High Productivity Language Systems: Next-Generation Petascale Programming

Presented by

Aniruddha G. Shet, Wael R. Elwasif,
David E. Bernholdt, and Robert J. Harrison

Computer Science and Mathematics Division
Oak Ridge National Laboratory

Revolutionary approach to large-scale parallel programming

- Million-way concurrency (and more) will be required on coming HPC systems.
- The current "Fortran+MPI+OpenMP" model will not scale.
- New languages from the DARPA HPCS program point the way toward the next-generation programming environment.
- Emphasis on performance and productivity.
- Not SPMD:
  - Lightweight "threads," *LOTS* of them
  - Different approaches to locality awareness/management
- High-level (sequential) language constructs:
  - Rich array data types (part of the base languages)
  - Strongly typed object oriented base design
  - Extensible language model
  - Generic programming

Candidate languages:
- Chapel (Cray)
- Fortress (Sun)
- X10 (IBM)

Based on joint work with
- Argonne National Laboratory
- Lawrence Berkeley National Laboratory
- Rice University

And the DARPA HPCS program

Concurrency: The next generation

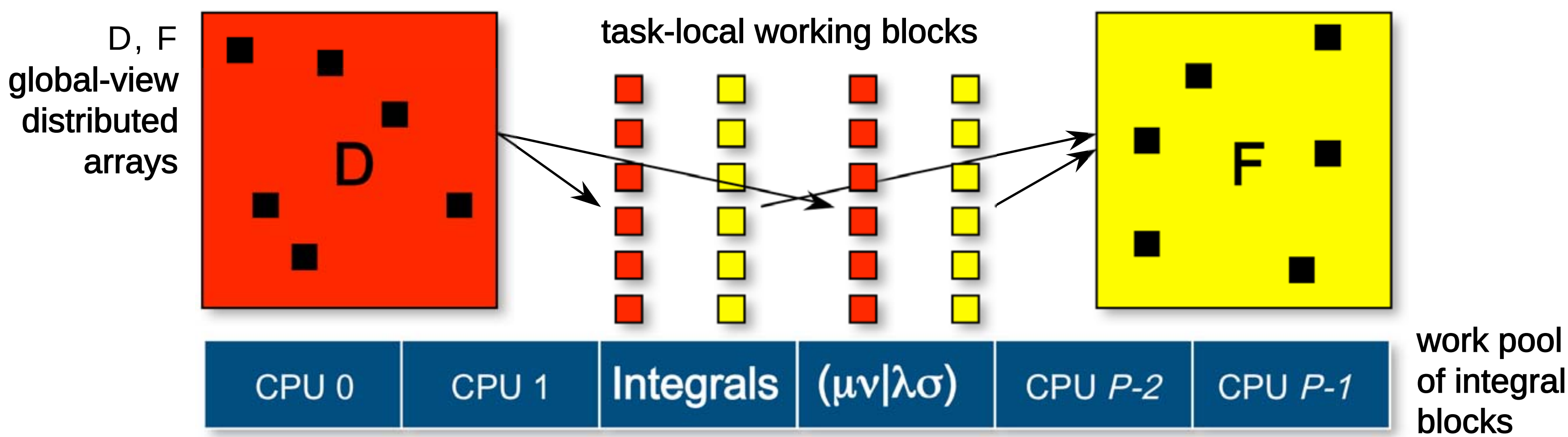
- Single initial thread of control
  - Parallelism through language constructs
- True global view of memory, one-sided access model
- Support *task* and *data* parallelism
- “Threads” grouped by “memory locality”
- Extensible, rich distributed array capability
- Advanced concurrency constructs:
  - Parallel loops
  - Generator-based looping and distributions
  - Local and remote futures

What about productivity?

- Index sets/regions for arrays
  - “Array language” (Chapel, X10)
- Safe(r) and more powerful language constructs
  - Atomic sections vs locks
  - Sync variables and futures
  - Clocks (X10)
- Type inference
- Leverage advanced IDE capabilities
- Units and dimensions (Fortress)
- Component management, testing, contracts (Fortress)
- Math/science-based presentation (Fortress)

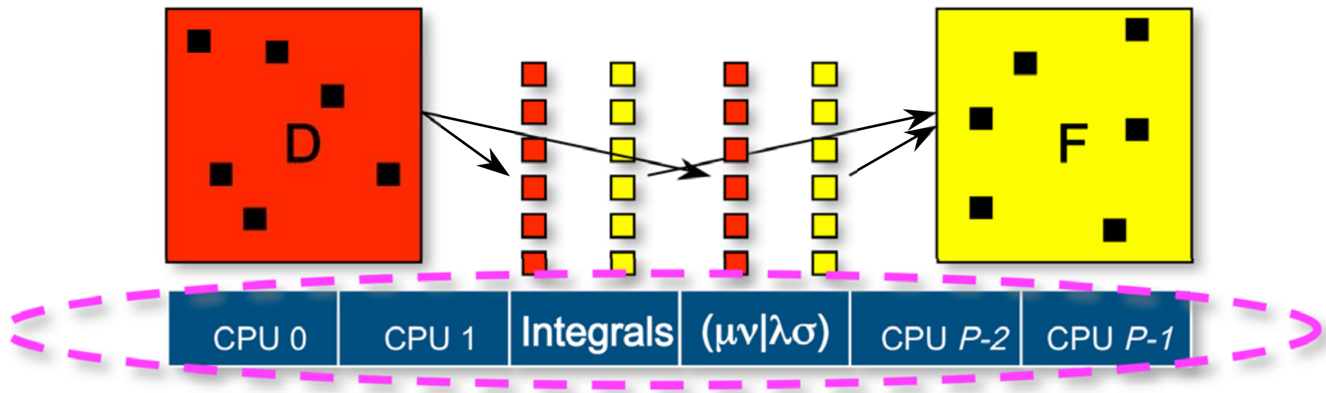
Exploring new languages: Quantum chemistry

- Fock matrix construction is a key kernel.
  - Used in pharmaceutical and materials design, understanding combustion and catalysis, and many other areas.
- Scalable algorithm is *irregular* in both data and work distribution.
  - Cannot be expressed efficiently using MPI.



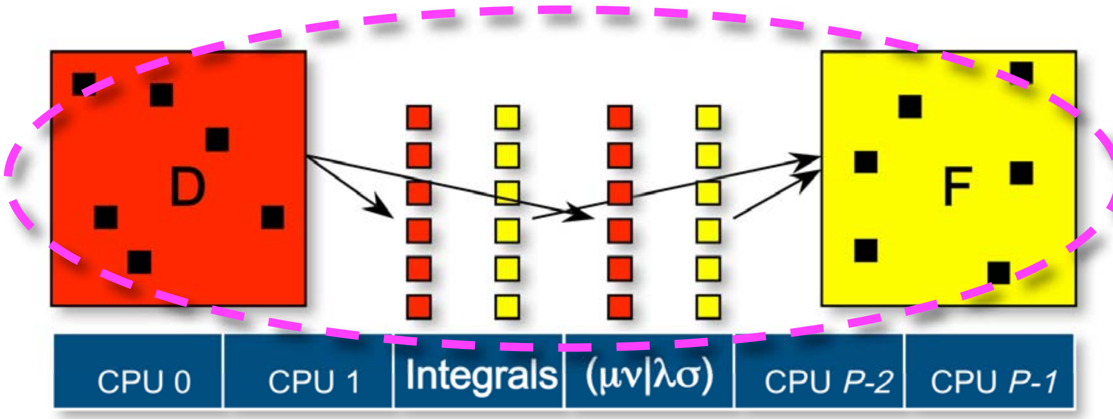
$$F_{\mu\nu} \leftarrow D_{\lambda\sigma} \, [\, 2\, (\mu\nu|\lambda\sigma) - (\mu\lambda|\nu\sigma) \,]$$

Load balancing approaches for Fock matrix build



Load balancing approach		Language constructs used		
		Chapel (Cray)	Fortress (Sun)	X10 (IBM)
Static, program managed		Unstructured computations + locality control	Explicit threads + locality control	Asynchronous activities + locality control
Dynamic, language (runtime) managed		Iterators + forall loops	Multigenerator for loops	*Not currently specified*
Dynamic, program managed	Task pool	Synchronization variables	Abortable atomic expressions	Conditional atomic sections + futures
	Shared counter	Synchronization variables	Atomic expressions	Unconditional atomic sections + futures

Parallelism and global-view data in Fock matrix build



Operations		Language constructs used: Chapel (Cray)	Fortress (Sun)	X10 (IBM)
Mixed data and task parallelism		Cobegin (task) + domain iterator (data)	Tuple (task) + for loop (data)	Finish async (task) + ateach (data)
Global-view array operations	Initialization	Array initialization expressions	Comprehensions / function expressions	Array initialization functions
	Arithmetic	Array promotions of scalar operators (+,*)	Fortress library operators (+,juxtaposition)	Array class methods (add,scale)
	Sub-array	Slicing	Array factory functions (subarray)	Restriction

Tradeoffs in HPLS language design

- Emphasis on parallel safety (X10) vs expressivity (Chapel, Fortress)
- Locality control and awareness:
  - X10: explicit placement and access
  - Chapel: user-controlled placement, transparent access
  - Fortress: placement "guidance" only, local/remote access blurry (data may move!!!)
  - *What about mental performance models?*
- Programming language representation:
  - Fortress: Allow *math-like* representation
  - Chapel, X10: Traditional programming language front end
  - *How much do developers gain from mathematical representation?*
- Productivity/performance tradeoff
  - Different users have different "sweet spots"

Remaining challenges

- (Parallel) I/O model
- Interoperability with (existing) languages and programming models
- Better (preferably portable) performance models and scalable memory models
  - Especially for machines with 1M+ processors
- Other considerations:
  - Viable gradual adoption strategy
  - Building a complete development ecosystem

Contacts

Aniruddha G. Shet
Computer Science Research Group
Computer Science and Mathematics Division
(865) 576-5606
shetag@ornl.gov

Wael R. Elwasif
Computer Science Research Group
Computer Science and Mathematics Division
(865) 241-0002
elwasifwr@ornl.gov

David E. Bernholdt
Computer Science Research Group
Computer Science and Mathematics Division
(865) 574-3147
bernholdtde@ornl.gov

Robert J. Harrison
Computational Chemical Sciences
Computer Science and Mathematics Division
(865) 241-3937
harrisonrj@ornl.gov

OAK RIDGE National Laboratory