Illinois Informatics Institute
Invent Imagine Innovate

Programming Models for Supercomputing in the Era of Multicore

Marc Snir

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

www.informatics.uiuc.edu

MULTI-CORE CHALLENGES

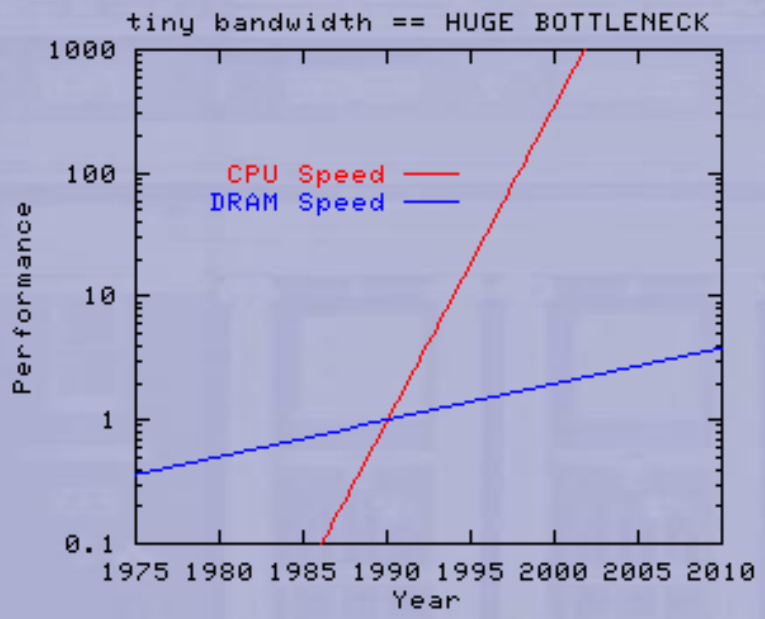
Moore's Law Reinterpreted

- Number of cores per chip doubles every two years, while clock speed decreases
  - Need to handle systems with millions of concurrent threads
    - *And contemplate with horror the possibility of systems with billions of threads*
  - Need to emphasize scalability – not best performance for fixed number of cores.
  - Need to be able to easily replace inter-chip parallelism with intra-chip parallelism
    - *Homogeneous programming model (e.g., MPI all around) is preferable to heterogeneous programming model (e.g., MPI+OpenMP).*

Memory Wall: a Persistent Problem

- Chip CPU performance increases ~60% CGAR
- Memory bandwidth increases ~25%-35% CGAR
- Memory latency decreases ~5%-7% CGAR
- Most area and energy chip budget is spent on storing and moving bits (*temporal and spatial communication*)

- *Locality and communication management are major algorithmic issues, hence need be exposed in programming language*



Reliability and Variance

- Hypothesis: MTTF per chip does not decrease – hardware is used to mask errors
  - Redundant hardware, redundant computations, light-weight checkpointing
- Programmers do not have to handle faults
- Programmers have to handle variance in execution time
  - Variance due to size – square root growth
  - Variance due to error correction
  - Variance due to power management
  - Manufacturing variance
  - Heterogeneous architectures
  - System noise (jitter) – not much of a problem, really
  - *Application variance:* adaptive codes (e.g., AMR), multi-scale codes
- *Loosely synchronous SPMD model with one thread per core breaks down – need virtualization*

NEW OPPORTUNITIES

Ubiquitous Parallelism

- Effective leverage of parallelism is essential to the business model of Intel and Microsoft
- Focus on turnaround, not throughput
- Need to enable large number of applications – need to develop parallel application development environment

However:

- Four orders of magnitude gap
- Focus on ease of programming and scalability, not peak performance
- Little interest in multi-chip systems

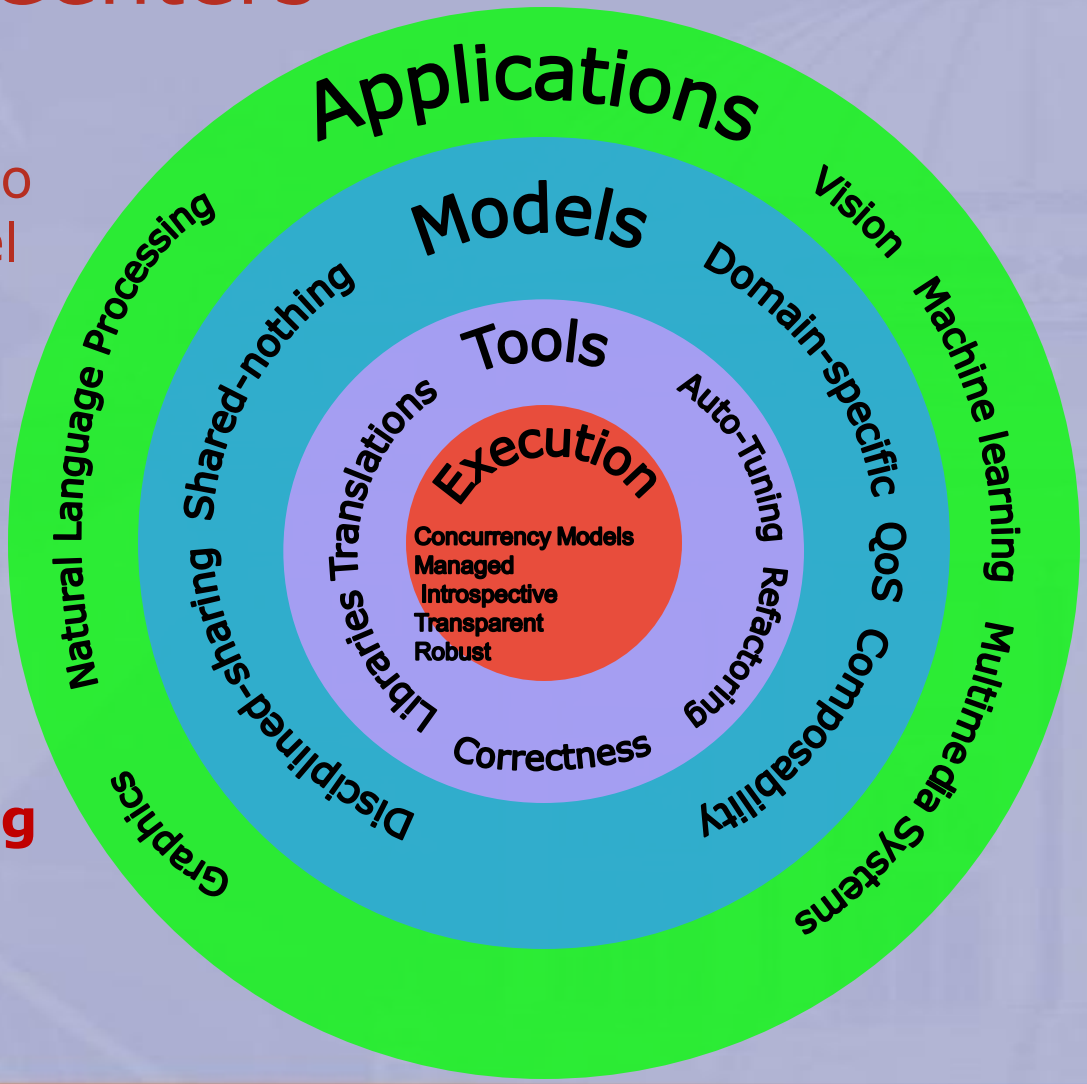
Q: how can HPC leverage the client parallel SW stack?

Expected "Trickle-Up" Technologies

- New languages (much easier to "stretch" well-supported parallel languages than have DARPA create a market for new HPC languages)
- Significantly improved parallelizing compilers, parallel run-times, parallel IDEs (bottleneck has been more lack of market, less lack of good research ideas)
- New emphasis on deterministic (repeatable) parallel computation models – focus on producer-consumer synchronization, not on mutual exclusion
  - Serial semantics, parallel performance model
    - *Parallel algorithms are designed by programmers, not inferred by compilers*
- Every computer scientist will be educated to "think parallel"

Universal Parallel Computing Research Centers

"Intel and Microsoft are partnering with academia to create two Universal Parallel Computing Research Centers…located at UC Berkeley and UIUC".



Goal:

Make parallel programming synonymous with programming

UPCRC One Slide Summary

- Parallel programming can be a child's play
  - E.g., Squeak Etoys
  - No more Swiss army knifes: need, and can afford, multiple solutions
- Simplicity is hard
  - Simpler languages + more complex architectures = a feast for compiler developers
- Need more abstraction layers that abstract both semantics and QoS
  - What is a QoS preserving mapping?
  - What hooks can HW provide to facilitate programming?
    - *Sync primitives, debug/perf support*
- Performance will enable new client applications
  - An intelligent PDA will need, eventually, the compute power of a human brain

KEY TECHNOLOGIES

Task Virtualization

- Multiple logical tasks are scheduled on each physical core; tasks are scheduled nonpreemptively; task migration is supported
  - Hides variance and communication latency
  - Helps with scalability
  - *Needed for modularity*
  - *Improves performance*
  - E.g., AMPI, Charm++ (Kale, UIUC), TBB (Intel) …
  - Supported by hardware and/or run-time
  - Can be implemented below MPI or PGAS languages

- Two styles:
  - Varying, user controlled number of tasks (AMPI)
    - *Locality achieved by load balancer*
  - Recursive (hierarchical) range splitter (TBB)
    - *Locality achieved implicitly*

Compiled Communication

- Replace message-passing library (e.g., MPI) with compiler generated communication (e.g., PGAS languages)
  - Avoids SW overhead and memory copies of library calls
  - Maps directly to platform specific HW communication mechanisms, in a portable manner
    - *Transparently port from shared memory HW to distributed memory HW*
  - Enables compiler optimization of communications, across multiple communications

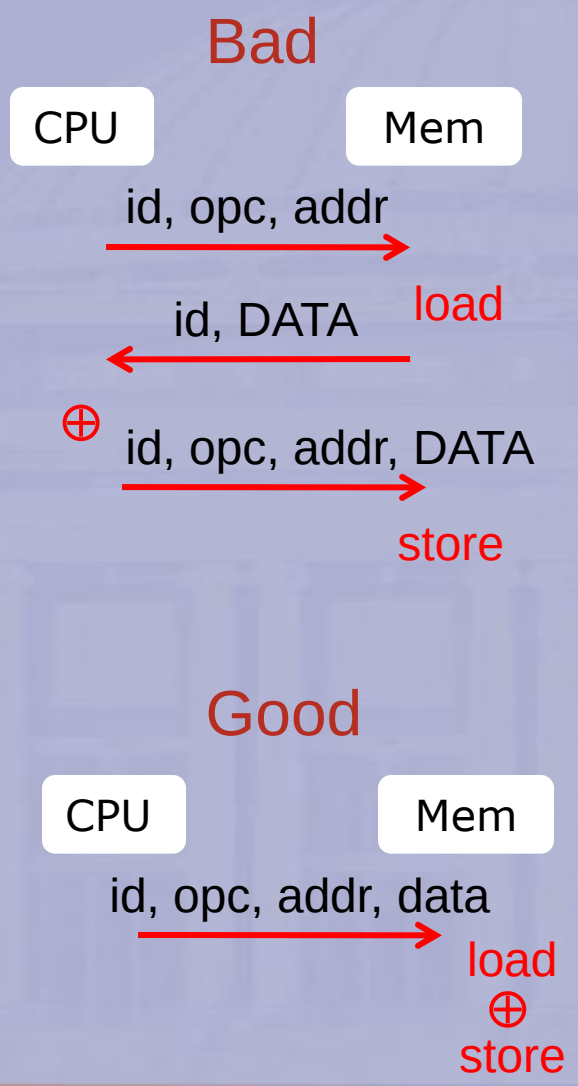
Shared Memory Models

- Global name space – variable name does not change when its location changes – simplifies programming
  - Not copying but caching
  - Well explored alternatives: Pure HW caching (coherent shared memory); and pure SW caching (compiled, e.g. for PGAS languages).
  - Need to better explore HW-assisted caching (fast, HW-supported, cache access; possibly slow cache update)
  - Probably need some user control of caching (logical cache line definition)
  - Probably don't need user control of home location (as done in PGAS languages)
- *Conflicting accesses must be synchronized*
  - Current Java and soon to be C++ semantics
- *Races must be detected and generate exceptions*
  - Can be done with some variants of thread level speculation

Synchronization Primitives

- Frequently needed: *Deterministic synchronization*
  - producer-consumer: barrier, disciplined use of full/empty bits (single writer)
  - Accumulate (reduction)
- Rarely needed: *Nondeterministic synchronization*
  - mutual exclusion, atomic sections (transactions)

- Note: transactional memory HW good for lightly contested atomic sections; not efficient for producer-consumer synchronization and for reductions
- Simple accumulates best done on memory side (if core contributes few values)
  - Significant reduction of memory bus traffic
  - Requests can be combined, to avoid congestions

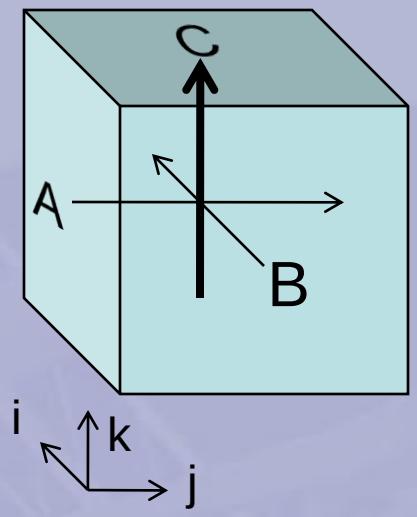


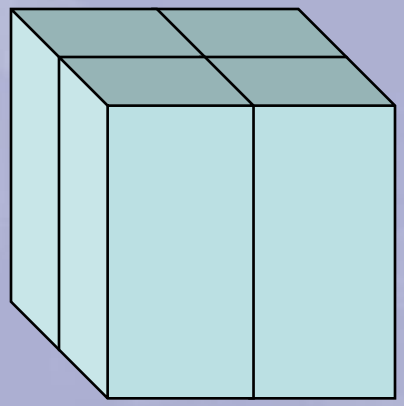
Parallel Patterns Challenge

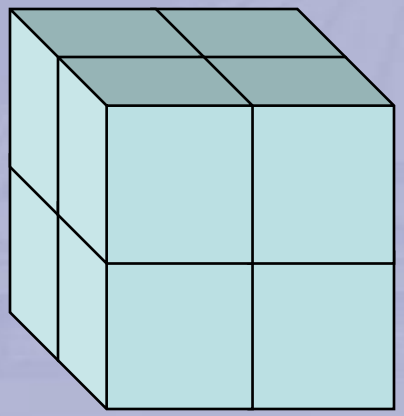
EASY TO EXPLAIN PARALLEL ALGORITHM = SIMPLE CODE

A x B = C

Matrix Product







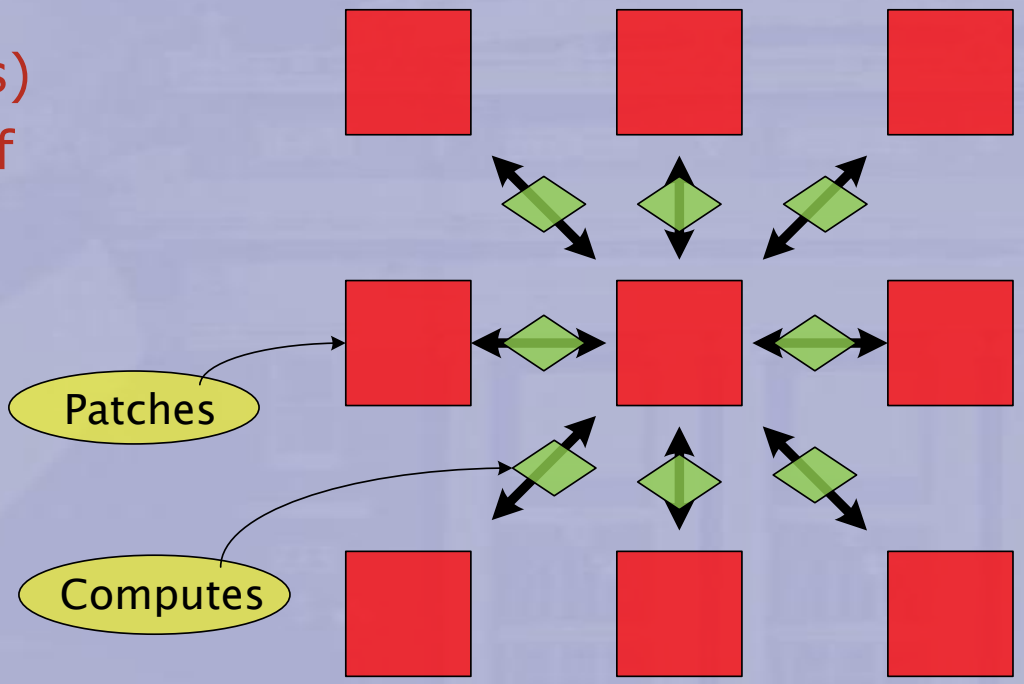
Generic | Data Parallel | Control Parallel

- Need to easily express 3D computation domain
  - One 3D iterator, not a triple nested loop
- Need to easily express partition into subcubes
  - Automatically generate parallel reduction
  - Avoid allocation of n^3 variables

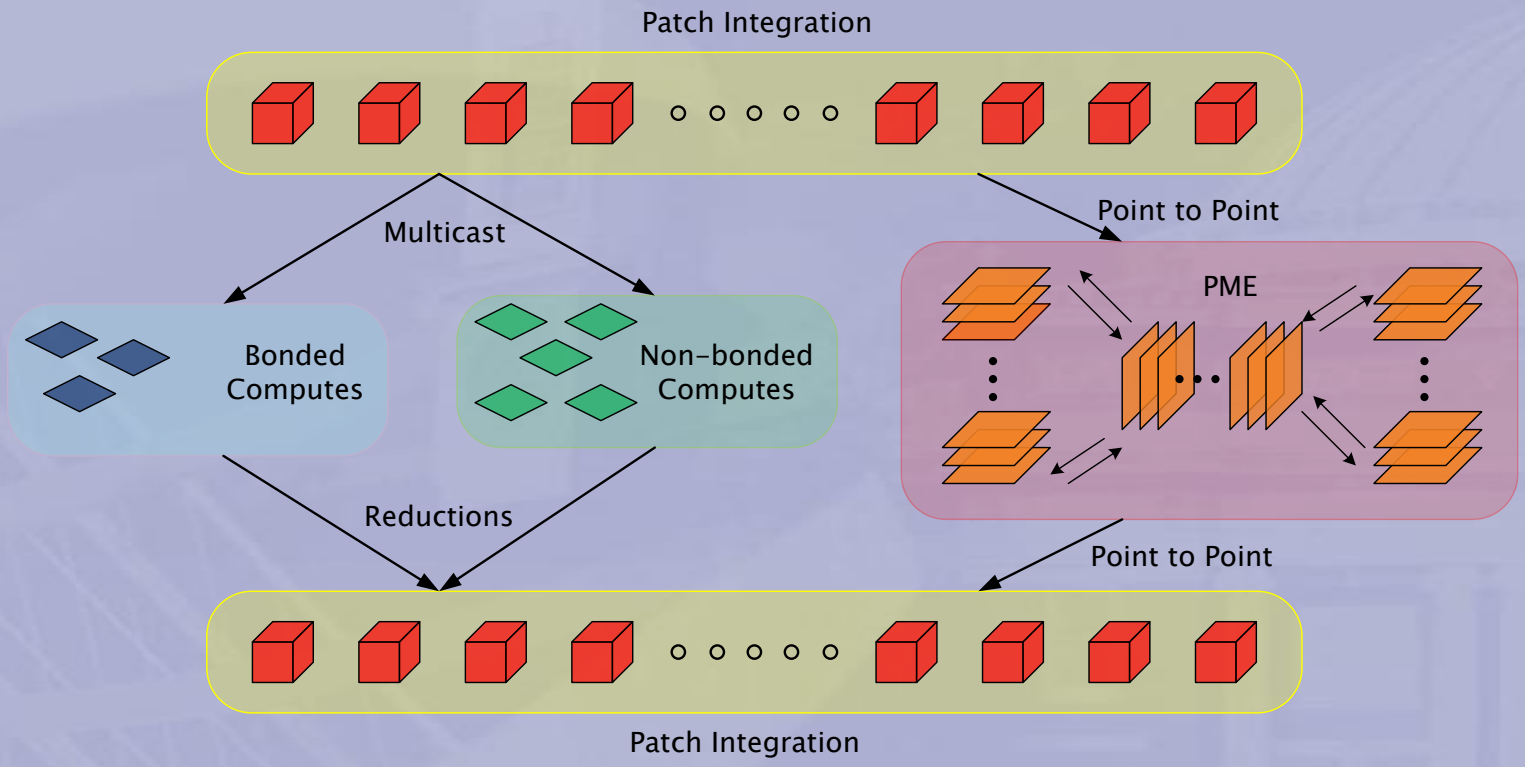
Less Trivial Example: NAMD

(Kale)

- Patches object: atoms in cell
  - Change each iteration (or each few iterations)
- Computes object: pairs of atoms from neighboring cells (to compute forces)
  - Avoid allocation of variable for each pair
- Can one go from such declarative description to code?



NAMD Communication

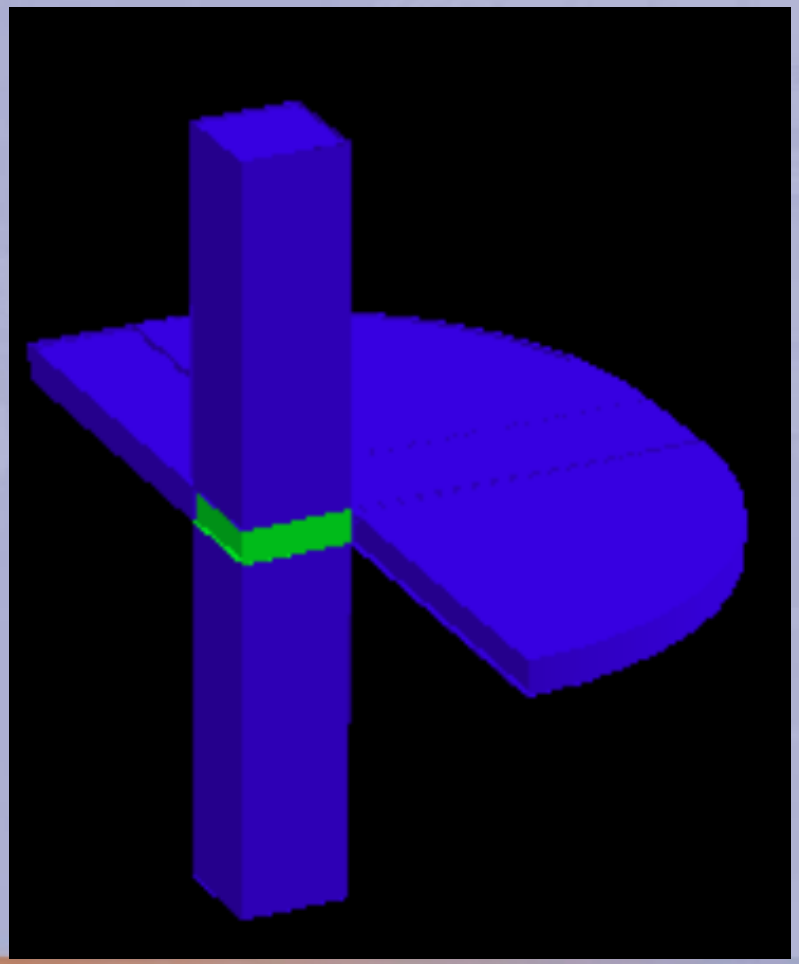


- Can one have composition language that expresses above diagram in natural way?

Algorithmic Changes, for Scale

- If cell size < cutoff radius, then computes object should consist of pairs of atoms from *subset* of cells within cutoff radius [Snir/Shaw]
- Can choose 1D FFT or 2D FFT

- Can one delay binding until problem and machine parameters are known?

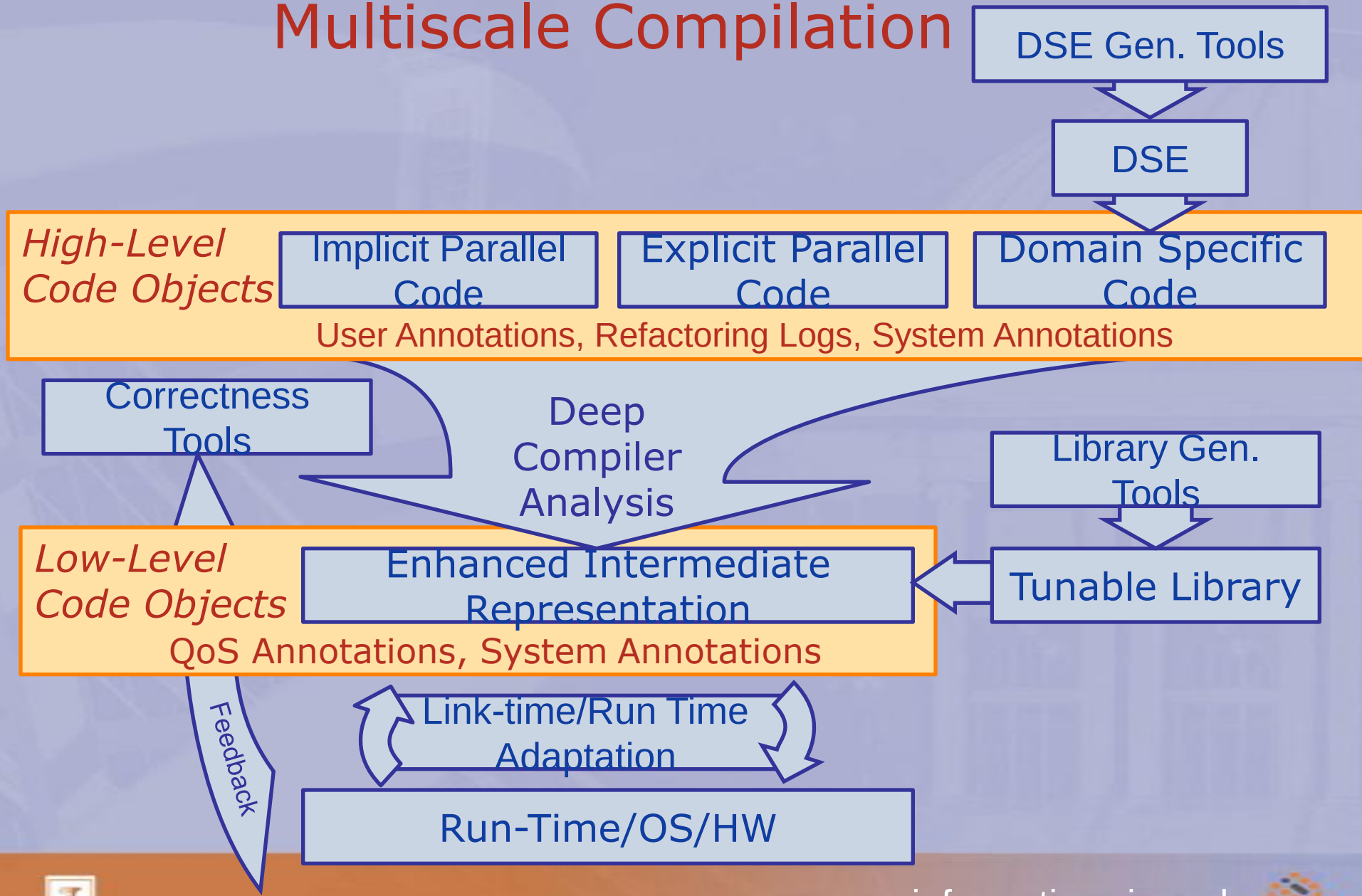


THE SOLUTION IS MULTITHREADED

Multiscale Programming

- Multiple languages (C, C++, Fortran, OpenMP, Python, CAF, UPC) and libraries
- Multiple levels of code generation & tuning
  - Domain specific code generators & high-level optimizers (Spiral – Puschel et al, quantum chemistry -- Sadayappan et al)
  - Library autotuning – tuning pattern selection
  - Algorithm selection
  - Refactorings and source to source transformations
  - Static compilation
  - Template expansion
  - Run-time compilation – continuous optimization
- *Do not think parallel language; think programming environment that integrates synergistically all levels*

Multiscale Compilation



Summary

- The frog is boiling:
  - tuning code is ridiculously hard and is getting harder
- We have the power to change:
  - We can build much better parallel programming environments – the problem is economics, not technology
- There is no silver bullet:
  - Not one technology, but a good integration of many
- I ran out of platitudes
  - Time for

Questions?