BERKELEY LAB
Bringing Science Solutions to the World

UCMERCED
UNIVERSITY OF CALIFORNIA

U.S. DEPARTMENT OF ENERGY

ArrayUDF: User-Defined Scientific Data Analysis on Arrays

Bin Dong[1], Kesheng Wu[1], Surendra Byna[1], Jialin Liu[1]

Weijie Zhao[2], Florin Rusu[2]

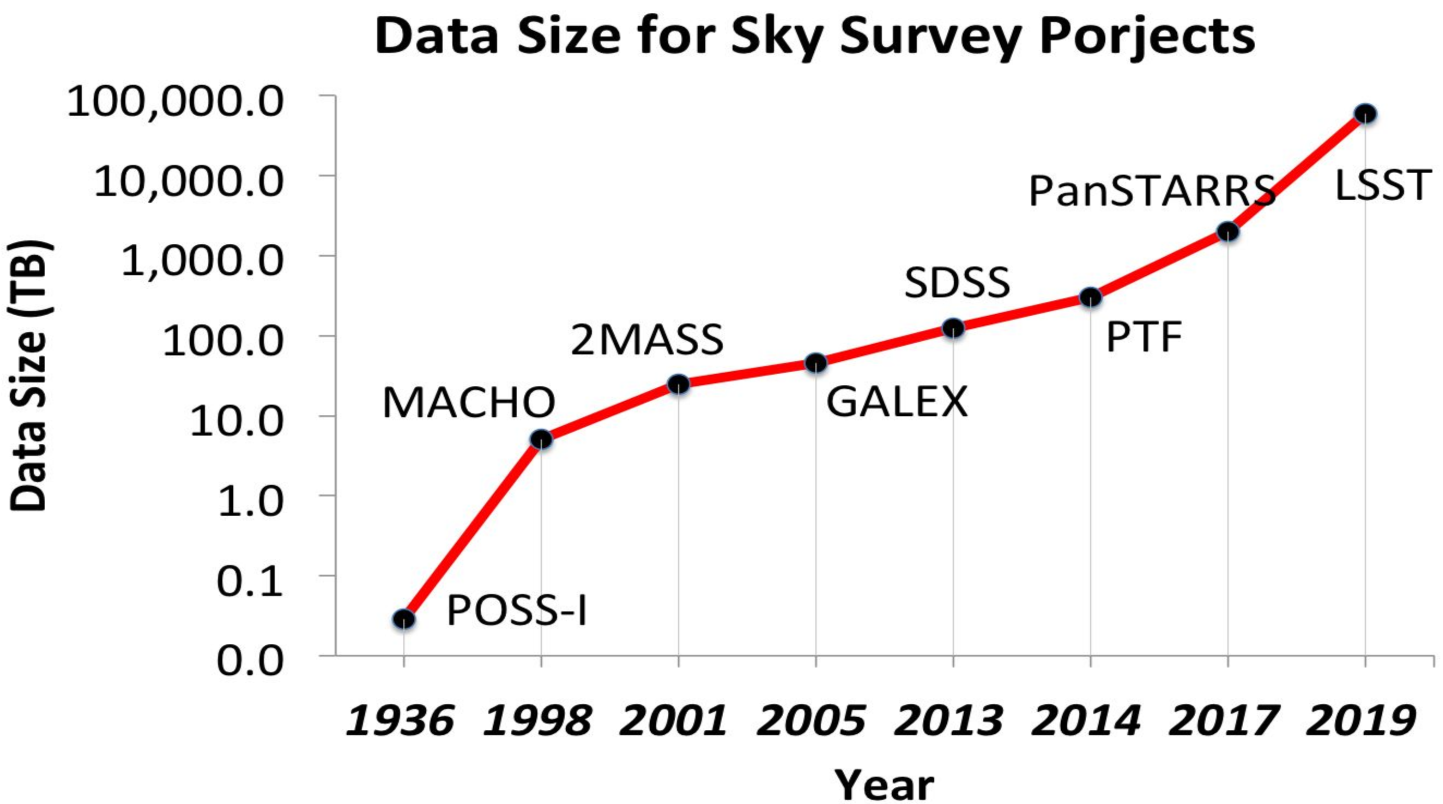
[1]LBNL, Berkeley, CA

[2]UC Merced, Merced, CA

HPDC 2017, Washington D.C. June 28, 2017.

Scientific Activities Evolved into Big Data Analysis

Example: scientific projects for dark matter/energy, supernovae, etc.



Data source:
Rick White,
J. Hart,
R. Cutri,
Ian Foster,
C. J. Grillmair,
etc.

Q1: How many data analysis operations that have been and will be developed ?

Q1: How many data analysis operations that have been and will be developed ?

➡ Large population

Implication from popular data analysis languages

	Total amount	Avg Growth
Python modules	110,486	69/day
R packages	10,877	8/day

*Data from http://www.modulecounts.com/ on June 21 2017

Q1: How many data analysis operations that have been and will be developed ?

→ Large population

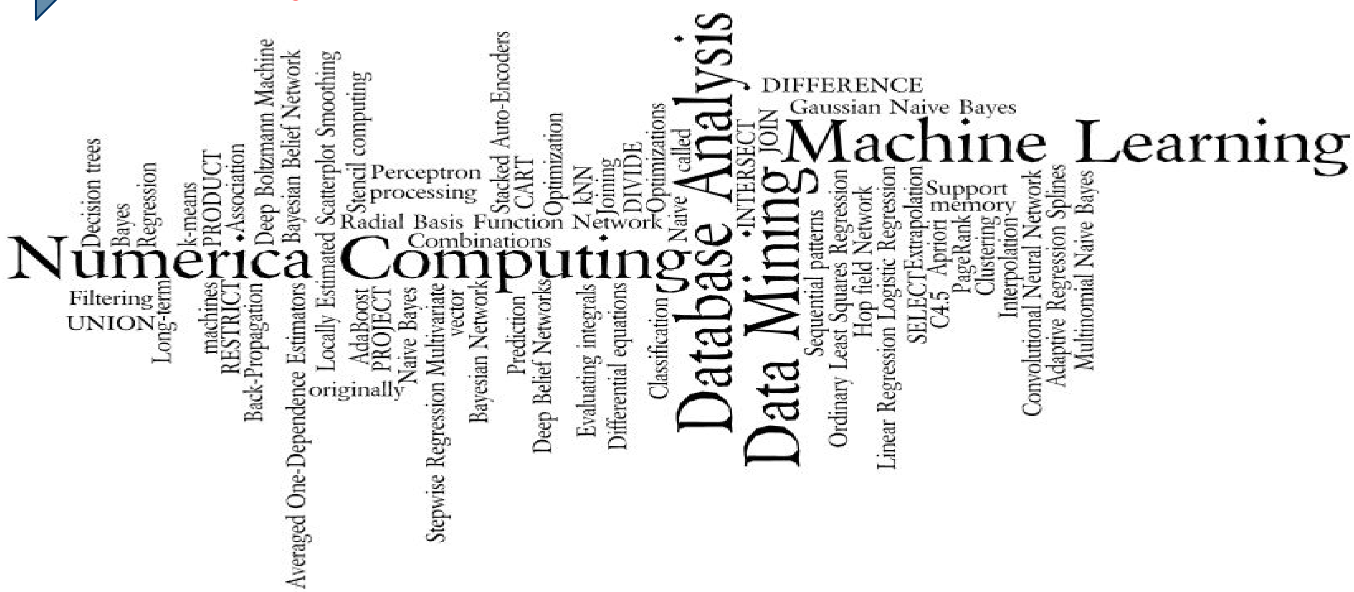
Q2: What are the functions of these data analysis operations ?

Q1: How many data analysis operations that can be used to extract scientific meaning ?

→ Large population

Q2: What are the functions of these data analysis operations ?

→ Variety



Two Methods to Embrace Large Population & High variety in Data Analysis Operations

Customized Solutions

For each operation *P* **Do**

Develop *P*'s :

- Data management → *Redundant* ✗
- Expression execution → *Diverse* ✔
- X components: → *Redundant* ✗
  parallel,
  communication
  cache,
  etc.

End For

Amateurishly tuned ✗

User-defined Functions (UDF)

Operation expression 1 → *Diverse* ✔

UDF API

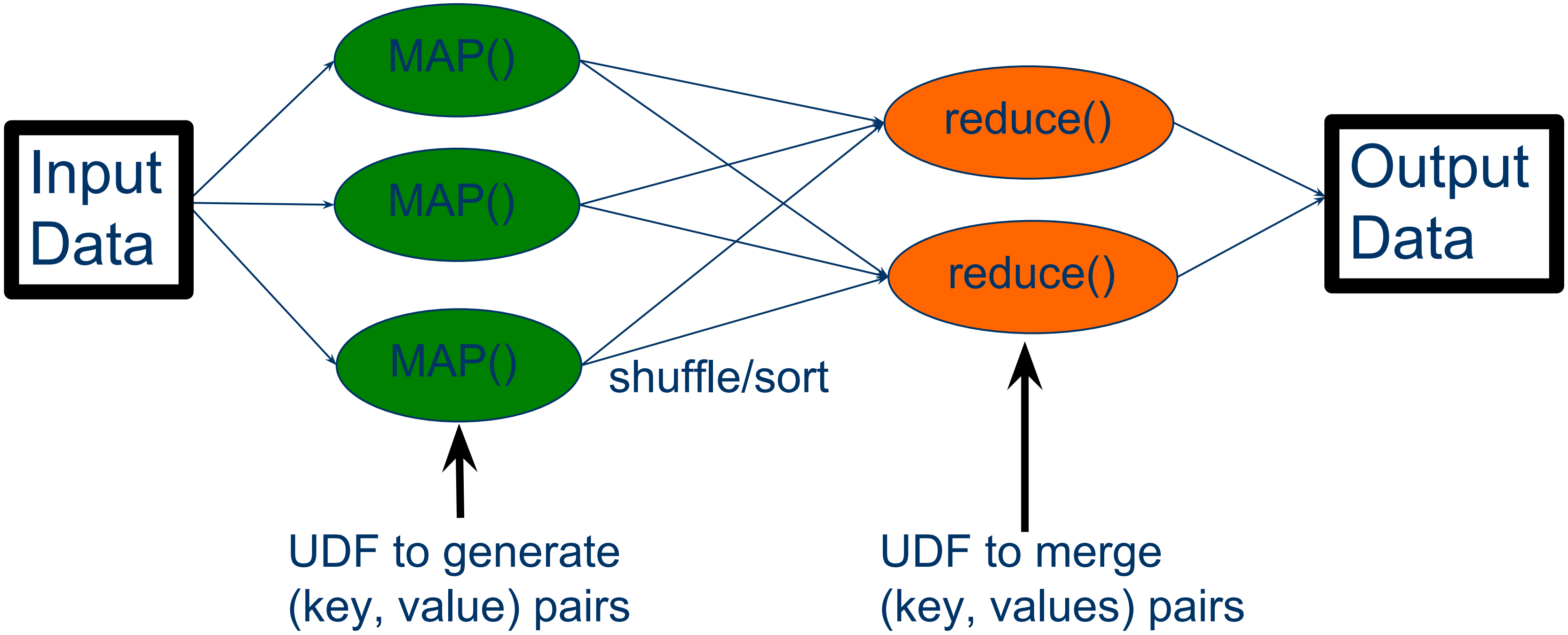
- Data management
- Generic exec. engine
- X components:
  parallel, comm.,
  cache, etc.

One single and shared ✔

Professionally tuned ✔

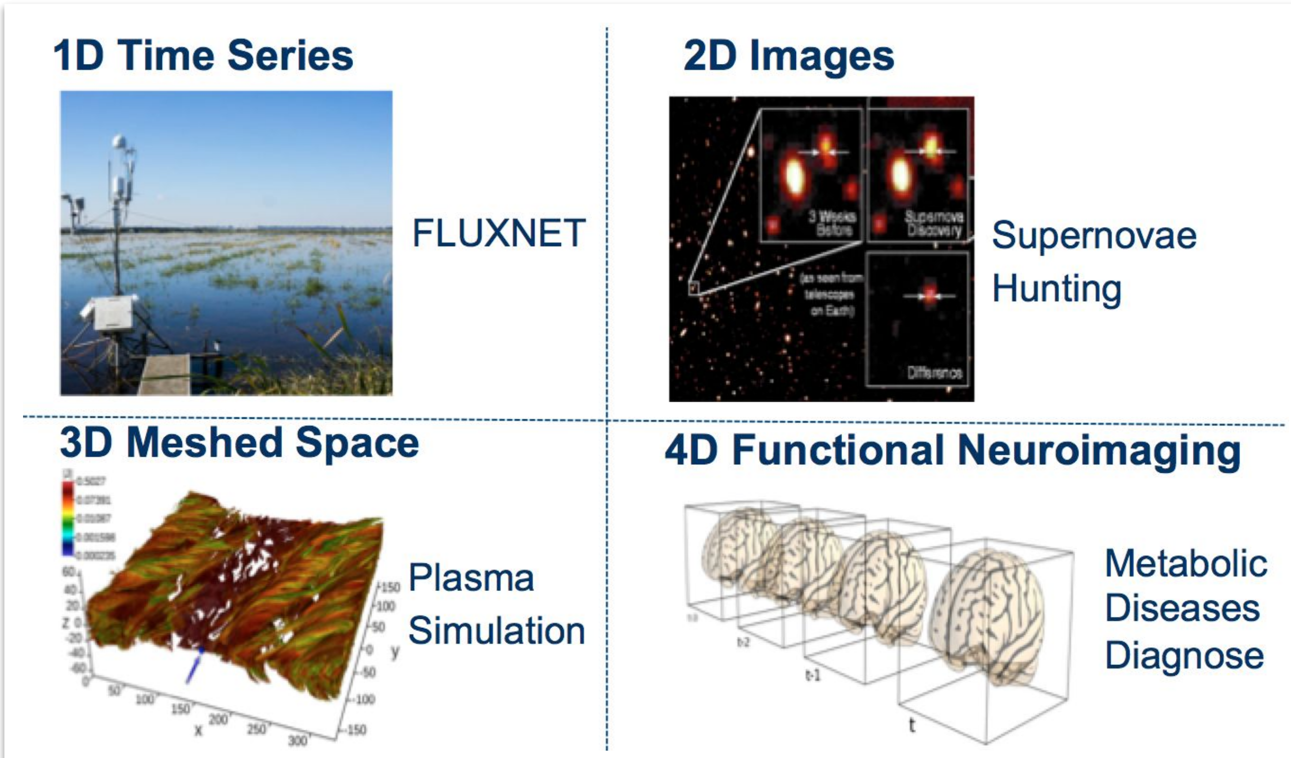
UDF is at Heart of Modern Big Data system

Examples: MapReduce in Apache Hadoop and Spark



MapReduce Doesn’t Fit Scientific Data Analysis

Reason 1: Most Scientific Data are Multi-dimensional Arrays



Converting array to (key, value) is expensive

MapReduce Doesn’t Fit Scientific Data Analysis

Reason 1: Most Scientific Data are Multi-dimensional Arrays

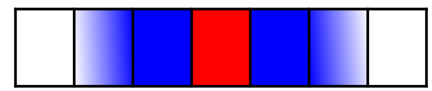
Reason 2: Most Scientific Data Analysis Operations Own Structure Locality Property

Structure Locality:

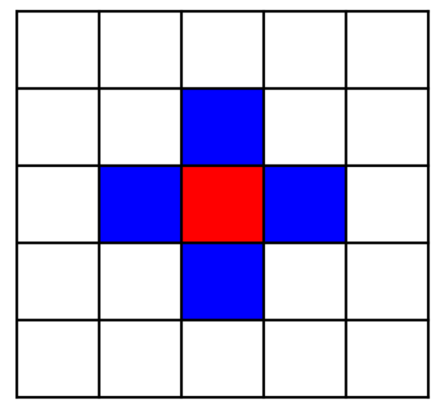
The analysis operation on a single cell accesses its neighborhood cells

- Map deals with a single element at a time
- Reduce requires to duplicate an each cell for all neighborhood cells
- Reduce only happens after expensive shuffle

Moving Average



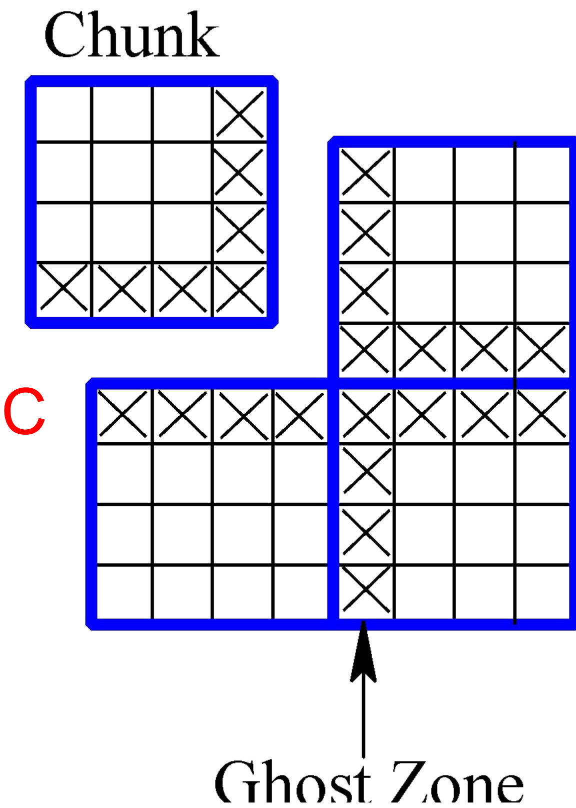
2D Poisson Equation Solver (Discrete)



Pictures Credit: Kyle Hemes, Peter Nugent, Suren Byna, etc.

ArrayUDF: User-Defined Scientific Data Analysis on Arrays

- Stencil based User-defined Function
  - Structural locality aware array operations
- Native Multidimensional Array Data Model
  - In-situ data processing in scientific data formats, e.g., HDF5
- Optimal Chunking and Ghost Zone Method
  - Efficiently parallel array processing on HPC system



Stencil Definition

- Stencil S Definition for *d*-dimensional array

$$S = \{c_{i_1+\delta 1,\bullet\bullet\bullet,i_d+\delta d}, \mid \delta \in [L,R], L \in [-i,0], R \in [0, N-i]\}$$

➡ Materialized structure locality

➡ Flexible neighborhood expression

- Centre of Stencil: $s_{0,\bullet\bullet\bullet,0} = c_{i_1,\bullet\bullet\bullet,i_d}$
- Element at offsets $\delta_1, \bullet\bullet\bullet, \delta_d$:

$$s_{\delta_1,\bullet\bullet\bullet,\delta_d} = c_{i_1+\delta_1,\bullet\bullet\bullet,i_d+\delta_d}$$

2D Example:

	$s_{-1,-1}$	$s_{-1,0}$	$s_{-1,1}$	
	$s_{0,-1}$	$\boxed{s_{0,0}}$	$s_{0,1}$	
	$s_{1,-1}$	$s_{1,0}$	$s_{1,1}$	

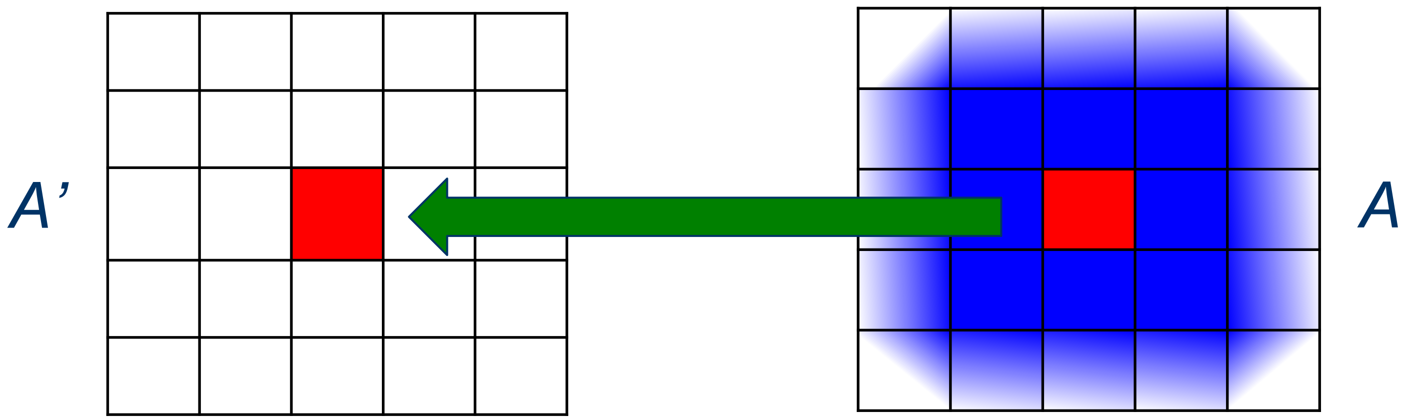
Stencil based Computing Model

$$c'_{i_1,\bullet\bullet\bullet,i_d} = f\left(\{s_{\delta_1,\bullet\bullet\bullet,\delta_d} \mid \delta \in [L,R]\}\right)$$

The cell in array *A'* *at coordinate* $i_1 \bullet\bullet\bullet i_d$

The stencil *set* in array *A* *at offset:* $\delta_1 \bullet\bullet\bullet \delta_d$

- A = A' when in-situ updates
- Dim(A) = Dim(A') in most cases



Stencil based Computing Model vs Others

	Input	Output	UDF
SQL	Tuple *t*	Tuple *t'*	$t'=\mathbf{f}(t)$
SciDB	Cell *c*	Cell *c'*	$c'=\mathbf{f}(c)$
MapReduce	KeyValue *kv*	KeyValue *kv''*	$kv'=Map(kv)$ $kv''=Reduce(kv'_1, kv'_2, \ldots)$
ArrayUDF	Stencil *s*	Cell *c'*	$c'=\mathbf{f}(s)$

vs. MapReduce:

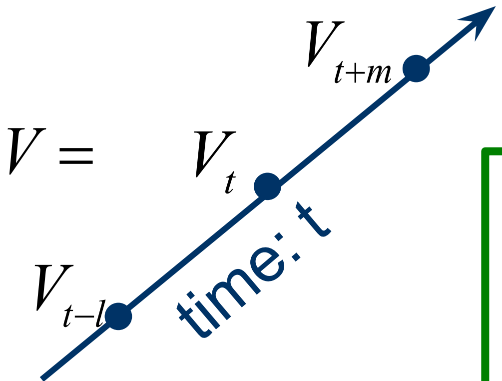
ArrayUDF generalizes its two steps as a single step on array

vs. SciDB:

ArrayUDF supports structure-locality based computing on array

Examples of ArrayUDF

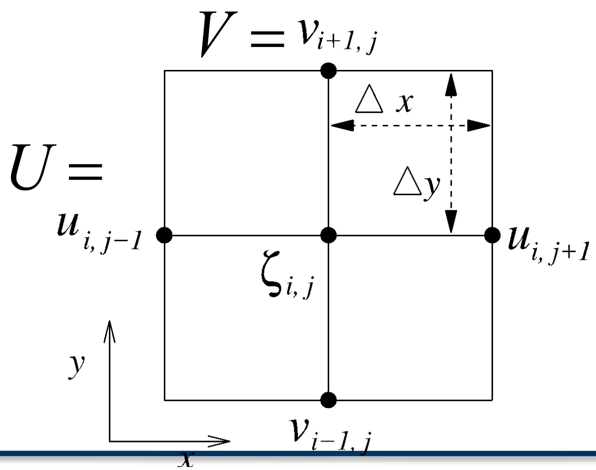
Example 1: Moving average



$$V'_t = \frac{w_t V_{t-k} + \ldots + w_t V_t + \ldots + w_{t+m} V_{t+m}}{k+m+1}$$

```
UDF_MV(Stencil s){
    return (s(-k)*w_t-k+ … s(0)*w_t+ … s(m)*w_t+m)/(k+m-1)
}
V.Apply(UDF_MV)
```

Example 2: Vorticity computation



$$\xi_{i.j} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta x} + \frac{v_{i+1,j} - v_{i-1,j}}{2\Delta y}$$

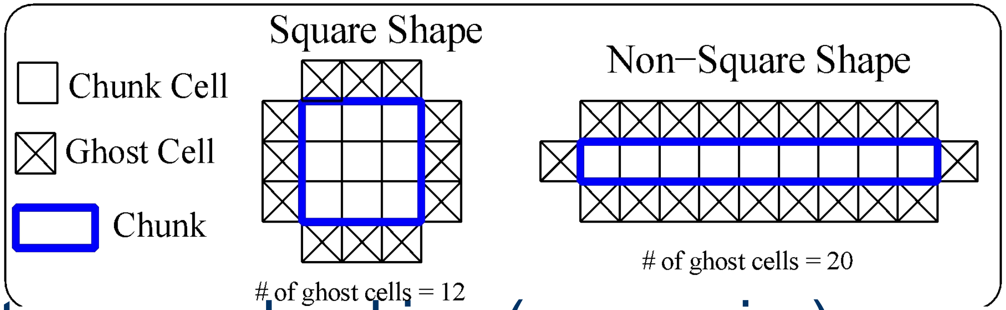
```
UDF_VC_U(Stencil u){
    return u(0,1)- u(0, -1)
}
U.Apply(UDF_VC_1)
```

```
UDF_VC_V(Stencil v){
    return v(1,0)- u(-1, 0)
}
V.Apply(UDF_VC_V)
```

ArrayUDF's C++ implementation

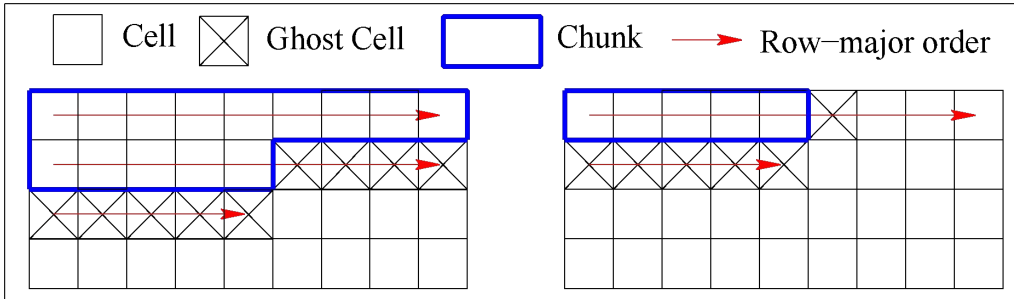
UDF Support System: Chunking

- Chunking enables parallel and out-of-cores processing
  - general chunking (layout unknown)
    - minimize ghost cells



  - layout-aware chunking (row-major)
    - maximize contiguous disk read

See theoretical proof in paper



UDF Support System: Ghost Zone handling

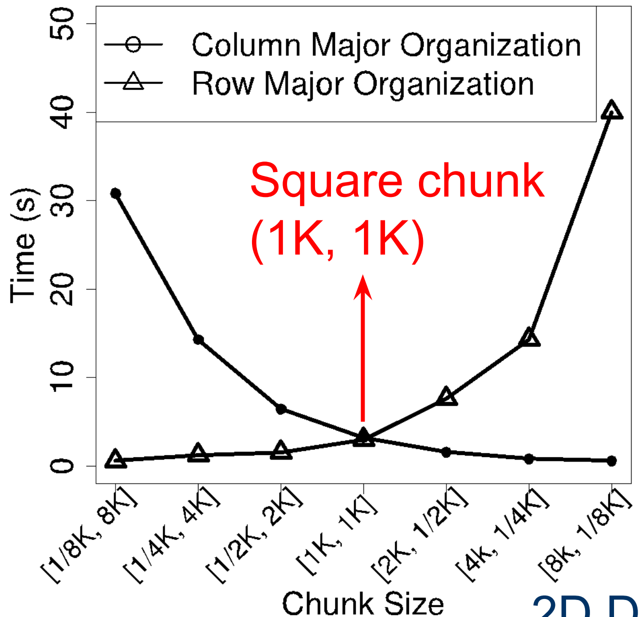
- ArrayUDF processes chunks in parallel and/or in out-of-core manner
- Ghost zone avoids communications between chunks
- Ghost zone size might be unknown in advance
  - UDF source code might be unavailable
- Trail-run is used to find ghost zone size:
  - Run UDF on a sample Stencil instance, that collects the offsets applied within UDF

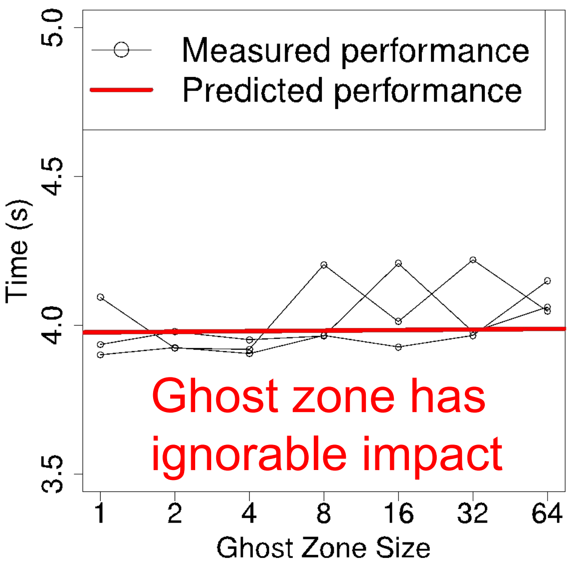
Evaluations

- Hardware:
  - Edison, a Cray XC30 supercomputer at NERSC
  - 5576 computing nodes, 24 cores/node, 64GB DDR3 Memory
- Software
  - ArrayUDF 0.0.1
  - Spark 1.5.0
  - SciDB 16.9
  - RasDaMan 9.5 (sequential version)
  - EXTASCID, hand-optimized version
  - Hand-optimized C/C++ code
- Workloads
  - Two synthetic data sets (i.e., 2D and 3D array) for micro benchmarks
    - Chunking strategy, trail-run, etc.
  - Four real scientific data sets (i.e., S3D, MSI , VPIC , CoRTAD)
    - Overall performance tests /w generic UDF interface

Chunking Strategy Evaluation

- general chunking (for average cases)
  - minimize ghost cells # to reduce I/O cost
- layout-aware chunking (for layout special case)
  - maximize contiguous disk read
  - ignore the impact of ghost zone





2D Dataset (100000, 100000)

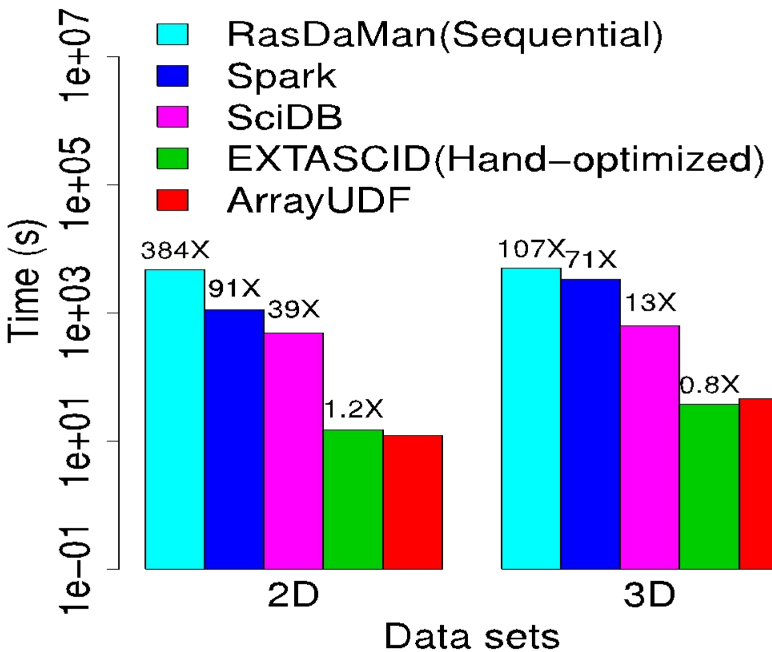
Comparison with peer systems with standard "window" operators

- "window" comes from SCiDB and RasDaMan
- "window" supports certain structure locality but lack the link to UDF function

Poisson equation solver /w
Stencil S of ArrayUDF

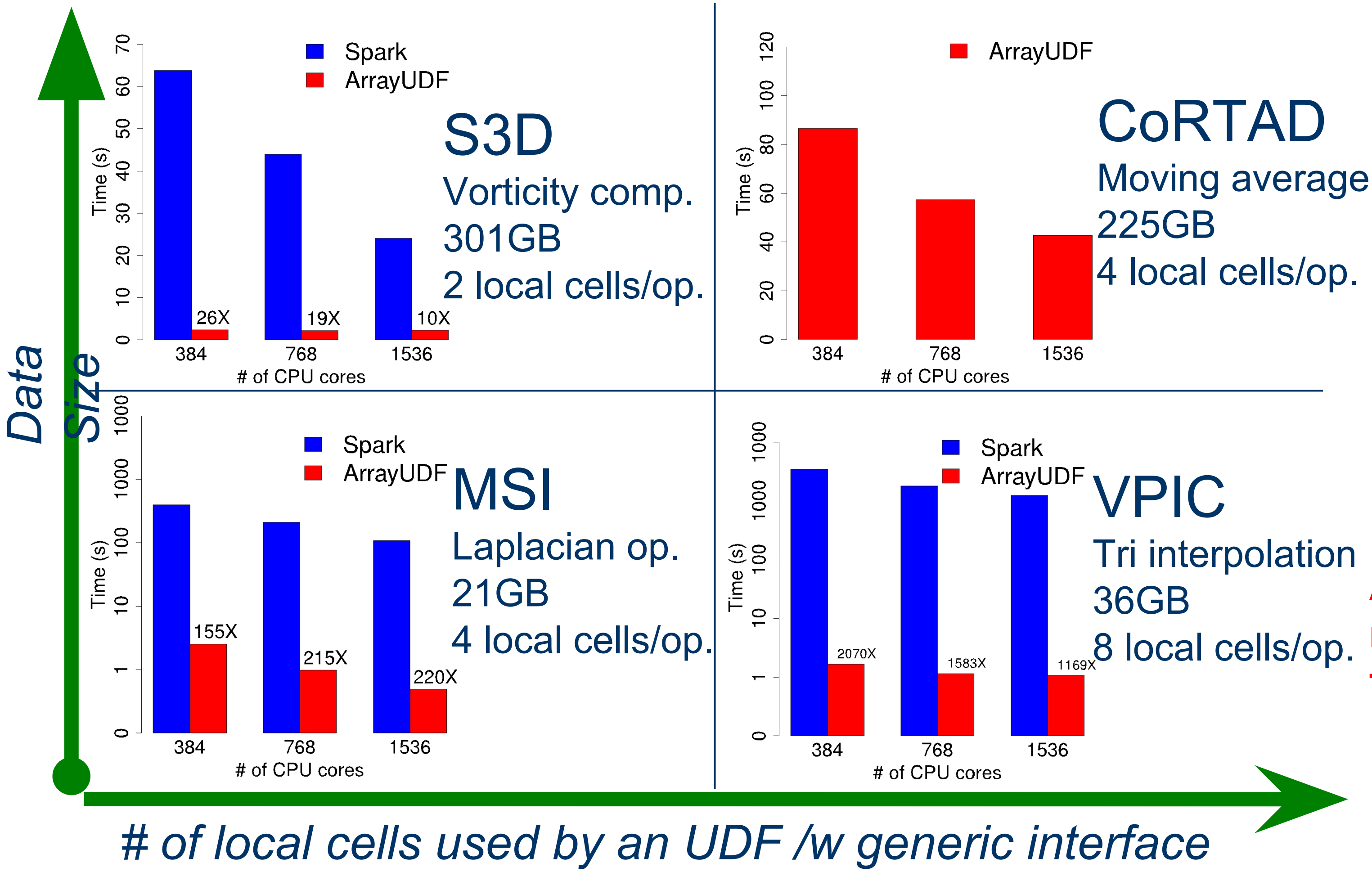
2D : 4S(0,0)−S(−1,0)−S(0,1)
−S(1,0)−S(−1,0)

3D : 6S(0, 0,0)−S(−1,0,0)−S(0, 1, 0)
−S(1,0, 0)−S(−1,0,0)−S(0,0,−1)
−S(0, 0,1)



- ArrayUDF has close performance to hand-optimized code
- ArrayUDF is at least 13X faster than peer systems

Comparison with Spark in supporting real applications operations



Spark's Out-Of-Memory:

- large data size

- more local cells

ArrayUDF is as much as *2070X* faster than Spark

Conclusions

- ArrayUDF: User-Defined Scientific Data Analysis on Arrays
  - Stencil based UDF for structural locality-aware operations
  - Native array model & In-situ array processing in HDF5, etc.
  - Optimal chunking and ghost zone methods for large array
- ArrayUDF provides close performance to hand-optimized code
- ArrayUDF is as much as *2070X* faster than Spark
- ArrayUDF source code: https://bitbucket.org/arrayudf/
- Future work
  - Python and other language interface
  - Communication optimizations

Acknowledgments

- Nicholas Chaimov from University of Oregon for suggestions to set up Spark on Editon at NERSC
- Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, support for the SDS project (Program manager: Dr. Lucy Nowell) under contract number DE-AC02-05CH11231

U.S. DEPARTMENT OF ENERGY | Office of Science

- National Energy Research Scientific Computing Center

NERSC

BERKELEY LAB

UCMERCED
UNIVERSITY OF CALIFORNIA

Thanks

Bin Dong
dbin@lbl.gov
http://crd.lbl.gov//dongbin

Trail-run overhead

- Detect ghost zone size automatically
  - Run the UDF on a single Stencil but the UDF might access more neighborhood cells

Data sets	The number of cells used by UDF						
	4	8	16	32	64	128	256
S1	0.37	0.38	0.46	0.48	0.54	0.59	0.80
S2	0.48	0.52	0.65	0.75	0.79	0.84	1.04

Unit: microsecond

≈ 1 ms when 256 cells are used in the UDF