Programming Languages for HPC

Is There Life After MPI?

Marc Snir
HPCS Assumptions

- HPC is hampered by lack of good programming language support
- In particular, the use of MPI leads to low software productivity
- Problem can be resolved by doing research on new programming languages
Paradigm Shifts

- New languages/models succeed only if they enable new capabilities.
  - In HPC, the drive has always been the need to exploit the performance of a new computer architecture: shift to vector, next to MPP

- Obstacles to the introduction of new languages are higher today than 20 years ago
  - Weight of existing software
  - Investment needed to create good compilers and good ADE’s.
Productivity Wall?

- Better language reduces programming time in some experiments (for non-expert programmers and short programs)
- Effect is probably small for large programs using OOP and for expert programmers
- Advantage may be reduced or negated if performance tuning is taken into account

Chart: Courtesy of IBM
Performance Wall (circa 2020)
Extrapolation of current Trends

- Single chip performance:
  - Memory wall: a processor chip executes ~ 100Kflop/s in the time needed to satisfy one load; need ~ 750 pending loads at anytime.
  - Heterogeneity: deep memory hierarchy and multiple forms of parallelism on chip

- “Everybody” has to face these problems.
  - Programming models that palliate these problems will come as a result of broad market need
  - But HPC community faces them earlier… (low cache utilization, compute intensive codes)
Performance Wall (circa 2020)
(continued)

- Large System Issues:
  - Global latency: 200 nsec = 0.7 Mflop/s
  - Efficient use of machines with > billion of concurrent operations
    - True, whether one uses many “light nodes” or fewer “heavy nodes”
  - Reliability
    - Problem for any large systems but harder for large, tightly coupled computations
    - Jitter, due to hardware, software or application
New Language has a Chance

- New language needed, not for software productivity, but for performance at Petascale.
- While we are at it, we may also improve software productivity.
What a New Language Should Do

- Address performance issues of large future systems
- Express well common HPC programming patterns
- Support well performance programming, with incremental code refinement
- Support OO
- Take advantage of advances in PL’s: strong typing, type and memory safety, atomicity, efficient support for generic programming
- Take advantage of advances in compilers: dynamic compilation, heuristic search, telescoping languages
- Coexist with existing languages
- Provide state of the art ADE
  - Largely built on language with large market
**PGAS Languages**

**Partitioned Global Address Space**

- Fixed number of processes, each with one thread of control
- Global partitioned arrays that can be accessed by all processes
  - Global arrays are syntactically distinct from local variables
  - Compiler generates communication code for each access
- Limited number of global synchronization calls

- CAF (Fortran), UPC (C), Titanium (Java)
Co-Array Fortran

- **Global array** $\equiv$ one extra dimension
  - `integer a[*]` - one copy of `a` on each process
  - `real b(10)[*]` - one copy of `b(10)` on each process
  - `real c(10)[3,*]` - one copy of `c(10)` on each process; processes indexed as 2D array

- **SPMD**
  - code executed by each process independently
  - communication by accesses to global arrays
  - split barrier synchronization
    
    `notify_team(team)`    `sync_team(team)`
Unified Parallel C

- (Static) global array is declared with qualifier shared
  - shared int q[100] – array of size 100 distributed round-robin
  - shared [*] int q[100] – block distribution
  - shared int* q – local pointer to shared

- SPMD model
  - code executed by each process independently
  - communication by accesses to global arrays
    - global barrier or global split barrier
    - upc_barrier, upc_notify, upc_wait
  - simple upc_forall: each iteration is executed on process specified by affinity expression
X10 (IBM)

- Based on Java
- Fixed number of places
  - places could migrate
- Each datum has one fixed place
  - arrays can be distributed
- Each place supports variable number of threads
  - thread can be spawned on locale
- Remote data can be accessed or updated only by spawning asynchronously remote activities (which may return a value – future)
- Synchronization constructs: “finish”, atomic sections and clocks
Chapel (Cray)

- Not based on existing language, but supports OO, and generic programming
- Data is distributed over “locales” - fixed location
  - Can add cache protocol (?)
- Mostly data parallelism
  - Array expressions
  - generalized forall
- Cobegin for parallel blocks
- Execution location may be controlled (in foralls and parallel blocks) via “on” expression
- Each thread can touch any variable
  - Weak memory consistency model
- Support for atomic sections
- Support for parallel reductions
Fortress (Sun)

- Not based on existing language
  - Safety features of Java, support for OO and generic programming, “math-like” syntax, support for vectors, matrices, etc.
- Shared global address space
- Loops are parallel by default
- data and loop iterates are distributed
  - distribution is defined by type with a distribution policy defined by a type-associated library (standard or user defined)
- Support for atomic sections
- Language is extendible via libraries that have syntactic and semantic compiler support (telescoping languages, Rose...)

Fortress Syntax

**ASCII**

\[ \rho_0 = r \cdot r \]

\[ v_{\text{norm}} = v / \text{norm } v \]

\[ \text{SUM}[k=1:n] \ a[k] \ x^k \]

\[ C = A \text{ UNION } B \]

**UNICODE**

\[ \rho_0 = r \cdot r \]

\[ v_{\text{norm}} = v / \|v\| \]

\[ \sum_{k=1}^{n} a_k x^k \]

\[ C = A \cup B \]

**Two-dimensional**

\[ \rho_0 = r \cdot r \]

\[ v_{\text{norm}} = \frac{v}{\|v\|} \]

\[ \sum_{k=1}^{n} a_k x^k \]

\[ C = A \cup B \]
Key Ideas

- Use of global name space (Convenience; All)
- Locales are explicit entities (Essential for managing locality; all but Fortress)
- “Local” and “Global” data accesses are syntactically distinct (Essential for efficient compilation; all)
- Extensibility (Fortress)
Requirements for Efficient Use of Global Name Space

- Names of data and threads are independent of their location
- Arrays are distributed (all)
- Rich set of distributions; e.g. block-block, user-defined (Chapel, Fortress)
- Arrays can be redistributed (possible in Fortress via suitable library)
- Computations can be dynamically allocated and reallocated (?)
Why Dynamic Data Redistribution?

courtesy Steve Ashby
Why Dynamic Process Migration?

Expanding cylinder model with 50 grains at each FE integration point

1/4 pipe cross section with shear bands

Exploding cylinder
Shear bands in a Ta-10W tube

With *adaptive sampling*, fine scale models do not run everywhere, but only where interpolation from previous response functions is not sufficiently accurate

courtesy Steve Ashby
Locale Virtualization

- **Static model**: Parallel programs written for constant number of processors, running at same speed and with same storage
  - Most new languages use fixed number of locales
- **Programs written for static model do not compose**
  - Composition requires either separate set of processors or transfer of control on all processors
- **Need virtualized locales** *(Fortress? X10?)*
- **Multiple gains accruing from virtualization**
  - Load balancing, communication/computation overlap, communication aggregation, improved cache performance...
New Languages -- Diagnostic

- Each of the proposed languages would be an advance over MPI, if properly implemented.
- All of the proposed languages still miss key features.
- None address directly node performance bottlenecks and scaling problems.
- X10, Chapel and especially Fortress require sophisticated compiler technology.
Non Technical Obstacles

- It takes money to make a good compiler; there is no market for HPC unique optimizations.
- It takes time to make a good compiler; there is no funding mechanism for a sustained 5 years development effort.
- It takes people to make a good compiler; there is no independent compiler company.
  - Should hw vendors develop the HPC ADE?
Minimal Solution Beyond MPI

- Compiled communication, to avoid software overhead
  - Possibly, inlining and optimization of key MPI calls
  - Alternatively, simple language extensions for access and update of remote variables (v; v@proc)
    - Not that different from CAF!
- Process virtualization, to support composability, load balancing, communication/computation overlap, communication aggregation, improved cache performance...
A Good ADE is More than Language

- Porting tools
- Good support for performance tuning
  - Tools for refactoring
  - Notation for capturing tuning decisions
- Good observability
  - Integrated performance stream mining
Summary

- HPC is hampered by lack of good software support
- Language is only part of the problem
- Most obstacles are not technological
- Key issues for petascale computing are not yet being addressed
  - HPCS is driving high quality parallel computing language research, but this research pays little attention to petascale