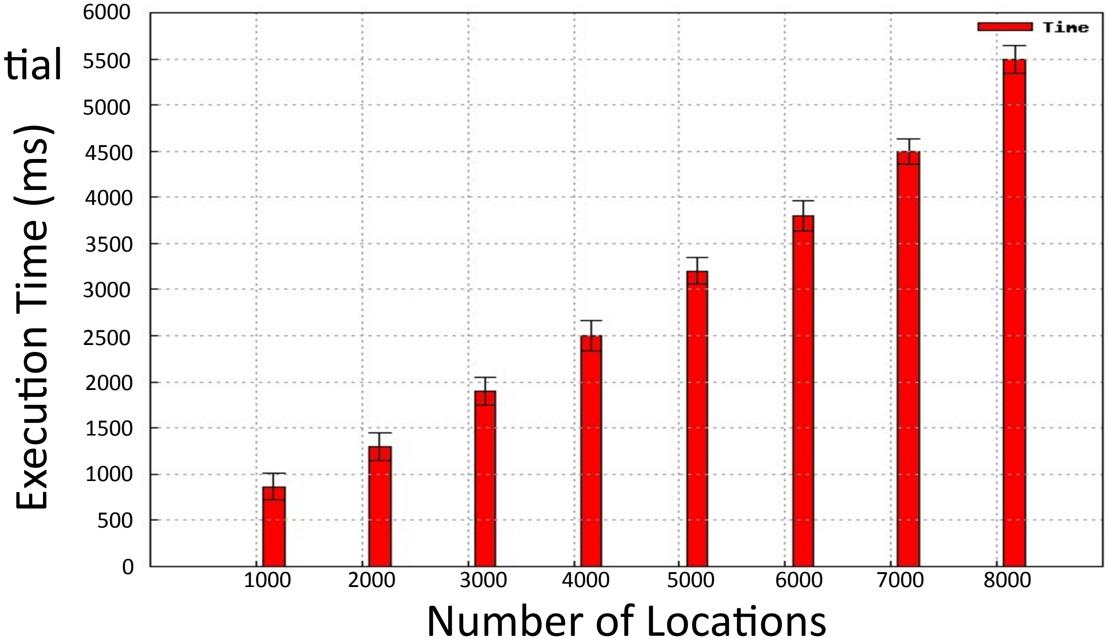
Parallel



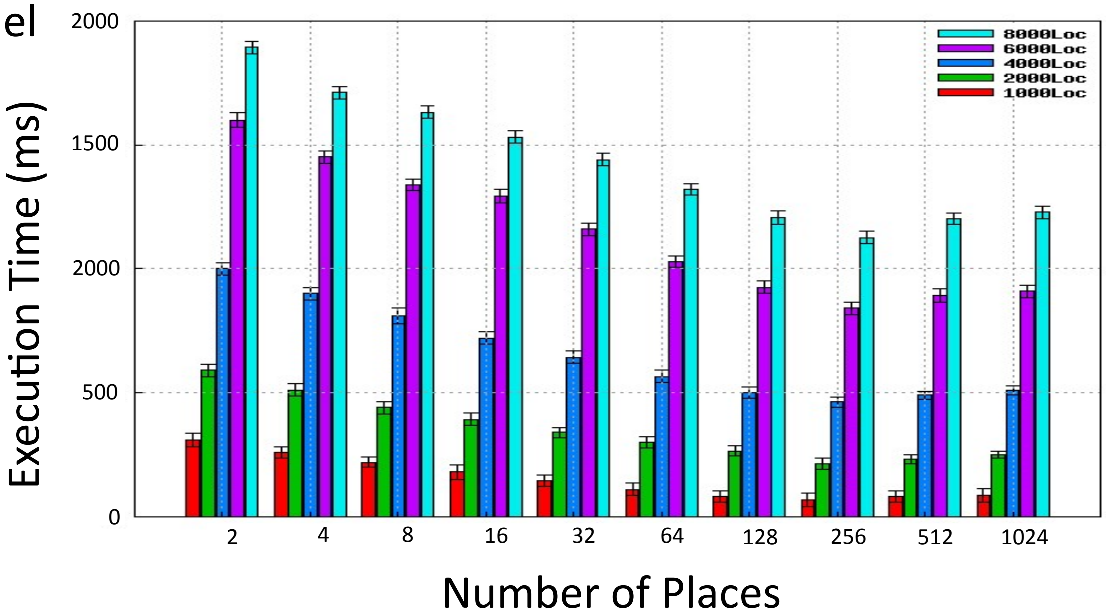
Performance Reports

Sequential

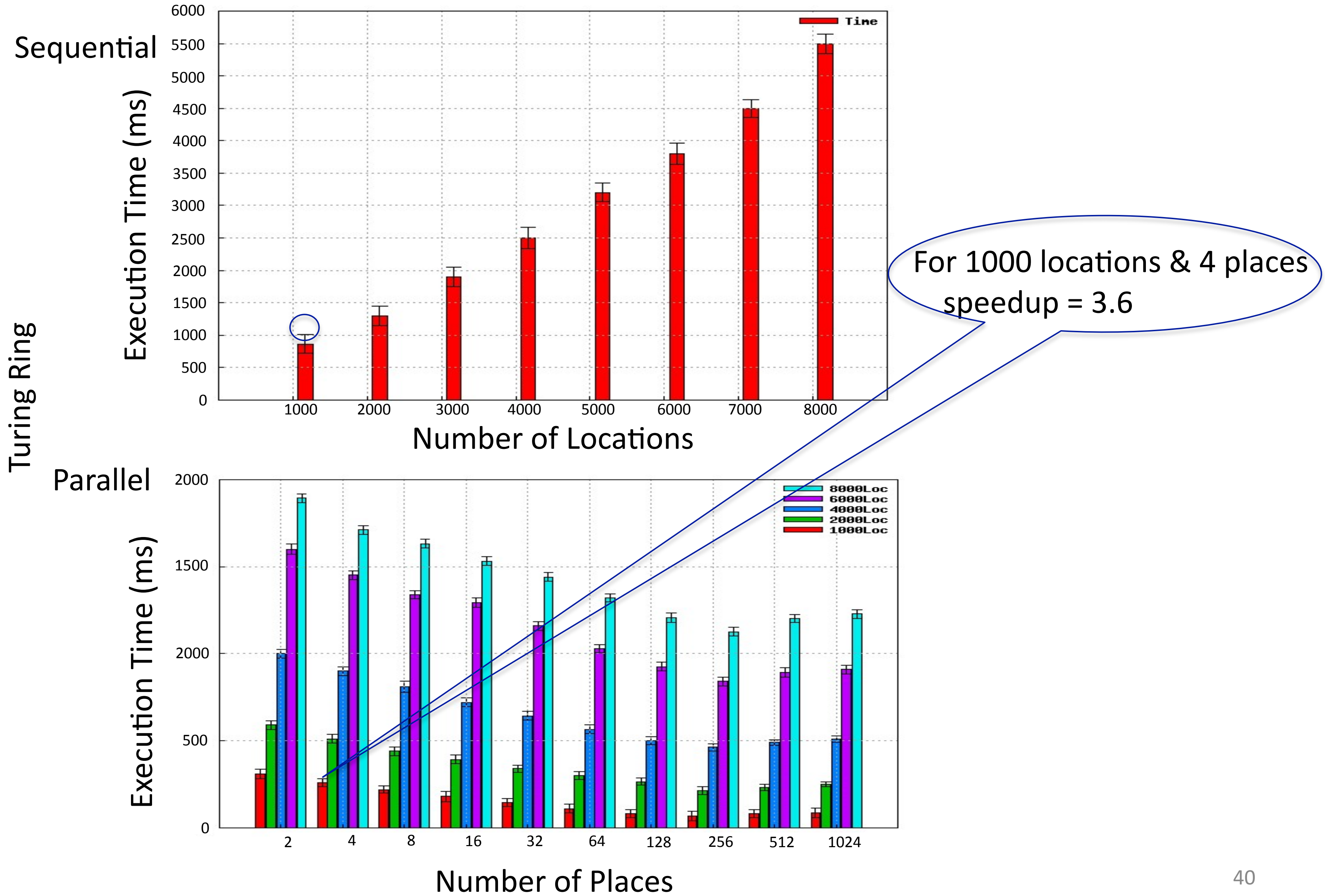
Turing Ring



Parallel



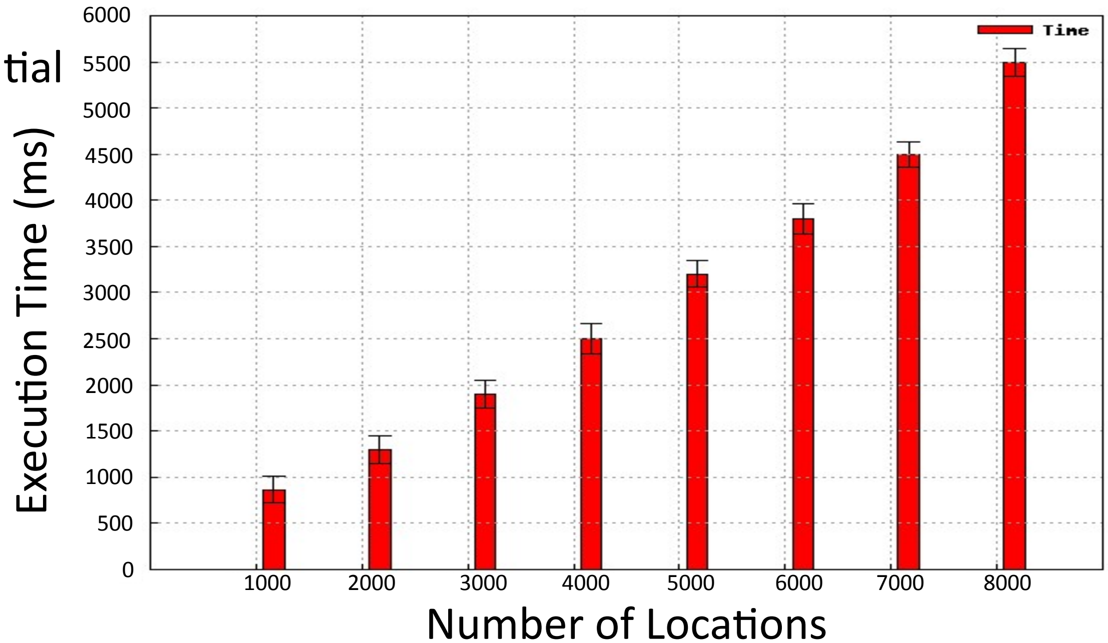
Performance Reports



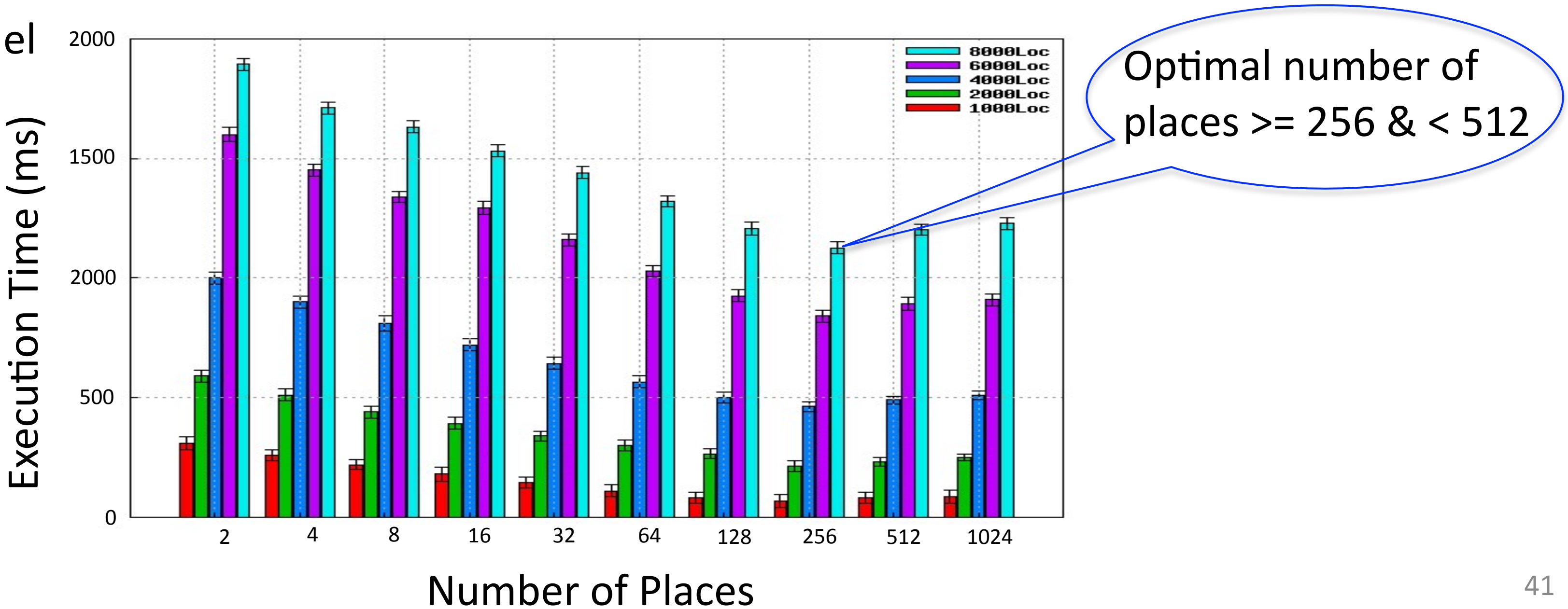
Performance Reports

Turing Ring

Sequential



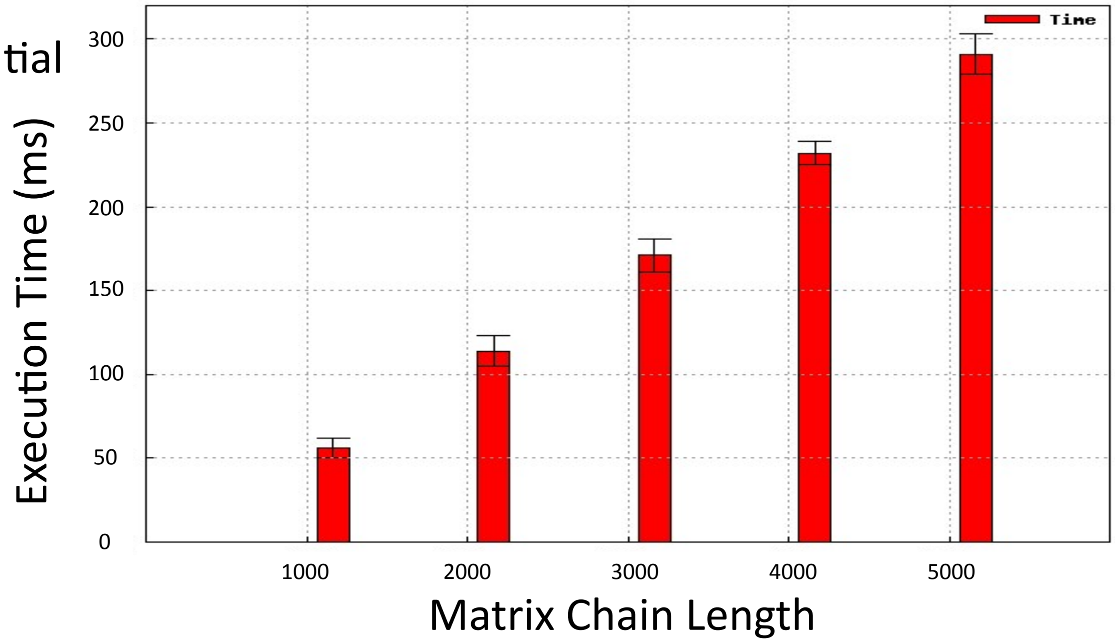
Parallel



Performance Reports

Sequential

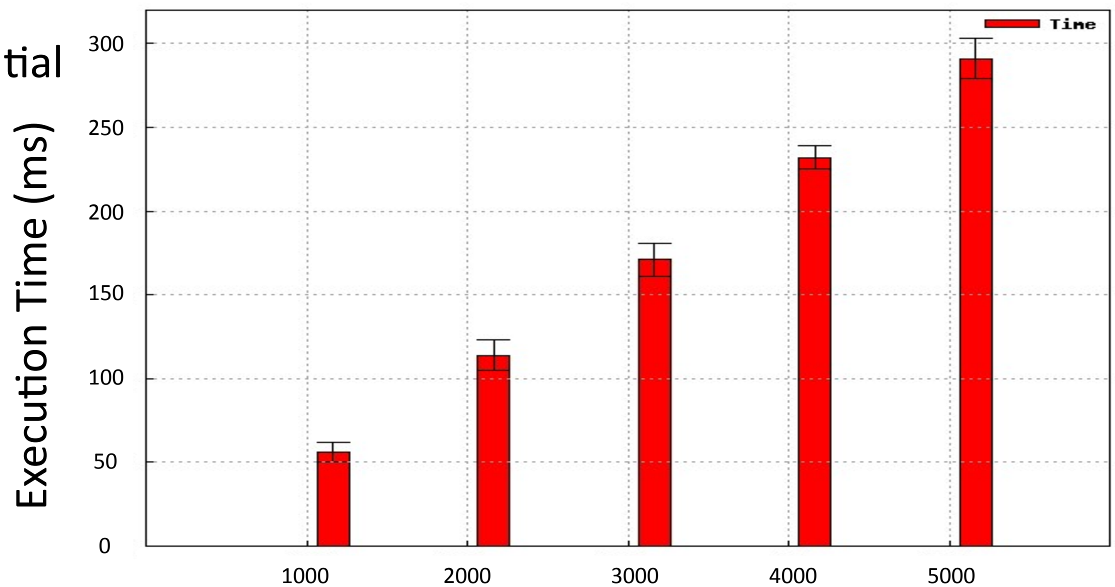
Skyline Matrix



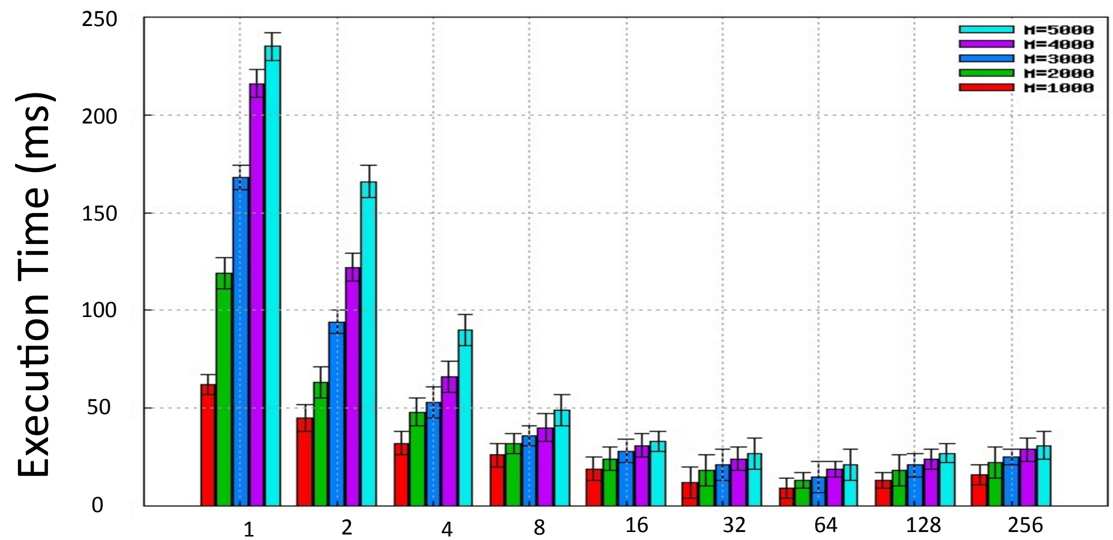
Performance Reports

Sequential

Skyline Matrix



Matrix Chain Length

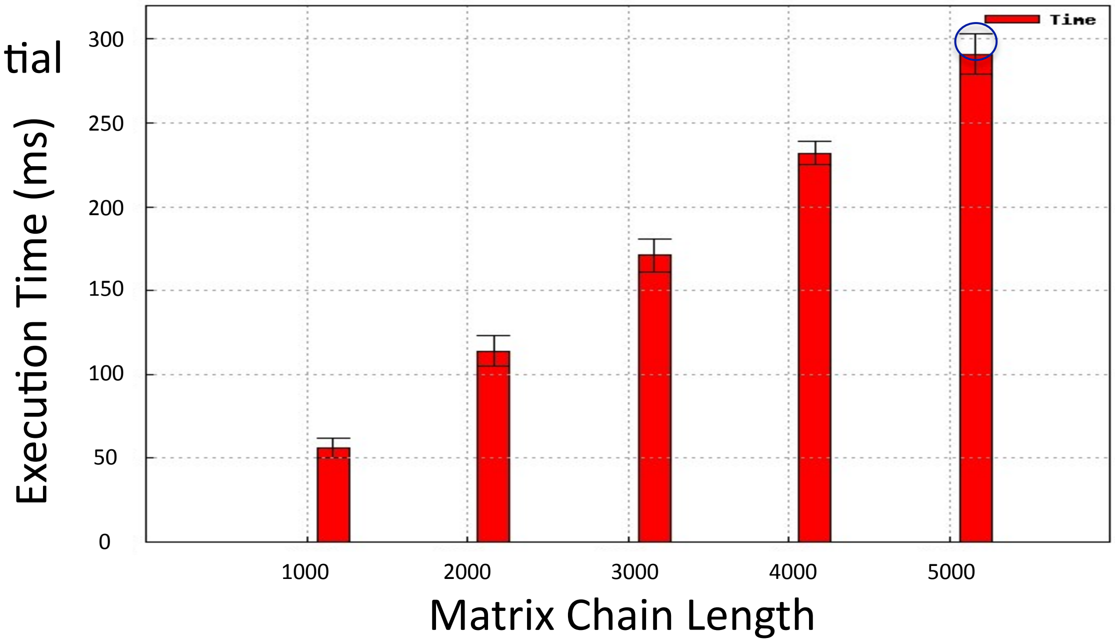


Number of Places

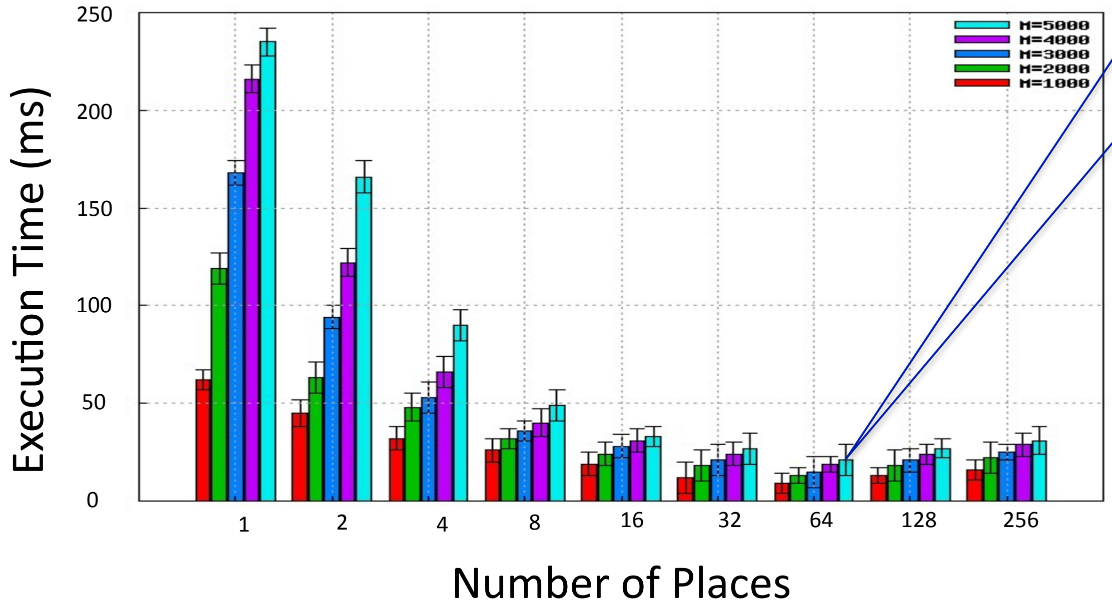
Performance Reports

Sequential

Skyline Matrix



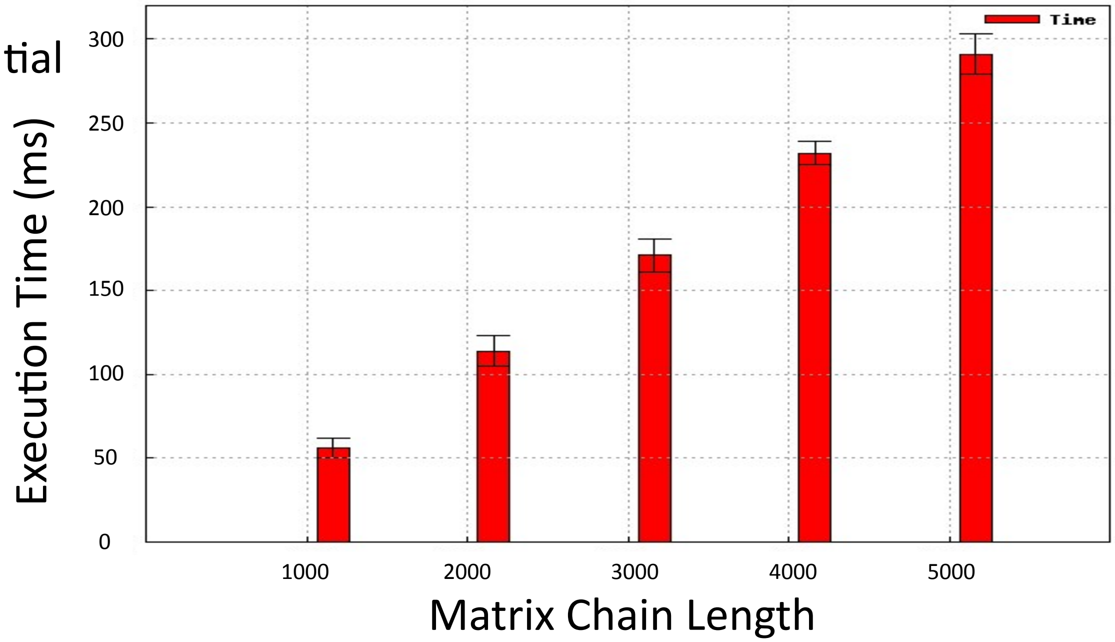
For chain length 5000 & 64 places, speedup = 11.2

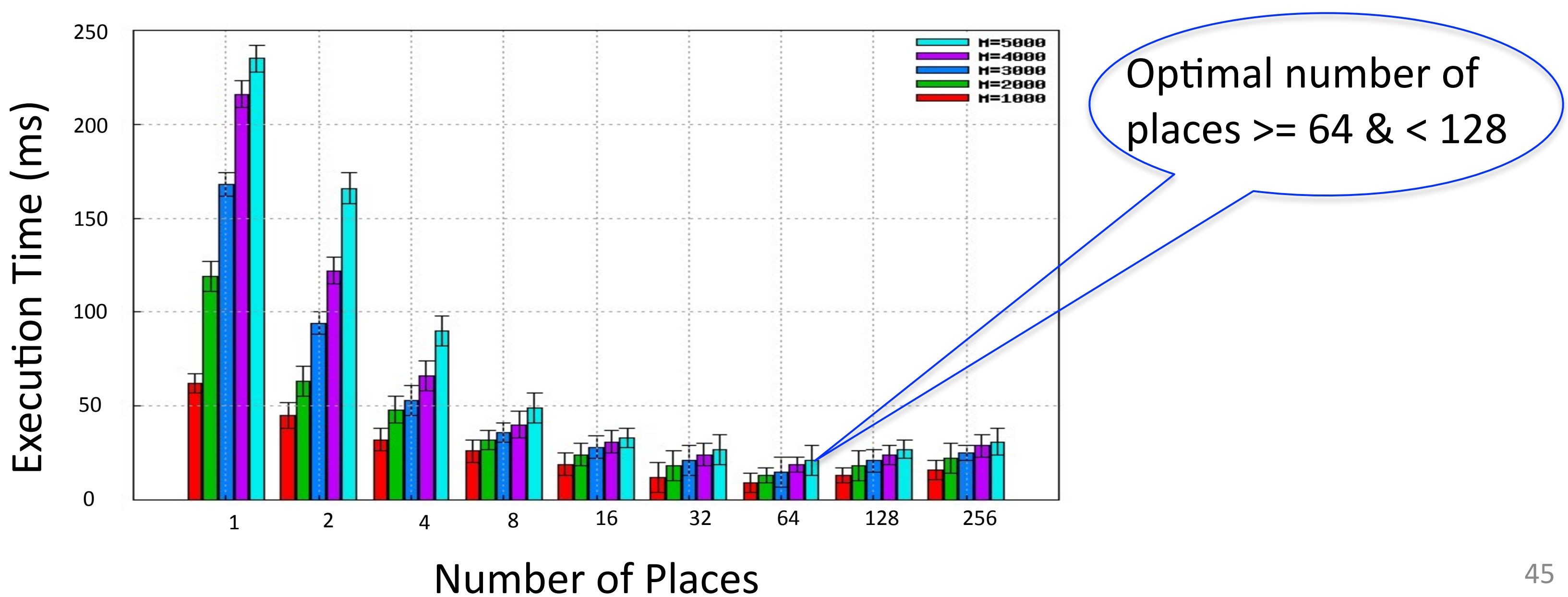


Performance Reports

Skyline Matrix

Sequential





Insights

Expressiveness:

Data representation & distribution

Task and data parallelism

Aggregate operations

Language Deficiencies:

Fine-grained async -- poor performance

Concurrent Input/Output -- useful

Error Messages -- cryptic

Comparison with Other Languages

X10 and UPC:

- both exploit data layout in memory (PGAS)
- X10 richer through async activities (APGAS)
- expressive
  - data and task parallelism
  - stripped allocation of arrays
  - collective operations
- remote shared memory accesses
- compiler technology

Comparison with Other Languages

X10 and OpenMP:

- OpenMP restricted to shared memory systems
- lacks efficient data locality support
- scalability issue

Comparison with Other Languages

X10 and Orca:

Orca uses shared objects for communication
shared objects encapsulated in abstract data types

X10 relies on APGAS model and uses messages
only immutable values are shared

Summary

As an emerging programming model, performance analyses necessary

Measured

- programmability of X10 language
- performance of X10 code

Better than UPC, OpenMP because of APGAS memory model

Runtime performance improvements needed

Additional language constructs useful

Questions?