Evaluating Next Generation PGAS Languages for Computational Chemistry

CHIUW: Chapel Implementers and Users Workshop (collocated with IPDPS'14)

Daniel Chavarría-Miranda
Joseph Manzano
Sriram Krishnamoorthy
Abhinav Vishnu

High Performance Computing
Pacific Northwest National Laboratory

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

Introduction/Motivation

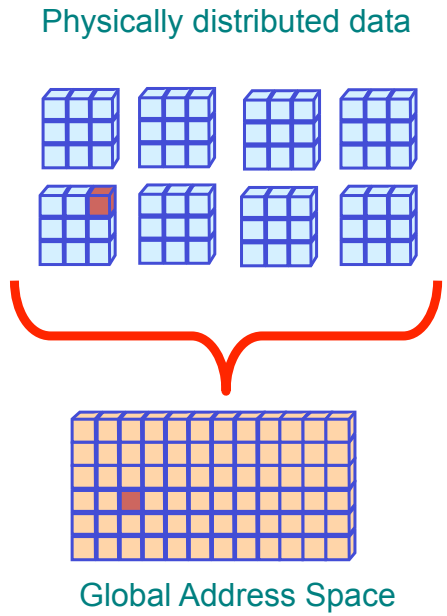
- Computational Chemistry applications have made extensive use of the PGAS paradigm
  - Scalable, high performance implementations including NWChem
  - In particular, the Global Arrays (GA) PGAS library
- Need for PGAS comes from:
  - Block-sparse data access patterns
  - Load imbalance induced by those patterns
- Contrast to physics applications:
  - Chemistry algorithms cannot easily exploit domain decomposition and spatially constrained interactions
    - Main mechanisms to enhance locality and reduce the scope of inter-process communication
  - Spatial & temporal locality on a block basis

Introduction/Motivation (cont.)

- Contrast to physics applications:
  - Locations of the blocks in the global space is input- and data-dependent
  - Does not easily match common array distributions on the participating processes
- Evaluate productivity & performance of next-generation PGAS languages
  - For computational chemistry algorithms
  - Chapel & X10
- Selected a kernel from the Self-Consistent Field (SCF) method
  - Two-electron contribution to the Hartree-Fock matrix build
  - Basis for higher-order methods
  - Exhibits common behavior

Global Arrays – global-view distributed dense arrays

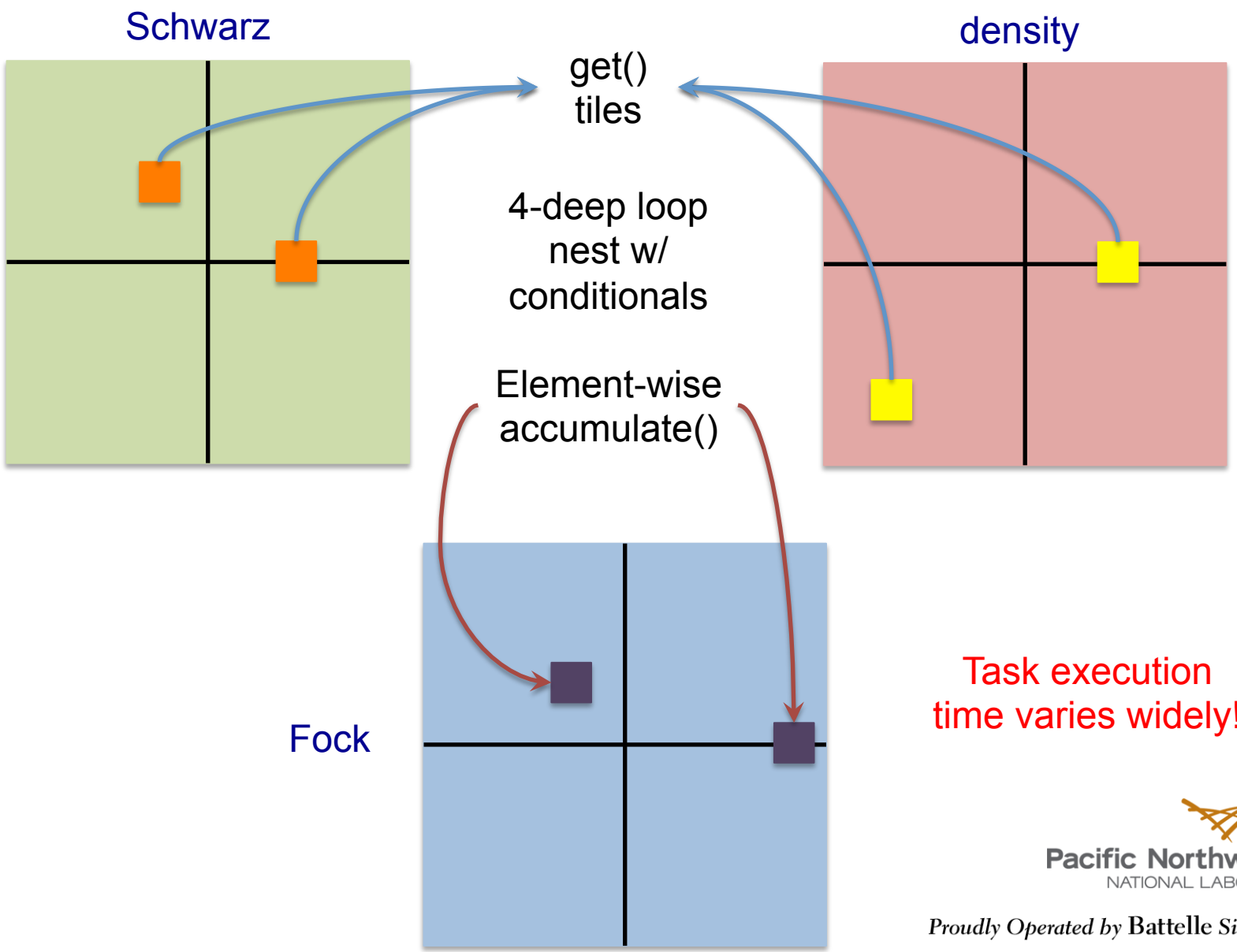
- Global Arrays (GA) is a library-based partitioned global address space (PGAS) programming model
  - Focused on enabling global-view access to distributed dense arrays
  - Developed over the past 20 years
  - High performance for production applications
- GA focuses on providing one-sided access to array slices in the global space
  - GA_Put(), GA_Get(), GA_Acc() primitives
- Communication support is provided by the Aggregate Remote Memory Copy Interface (ARMCI) runtime
  - Native ports over networks, as well as MPI ports
- SPMD control paradigm (same as MPI)



Two Electron Kernel

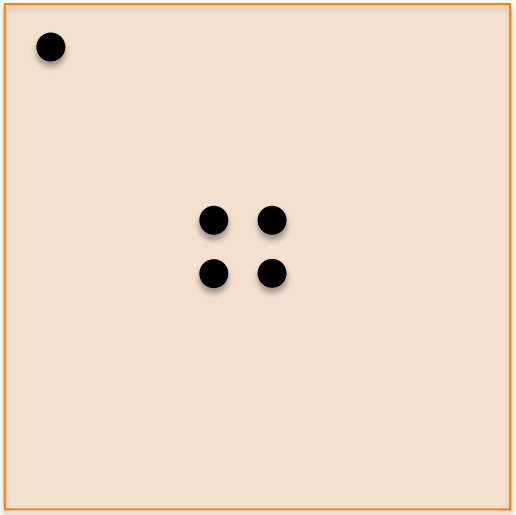
- Original code written using Global Arrays
  - GA instances for the Schwarz, density and Fock matrices
  - 2D block distribution of these matrices onto participating processes
  - Input: Schwarz and density matrices (read-only)
  - Output: Fock matrix (read-write)
- Two electron contribution:
  - Computationally sparse n^4 calculation over n^2 data space
  - Organized as a set of n^4 tasks which must be enumerated and evaluated
  - Most of the tasks do not add significant contributions to the Fock matrix
    - For larger inputs only <1% of them do

What is a task?

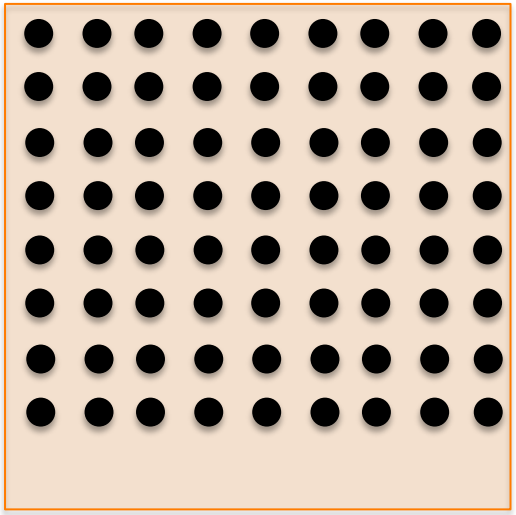


Task execution
time varies widely!

Data sparsity in the Schwarz matrix



Sparse Tile



Dense Tile

Determines which iterations (outer & inner) are executed in 4-deep loop nest

Use 40 x 40 tiles in the code

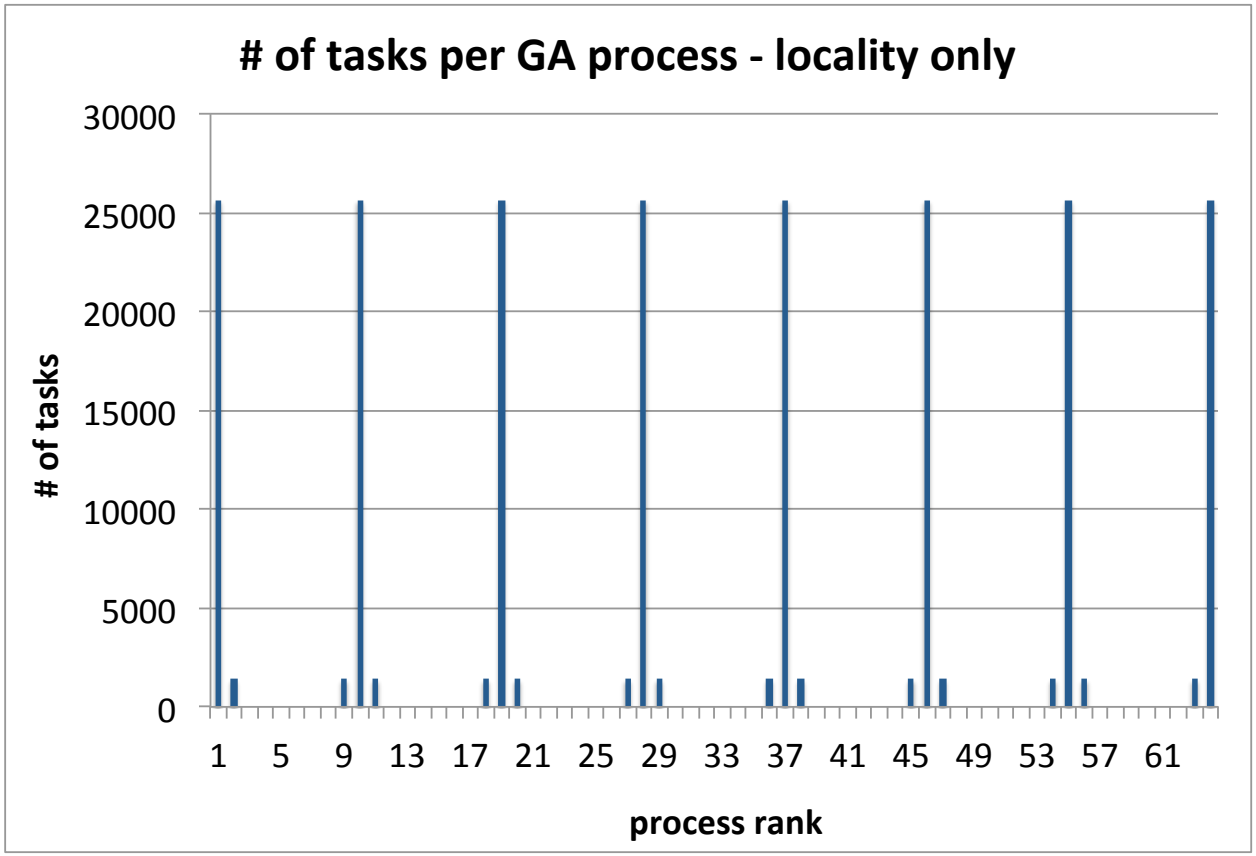
Structure of the PGAS test codes

1. Read files with Schwarz and density matrices
   - Captured from full SCF implementation
2. All process ranks/locales/places enumerate all n^4 tasks in a replicated manner
   - Skips tasks that access Schwarz tiles that do not belong to "me"
     - According to array distribution
3. Locally obtain those Schwarz tiles, analyze them for "non-zeroes"
   - Skip tasks that only have elements below threshold
4. Get 2nd Schwarz tile for non-zero tasks
   - May involve remote access
   - Skip tasks where the absolute value of all elements is below threshold

Structure of the PGAS test codes (cont.)

5. Compute “weight” of all tasks that passed tests
   - Weight is the number of non-zeroes in the element-wise product of the two Schwarz tiles
7. Load balance tasks on master rank/locale/place
   - Sort tasks in reverse order of weight
   - Distribute tasks to ranks/locales/places in round-robin (cyclic) manner
   - Very simple load balancing scheme suitable only for a few ranks/locales/places (easy to implement ☺)
8. Each rank/locale/place executes its list of tasks
   - Get Schwarz & density tiles, execute loop for each task, element-wise accumulate onto Fock tiles
9. Compute checksum on Fock matrix at the end
   - Validate correctness!

Work sparsity for tasks



What if we execute tasks where the data is located?

Pacific Northwest
NATIONAL LABORATORY

PGAS Implementations

- Baseline in GA with C++
  - Uses SPMD execution, non-multithreaded
- Chapel version using 2D block distributed arrays
  - Uses two levels of parallelism:
    - Locales
    - Multithreading for enumerating and executing tasks
- X10 version using 2D block distributed arrays
  - Uses a single level of parallelism
    - Places
  - Closer to the GA version
- Array distribution is equivalent between GA & X10, but not Chapel
  - We are distributing onto less locales

Language idioms & constructs used (Chapel)

- dmaps() for 2D block distributed arrays
- Standard module “templated” list for task lists
- 2D local type for 40x40 tiles
- Multi-level parallelism
  - coforall() over locales
  - forall() inside each locale
- Tiled array assignment:
  - `s_ij = schwarz(lo(1)..hi(1), lo(2)..hi(2));`
  - Just works: from distributed array to local tile
- Seamless remote data access, even for non-arrays
  - `ftaskLists(locid).append(fvtinfo(i));`
- Easy reductions:
  - `var gschwmax = max reduce schwarz;`

Pacific Northwest
NATIONAL LABORATORY

Language idioms & constructs used (X10)

- regionarrays for 2D block distributed arrays
- ArrayList for task lists
- 2D local type for 40x40 tiles
- Single-level parallelism
  - One async per place
  - `finish (for p in Place.places()) at (p) async { }`
- Biggest difference between Chapel and X10:
  - No remote access to data in X10
  - Must ship asyncs() to place where data is and copy it explicitly
- Easy reductions too:
  - `val schwmax = schwarz.reduce((a: Double, b: Double) => ((a > b) ? a : b), Double.MIN_VALUE);`

Pacific Northwest
NATIONAL LABORATORY

Subjective Qualitative Assessment

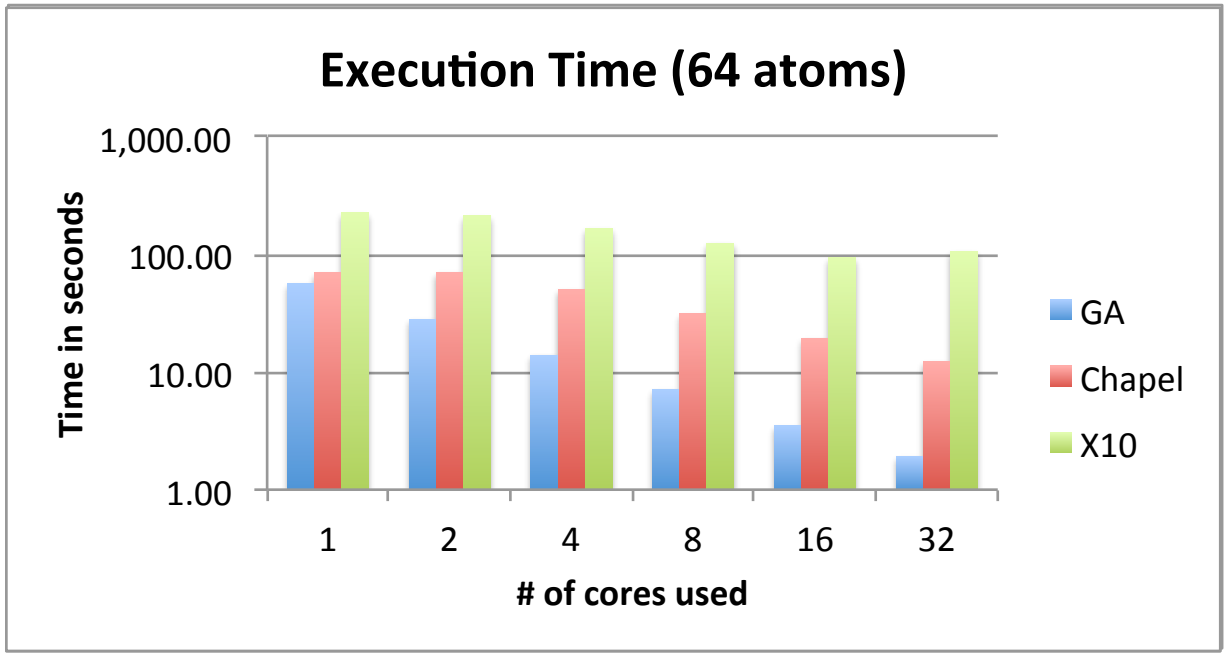
- Main differences:
  - Very different control style between GA & Chapel, X10
  - SPMD is really in your face in GA
    - `if (me == 0) {`
  - “Fork/join” feel in Chapel & X10
  - Data access paradigm is very different in all three:
    - GA: local data uses C/C++/Fortran semantics, global space data uses library semantics
    - Chapel: no syntactic distinction for local, remote accesses. It’s all in the declarations.
    - X10: no real remote access, must ship async() which can capture input/output data
- Similarities:
  - Distributed arrays are well supported in all three paradigms
  - Simpler “array distribution algebra” in GA, X10, richer in Chapel

Experimental Results

- Ran on up two nodes of our local Infiniband cluster:
  - Dual socket AMD Interlagos processors, 16 cores per socket, 64 GB RAM per node, QDR Infiniband
  - Used GCC 4.7.2 as the underlying compiler
  - Used OpenMPI 1.6.3 as the transport for X10 & Chapel
    - Could not get native GASNET Infiniband to work ☹ (SEGFAULT)
  - Used up to two worker threads per X10 place
  - Used ARMCI Infiniband native for GA
  - Full optimization for all three versions
- Input size used:
  - 64 atoms, resulting in 960^2 Schwarz, density and Fock matrices
  - 40 x 40 tiles
  - 960 / 40 = 24; 24^4 = 331,776 total tasks
  - 12,408 tasks after filtering (3.74%)
- Focus on **execution time** after tasks have been balanced

Experimental Results (cont.)

- One node results:
  - Chapel compiled with `-local` and `CHPL_COMM=none`
  - Using from 1 to 32 cores (GA & X10 use more processes, Chapel uses more threads)



Pacific Northwest
NATIONAL LABORATORY

Experimental Results (cont.)

- Two node performance was not competitive
  - Chapel compiled with and `CHPL_COMM=gasnet (MPI substrate)`
    - Used bulk communication and strided optimizations, as well as noRefCount
  - X10 compiled with full optimization (`-NO_CHECKS -OPTIMIZE_COMMUNICATIONS`)
- Communication pattern is complex
  - Strided blocks spread all over the global space
  - 4 input tiles, two output tiles
- Output tiles (Fock) have to be accumulated in an atomic element-wise manner:
  - `fock(lo(1)..hi(1), lo(2)..hi(2)).fetchAdd(f_ij);`
  - `gm(rp).getAndAdd(ta(ppos));`

Conclusions/Future Work

- The syntactic and semantic elements of next-generation PGAS languages are highly useful for computational chemistry algorithms
  - No major obstacles to implement code
- Control paradigm of both languages is quite different from traditional HPC SPMD paradigm
  - Fork/join flavor
- Data access paradigm differs between Chapel & X10
  - Chapel has an RMA (Remote Memory Access) flavor
  - X10 has an active message-like flavor
- Two dimensional block-distributed arrays well supported in both languages
  - Chapel has more flexibility & generality built in its array algebra

Conclusions/Future Work (cont.)

- Chapel's performance is competitive on one node
  - More challenging on two nodes
  - Due to the block-sparse access pattern? Atomic accumulation for the Fock matrix?
- X10's performance has more overhead on one node
  - Scales better to two nodes
  - Communication might be more explicit with Active Messages?
- Explore more explicit (lower-level) expressions of the code in Chapel
  - In collaboration with the Chapel team ☺
- Opportunity to tune compilers/runtimes for this kind of access pattern
  - More detailed profile of where time is being spent

Acknowledgements

- Thanks to the Chapel team members for their help and advice on the code
  - Brad Chamberlain
  - Vassily Litvinov
- Thanks to the X10 team members for their help and advice on the code
  - Dave Grove (IBM)
  - Josh Milthorpe (Australian National University)