

# Language and Virtual Machine Challenges for Large-scale Parallel Systems

**Vivek Sarkar**

**IBM T.J. Watson Research Center**

**([vsarkar@us.ibm.com](mailto:vsarkar@us.ibm.com))**

**Workshop on the Future of**

**Virtual Execution Environments**

**Sep 15 – 17, 2004**



# Acknowledgments

- **IBM PERCS Team members**
  - Research
  - Systems & Technology Group
  - Software Group
  - PI: Mootaz Elnozahy
- **University partners:**
  - Cornell
  - LANL
  - MIT
  - Purdue University
  - RPI
  - UC Berkeley
  - U. Delaware
  - U. Illinois
  - U. New Mexico
  - U. Pittsburgh
  - UT Austin
  - Vanderbilt University
- **Contributors to X10 design & implementation ideas:**
  - David Bacon
  - Bob Blainey
  - Philippe Charles
  - Perry Cheng
  - Julian Dolby
  - Kemal Ebcioğlu
  - Guang Gao (U Delaware)
  - Allan Kielstra
  - Robert O'Callahan
  - Filip Pizlo (Purdue)
  - Christoph von Praun
  - V.T.Rajan
  - Lawrence Rauchwerger (Texas A&M)
  - Vijay Saraswat (contact for lang. spec.)
  - Vivek Sarkar
  - Mandana Vaziri
  - Jan Vitek (Purdue)



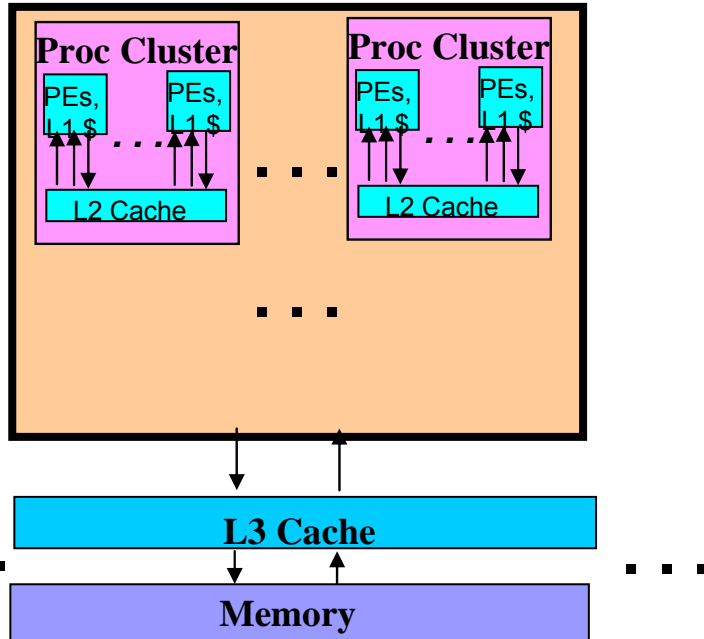
# Benefits of Virtual Execution Environments are well recognized ...

---

- **Safety**
- **Productivity**
- **Portability**
- **Interoperability**
- **Isolation**
- **Virtualization**
- ...

# ... but, are VEE's ready to address the hardware challenges facing future large-scale parallel systems?

1) Memory wall: Severe non-uniformities in bandwidth & latency in memory hierarchy



2) Frequency wall: Multiple layers of hierarchical heterogeneous parallelism to compensate for slowdown in frequency scaling

Clusters (scale-out)
SMP
Multiple cores on a chip
Coprocessors (SPUs)
SMTs
Vector (VMX)
ILP

3) Scalability wall: Software needs to deliver ~  $10^5$ -way parallelism to utilize large-scale (capability) parallel systems

# And what about Software Productivity, Reliability, and Business Challenges in large-scale parallel systems?

---

- **Productivity Challenges**

- Concurrent programming using today's programming models and tools is only (and barely) accessible to experts
- Poor debugging tools (for both correctness & performance debugging)
- Poor support for migrating legacy software

- **Reliability Challenges**

- Many long-running applications are just one fault away from total failure
- Current responsibility for providing recovery support falls outside the VEE/compiler layer

- **Business Challenges**

- System must demonstrate utility in multiple application domains, with opportunities for software reuse across multiple hardware generations

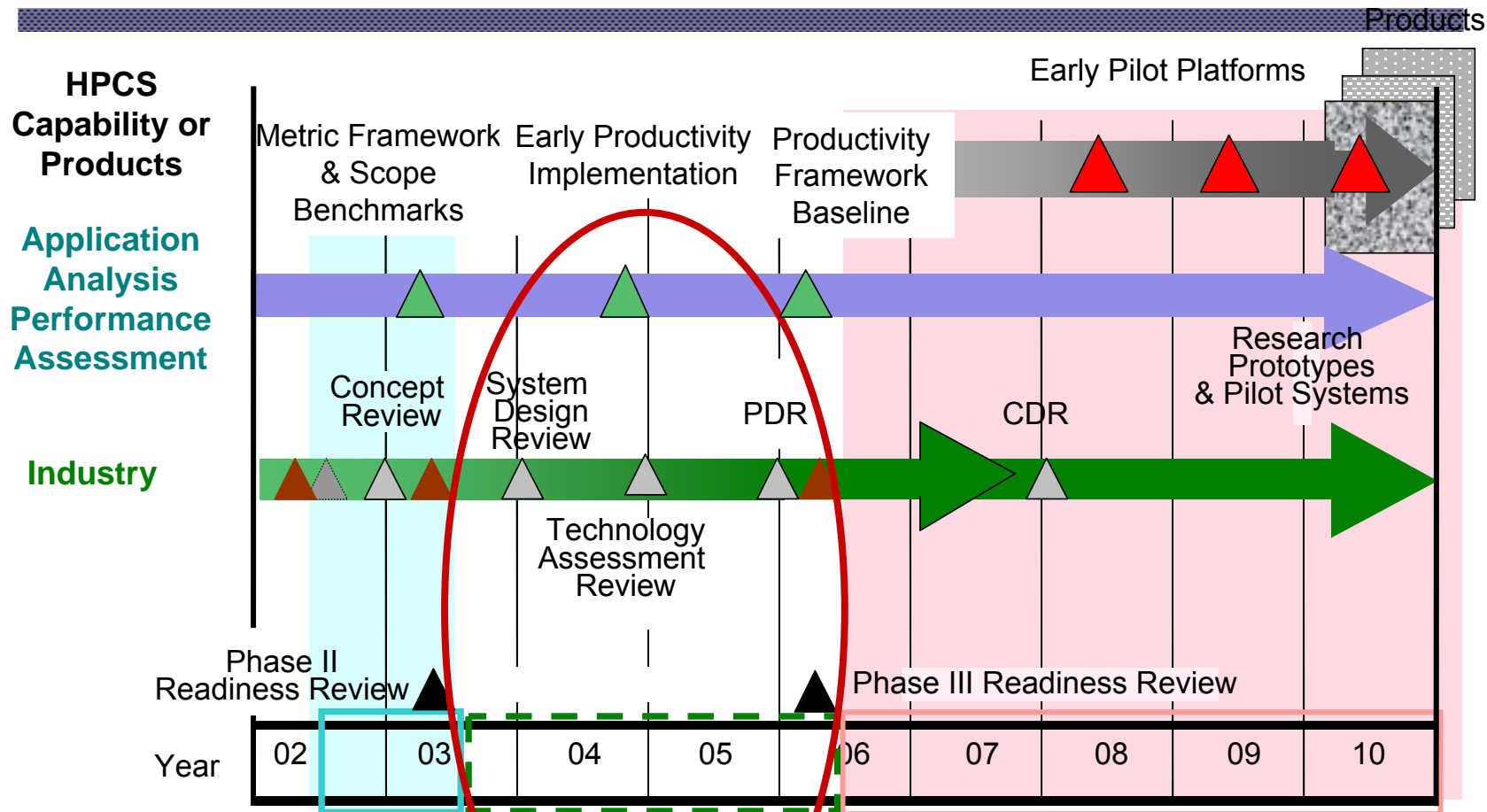
# Answer: not yet, but we believe that VEE's can evolve to address these challenges by 2010 ...

---

... some promising signs are as follows:

- In commercial domain
  - VEE's have contributed to increased programmer productivity
  - Heavy investment in virtualization and isolation is leading to more flexible & reliable software for VEE's in scale-out configurations
- In HPC domain
  - Experiences with distributed-memory (scale-out) configurations has improved the understanding of how to address Memory Wall and Scalability Wall issues
- DARPA program on High Productivity Computing Systems (HPCS)
  - Focused attention on improving productivity, while delivering acceptable performance on large-scale systems
    - VEEs are a core part of the agenda for 2 out of 3 HPCS vendors
- This workshop!
  - Innovations that can be expected in future VEE's

# HPCS Program Phases I - III



- △ Reviews
- ▲ Industry Procurements
- ▲ Critical Program Milestones



# IBM PERCS Project

## (Productive Easy-to-use Reliable Computing Systems)

*Increase overall productivity*

Increase number of applications written

**Increase development productivity**

**PERCS Programming Tools**  
performance-guided parallelization and transformation, static & dynamic checking, separation of concerns --- all integrated into a single development environment (Eclipse)

PERCS Programming Model

OpenMP

MPI

Static and Dynamic Compilers for base language w/ programming model extensions  
Mature languages: C/C++, Fortran, Java  
Experimental languages: X10, UPC, StreamIt, HTA/Matlab

Language Runtime + Dynamic Compilation + Continuous Optimization

PERCS System Software (K42)

PERCS System Hardware

Increase performance of applications

**Increase execution productivity**



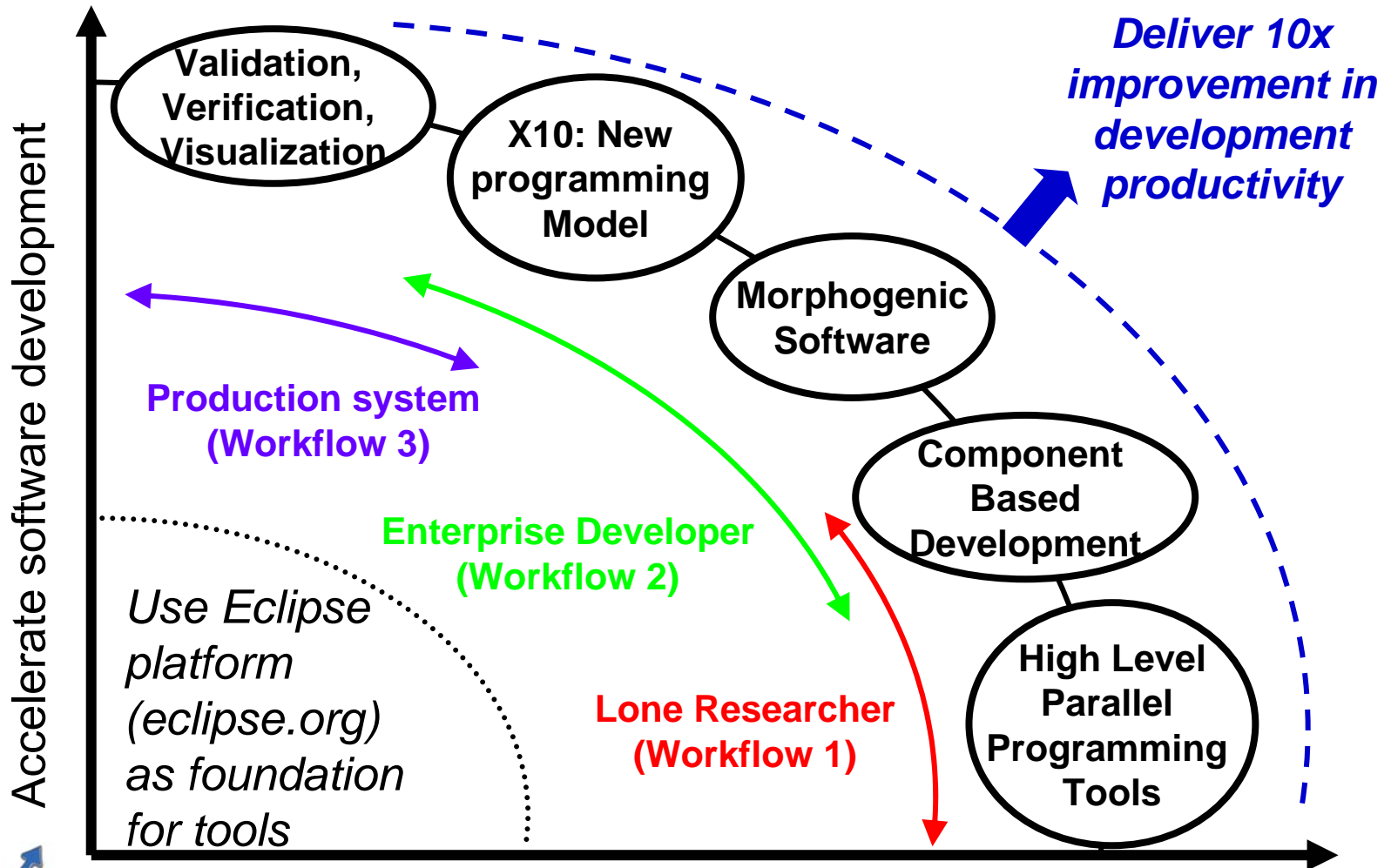
# Problem Statement for PERCS Programming Model & Tools

---

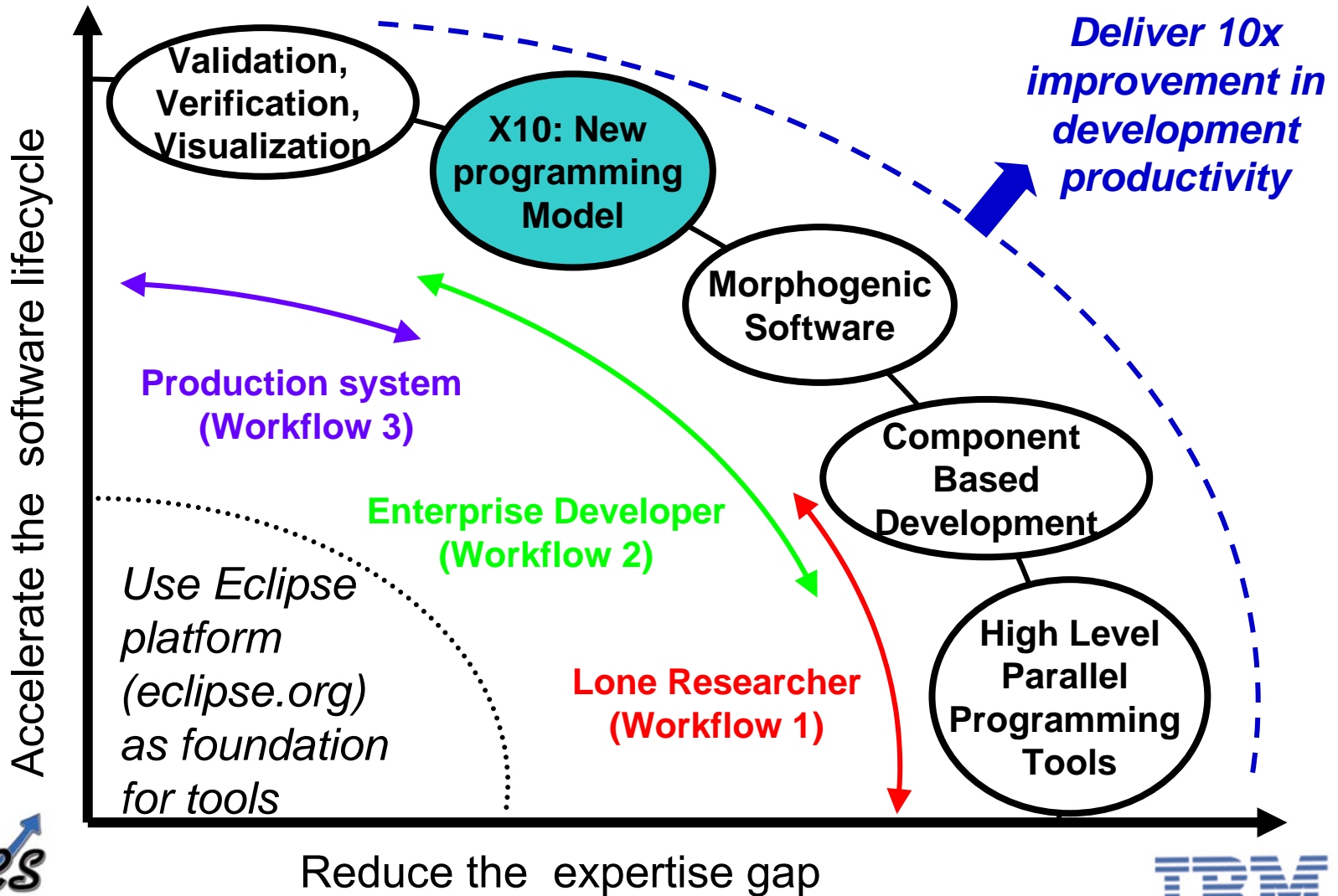
- **Grand Challenge**
  - *Deliver 10x improvement in HPC application development productivity over today's systems by 2010, while delivering acceptable performance on large-scale systems*
- **Our hypothesis:** the two fundamental obstacles to improving HPC application development productivity are
  1. **Expertise gap**
  2. **Programming complexities**



# PERCS Programming Model and Tools Agenda



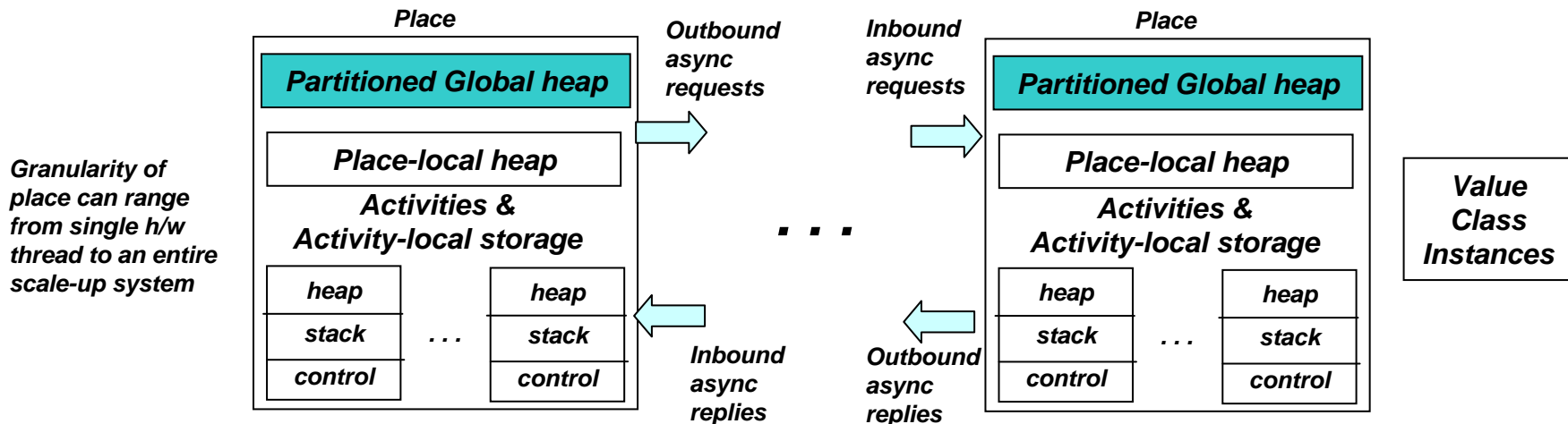
# PERCS Programming Model and Tools Agenda



# X10 Design Guidelines

- **Start with state-of-the-art OO language primitives as foundation**
  - No gratuitous changes
  - Build on existing skills
- **Raise level of abstraction for constructs amenable to optimized implementation**
  - Monitors → atomic sections
  - Threads → async ops
  - Barriers → clocks
- **Introduce new constructs to model hierarchical parallelism and non-uniform data access**
  - Places
  - Distributions
- **Support common parallel programming idioms**
  - Data parallelism
  - Control parallelism
  - Divide-and-conquer
  - Producer-consumer / streaming
  - Message-passing
- **Ensure that every program has a well-defined semantics**
  - Independent of implementation (compiler optimizations, system architecture, processor design, memory model etc).
- **Defer fault tolerance and reliability issues to lower levels of system**
  - Assume tightly-coupled system with dedicated interconnect

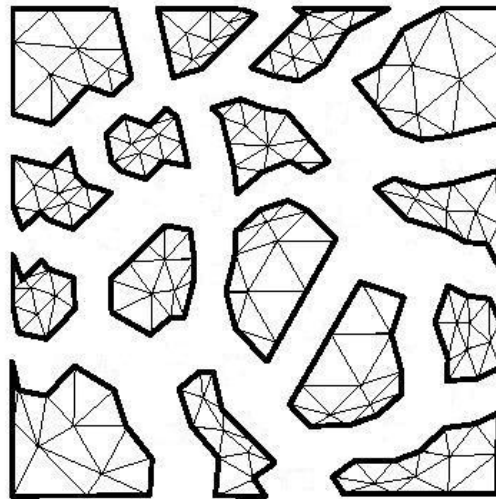
# Logical View of X10 Programming Model (Work in progress)



- **Place = collection of resident activities and data**
  - Maps to a data-coherent unit in a large scale system
- **Four storage classes:**
  - Partitioned global
  - Place-local
  - Activity-local
  - Value class instances
- **Activities can be created by**
  - Asynchronous statements (one-way msgs)
  - Asynchronous expressions (futures)
  - foreach and ateach constructs
- **Activities are coordinated by**
  - Atomic sections
  - Conditional atomic sections
  - Clocks (generalization of barriers)
  - Force (for result of future)

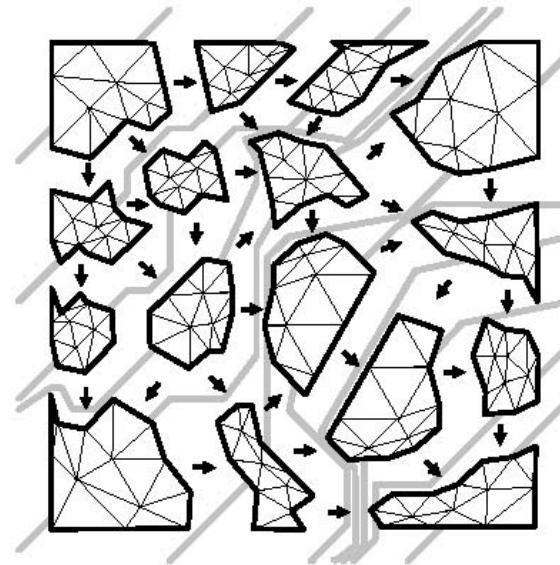
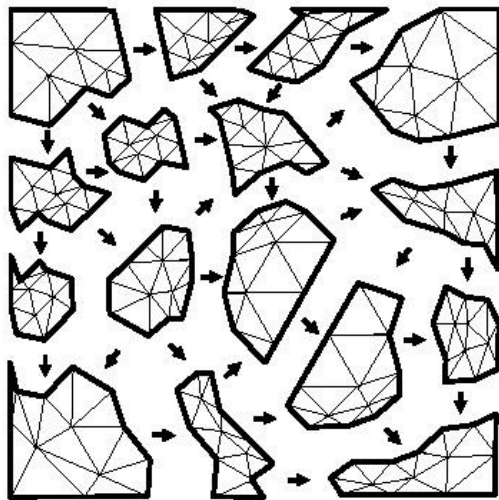
# Unstructured Mesh Transport Example (UMT2K)

- 3D, deterministic, multi-group, photon transport code
- Solves 1<sup>st</sup> order form of steady-state Boltzman equation
- Represented by an unstructured mesh
  - Partitioning strives to maintain load balance, reduce communicate/compute ratio



# Communication Structure

- Nearest neighbor communication in graph domain
- Communication can be minimized via judicious mapping of graph to system nodes



# UMT2k in X10: example of hierarchical heterogeneous parallelism

```
do {
  now ( c ) {
    ateach ( n : nodes ) { // Cluster-level parallelism
      foreach ( s : Sweeps ) { // SMP parallelism
        // receive inputs
        flows = new Flux[R] (k) { // SMT parallelism
          async (...) inputs[s][k].receive();
        }
        // Choice of using clock or force to synchronize on flows[*]
        // Thread-local with vector & co-processor parallelism
        flux = compute(s, flows);
        // send outputs
        ...
      } // foreach
    } // ateach
  } // now
  // use clock c to wait for all sweeps to complete
  next c;
  ...
} while ( err > MAX_ERROR );
```

Clusters (scale-out)
SMP
Multiple cores on a chip
Coprocessors (SPUs)
SMTs
Vector (VMX)
ILP



# C+MPI FixedPoint iteration (Simpler example than UMT2K)

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

int n;
double *A, *Tmp;
const double epsilon = 0.000001;
int main(int argc, char* argv[]) {
    int i, iters;
    double delta;
    int numprocs, rank, mysize;
    double sum;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (argc != 2) {
        printf("usage: fixedpt n\n");
        exit(1);
    }
    n = atoi(argv[1]);
    mysize = n * (rank+1)/numprocs - n * rank / numprocs;
    A = malloc((mysize+2)*sizeof(double));
    for (i = 0; i <= mysize; i++) A[i] = 0.0;
    if (rank == numprocs - 1) A[mysize+1] = n + 1.0;
    Tmp = malloc((mysize+2)*sizeof(double));
    iters = 0;
```

```
do {
    iters++;
    if (rank < numprocs -1)
        MPI_Send(&(A[mysize]), 1, MPI_DOUBLE, rank+1, 1,
            MPI_COMM_WORLD);
    if (rank > 0)
        MPI_Recv(&(A[0]), 1, MPI_DOUBLE, rank-1, 1,
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    if (rank > 0)
        MPI_Send(&(A[1]), 1, MPI_DOUBLE, rank-1, 1,
            MPI_COMM_WORLD);
    if (rank < numprocs-1)
        MPI_Recv(&(A[mysize+1]), 1, MPI_DOUBLE, rank+1, 1,
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    for (i=1; i <=mysize; i++) Tmp[i] = (A[i-1]+A[i+1])/2.0;
    delta = 0.0;
    for (i = 1; i <= mysize; i++) delta +=fabs(A[i]-Tmp[i]);
    MPI_Allreduce(&delta, &sum, 1, MPI_DOUBLE, MPI_SUM,
        MPI_COMM_WORLD);
    delta = sum;
    for (i = 1; i <= mysize; i++) A[i]=Tmp[i];
} while (delta > epsilon);
if (rank == 0) printf("Iterations: %d\n", iters);
MPI_Finalize();
}
```

Courtesy: Larry Snyder et al

*API-based control flow, distribution is hard-coded in program*



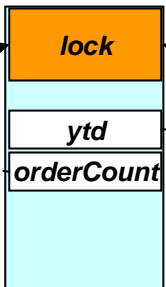
# Example of Atomic Sections

## SPECjbb2000: Java vs. X10 versions

### Java version:

```
public class Stock extends Entity {...
private float ytd;
private short orderCount; ...
public synchronized void
  incrementYTD(short ol_quantity) { ...
    ytd += ol_quantity; ...}...
public synchronized void
  incrementOrderCount() { ...
    ++orderCount; ...} ...
}
```

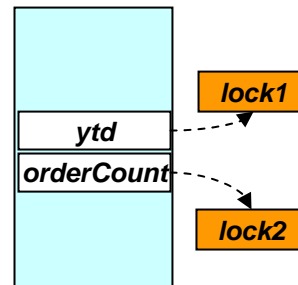
Layout of  
a "Stock"  
object



*These two methods cannot be executed simultaneously because they use the same lock*

### X10 version (w/ atomic section):

```
public class Stock extends Entity {...
private float ytd;
private short orderCount; ...
public atomic void
  incrementYTD(short ol_quantity) { ...
    ytd += ol_quantity; ...}...
public atomic void
  incrementOrderCount() { ...
    ++orderCount; ...} ...
}
```



*With atomic sections, these two methods can be executed simultaneously (can also use transactional memory)*

# Example of Conditional Atomic Sections

## SPECjbb2000: Java vs. X10 versions

### JAVA CODE

```
Main thread (see spec.jbb.Company): ...
// Wait for all threads to start.
synchronized (company.initThreadsStateChange) {
    while ( initThreadsCount != threadCount ) {
        try {
            initThreadsStateChange.wait();
        } catch (InterruptedException e) {...}
    }
}
// Tell everybody it's time for warmups.
mode = RAMP_UP;
synchronized (initThreadsCountMonitor) {
    initThreadsCountMonitor.notifyAll();
} ...
Worker thread (see spec.jbb.TransactionManager): ...
synchronized (company.initThreadsCountMonitor) {
    synchronized (company.initThreadsStateChange) {
        company.initThreadsCount++;
        company.initThreadsStateChange.notify();
    }
    try {
        company.initThreadsCountMonitor.wait();
    } catch (InterruptedException e) {...}
} ...
```

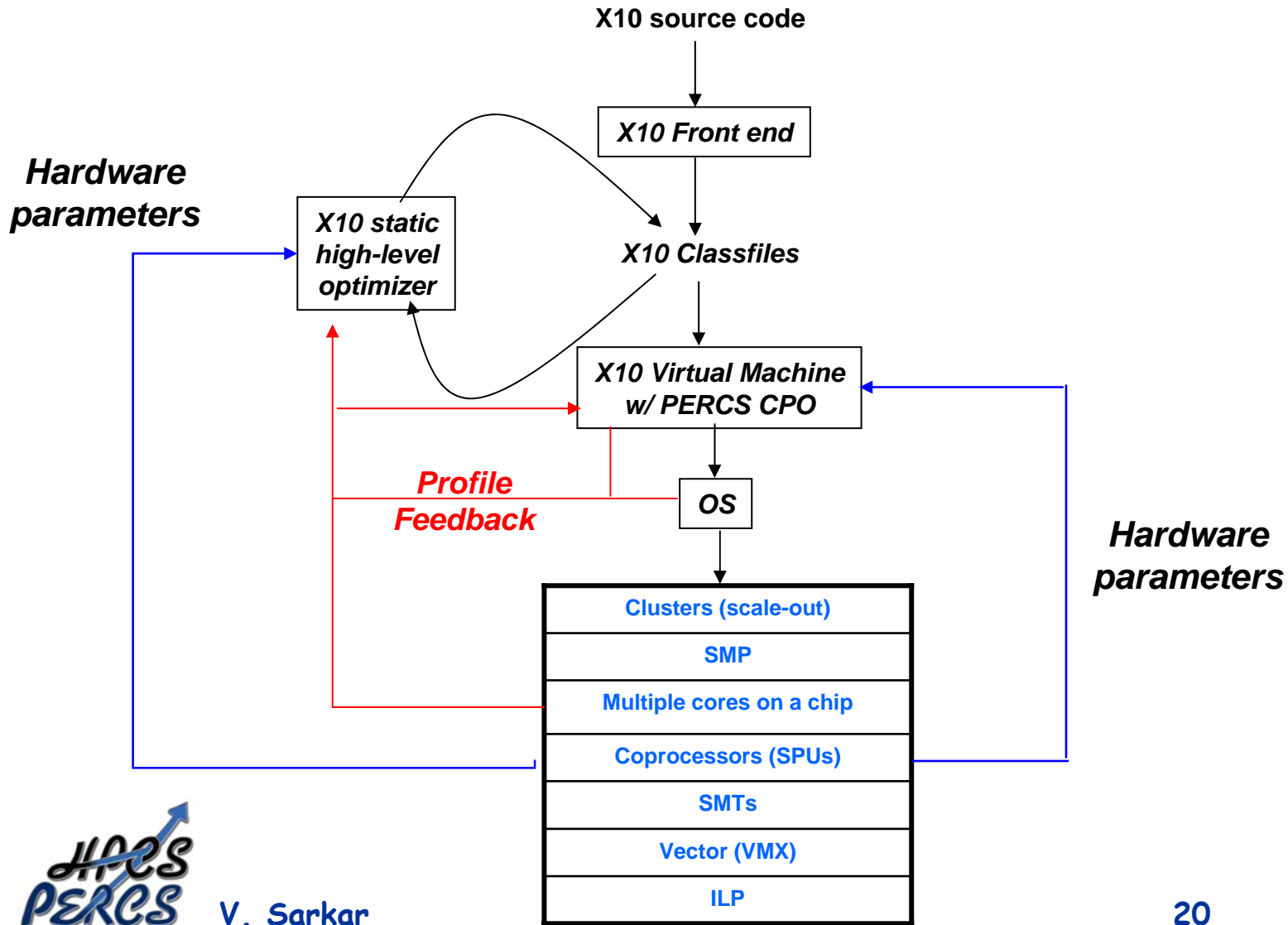
### X10 CODE WITH ATOMIC SECTIONS

```
Main thread: ...
// Wait for all threads to start.
when (company.initThreadsCount==threadCount)
atomic {
    mode = RAMP_UP;
    initThreadsCountReached = true;
} ...

Worker thread: ...
atomic {
    company.initThreadsCount++;
}
await ( initThreadsCountReached ); //barrier synch.
...
```



# Design of X10 compilation and VEE framework



# Relating X10 optimizations to optimizations for past programming paradigms

Programming paradigm	Activities	Storage classes	Important optimizations
Message-passing e.g., MPI	Single activity per place	Place local	Message aggregation, optimization of barriers & reductions
Data parallel e.g., HPF	Single global program	Partitioned global	SPMDization, synchronization & communication optimizations
PGAS e.g., Titanium, UPC	Single activity per place	Partitioned global, place local	Localization, SPMDization, synchronization & communication optimizations
DSM e.g., TreadMarks	Multiple	Partitioned global, activity local	Data layout optimizations, page locality optimizations
NUMA	Single activity per place	Partitioned global, activity local	Data distribution, synchronization & communication optimizations
Co-processor e.g., STI Cell	Single activity per place	Partitioned-global, place-local	Data communication, consistency, & synchronization optimizations
Futures / active messages	Multiple	Place-local, activity local	Message aggregation, synchronization optimization
Full X10	Multiple activities in multiple places	Partitioned-global, place-local, activity-local	All of the above

# Aggregating async's at the X10 level: ArrayCopy example

## Version 1:

```
<value T, D, E> public static void
arrayCopy( T[D] a, T[E] b) {
    // Spawn an activity for each index to
    // fetch and copy the value
    ateach ( i : D.region)
        a[i] = async b[i];
    next c; // Advance clock
}
```

## Version 2:

```
<value T, D, E> public static void
arrayCopy( T[D] a, T[E] b) {
    // Spawn one activity per place
    ateach ( D.places )
        for ( j : D | here )
            a[j] = async b[j];
    next c; // Advance clock
}
```

## Version 3:

```
<value T, D, E> public static void
arrayCopy( T[D] a, T[E] b) {
    // Spawn one activity per D-place and one
    // future per place p to which E maps an
    // index in (D | here).
    ateach ( D.places ) {
        region LocalD = (D | here).region;
        ateach ( p : E[LocalD] ) {
            region RemoteE = (E | p).region;
            region Common =
                LocalD && RemoteE;
            a[Common] = async b[Common];
        }
    }
    next c; // Advance clock
}
```



# X10 Status and Plans

---

- **Draft Language Design Report available internally w/ set of sample programs**
- **Implementation begun on Prototype #1 for 1/05**
  - **Functional reference implementation, not optimized performance**
- **Productivity experiments planned for 7/05**
  - **Compare X10 w/ MPI, OpenMP, Java threads, ...**
- **Prototype #2 scheduled for 12/05**
  - **Focus will be on optimization of parallelism, synchronization and locality**

# Migrating applications in existing programming models to X10

- **OpenMP application**
  - Can be initially implemented as single place w/ one activity per SPMD virtual processor
  - Partition into multiple places for improved performance
- **Multithreaded applications**
  - Can be initially implemented as single place w/ one activity per thread
  - Partition into multiple places for improved performance
- **MPI**
  - Partition into one place per processor
  - Replace message-passing operations by asynchronous operations
- *In all of the above cases, there is a low barrier for the initial migration, which can then be followed by additional tool-guided tuning and transformations*

# X10 Implementation Challenges

- **Type checking/inference** to enforce semantic guarantees
  - Clocked types
  - Place-aware types
- **Consistency management**
  - Lock assignment for atomic sections
  - Data-race detection
- **Activity aggregation**
  - Batch activities into a single thread.
- **Message aggregation**
  - Batch “small” messages.
- **Load-balancing**
  - Dynamic, adaptive migration of place from one processor to another.
- **Continuous optimization**
  - Efficient implementation of scan/reduce
- **Efficient invocation of components in foreign languages**
  - C, Fortran
- **Garbage collection across multiple places**



# Conclusions and Future Work

---

- Large-scale Parallel Systems pose unique challenges to VEEs
- Summarized overall approach in PERCS project, with a focus on the X10 language
- Next steps (2005):
  - Use applications and productivity studies to refine design decisions in X10
  - Prototype solutions to address implementation challenges
- Future work (beyond 2005)
  - Explore integration of X10 with other language efforts in IBM
    - XML (XJ), BPEL, ...
  - Community effort to build consensus on standardized “high productivity” languages for HPC systems in the 2010 timeframe