X10: Addressing Language, Compiler, and Runtime Challenges for Scalable Systems in 2010

Vivek Sarkar

IBM T.J. Watson Research Center

vsarkar@us.ibm.com

LCR 2004 Workshop

Oct 22 - 23, 2004

This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract No. NBCH30390004.

Acknowledgments: PERCS team

- **IBM PERCS Team members**
  - **IBM Research**
  - **IBM Systems & Technology Group**
  - **IBM 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**

- **X10 core team**
  - Philippe Charles
  - Kemal Ebcioglu
  - Christian Grothoff (Purdue)
  - Christoph von Praun
  - Vijay Saraswat
  - Vivek Sarkar

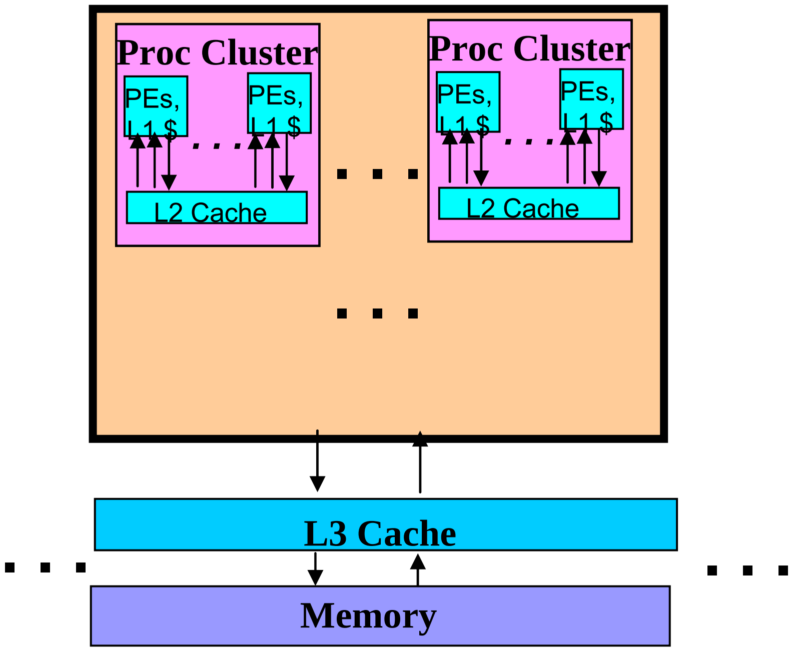
- **Additional contributors to X10 design & implementation ideas:**
  - David Bacon
  - Bob Blainey
  - Perry Cheng
  - Julian Dolby
  - Guang Gao (U Delaware)
  - Allan Kielstra
  - Robert O'Callahan
  - Filip Pizlo (Purdue)
  - V.T.Rajan
  - Lawrence Rauchwerger (Texas A&M)
  - Mandana Vaziri
  - Jan Vitek (Purdue)

Outline

1. Motivation
2. X10 language
3. Compiler challenges and opportunities
4. Runtime system challenges and opportunities
5. Conclusions

Performance and Productivity Challenges facing Future Scalable Systems

1) <u>Memory wall:</u> Severe *non-uniformities* in bandwidth & latency in memory hierarchy



2) <u>Frequency wall:</u> 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
SIMD
ILP

3) <u>Scalability wall:</u> Software will need to deliver ~ 10^5*-way parallelism* to utilize large-scale parallel systems

Impact of Programming Model on Productivity

1. **Safety** – how much of the burden of ensuring absence of errors falls on the user? e.g., Type errors, Initialization errors, Memory errors, Concurrency errors, Consistency errors, …

2. **Portability** – how much effort is required to move the application across multiple platforms and multiple system generations?

3. **Performance** --- how much of the burden of managing and tuning program resources falls on the user?

4. **Integration** --- to what extent can the programming model reuse existing Languages, Environment, Libraries, and Tools?

Example of Productivity Issues: Common errors in MPI communication

- Program error: MPI call w/ incorrect argument
  - Type, destination number …
- Non-unique or invalid message envelope
  - <source, destination, tag, communicator>
- Resource error: program exceeds available system resource
- Coordination error: improper handling of asynchronous calls (analogous to data races)
- Interaction with signals
- Interaction with multithreading/OpenMP

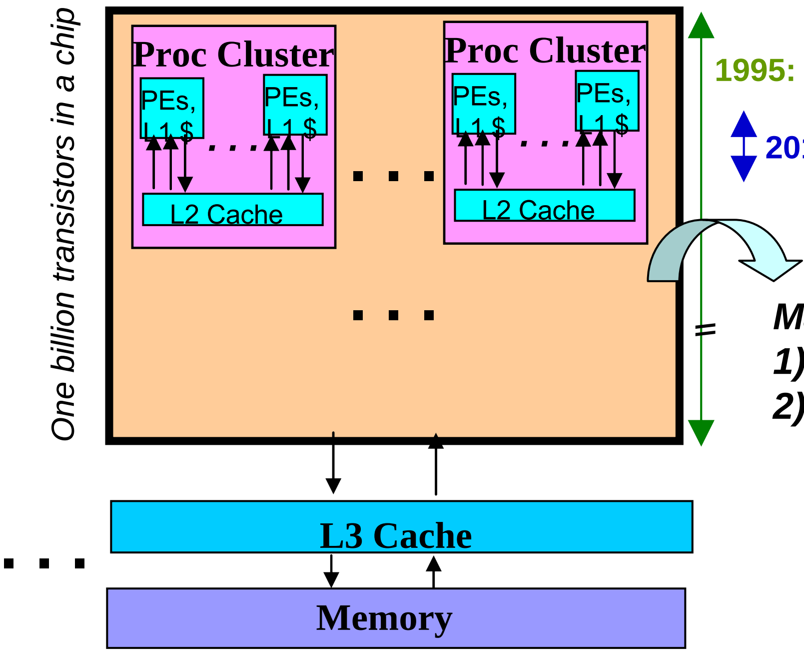
Example of Productivity issues: Compiler-Driven Performance in Current Parallel Programming Models

- **MPI: Local memories + message-passing**
  - **Parallelism, locality, and "global view" are completely managed by programmer**
  - **Communication, synchronization, consistency operations specified at low level of abstraction**
  - ➔ **Limited *opportunities* for compiler optimizations**
- **Java threads, OpenMP: shared-memory parallel programming model**
  - **Uniform symmetric view of all shared data**
  - **Non-transparent performance --- programmer cannot manage data locality and thread affinity at different hierarchy levels (cluster, SMT, …)**
  - ➔ **Limited *effectiveness* of compiler optimizations**
- **HPF, UPC: partitioned global address space + SPMD execution model**
  - **User specifies data distribution & parallelism, compiler generates communications using owner-computes rule**
  - **Large overheads in accessing shared data; compiler optimizations can help applications with simple data access patterns**
  - ➔ **Limited *applicability* of compiler optimizations**

Development Productivity: Grand Challenge from DARPA High Productivity Computing Systems (HPCS) program

- *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. **Programming complexity**
  2. **Expertise gap**

Obstacle #1 (Programming Complexity) --- High Complexity of HPC Systems Limits HPC Application Development Productivity



1995: entire chip can be accessed in 1 cycle

2010: only small fraction of chip can be accessed in 1 cycle

Major sources of complexity for application developer:
1) Severe non-uniformities in data accesses
2) Applications must exhibit large degrees of parallelism (up to ~ 10^5 threads)

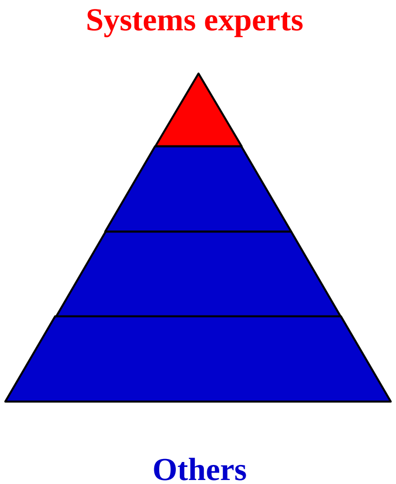
Complexity leads to increases in all phases of HPC Software Lifecycle related to parallel code

Requirements	Input Data	Algorithm Development	Written Specification	Source Code	Parallel Specification	Development of Parallel Source Code --- Design, Code, Test, Port, Scale, Optimize	Production Runs of Parallel Code	Maintenance and Porting of Parallel Code

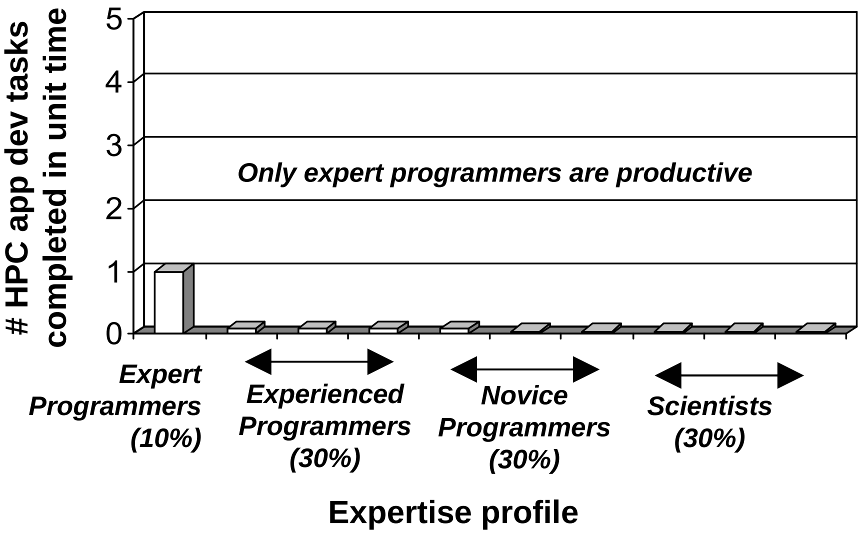
HPC Software Lifecycle

Obstacle #2 (Expertise Gap) --- Low Availability of Expert System Programmers who can develop/scale production HPC apps

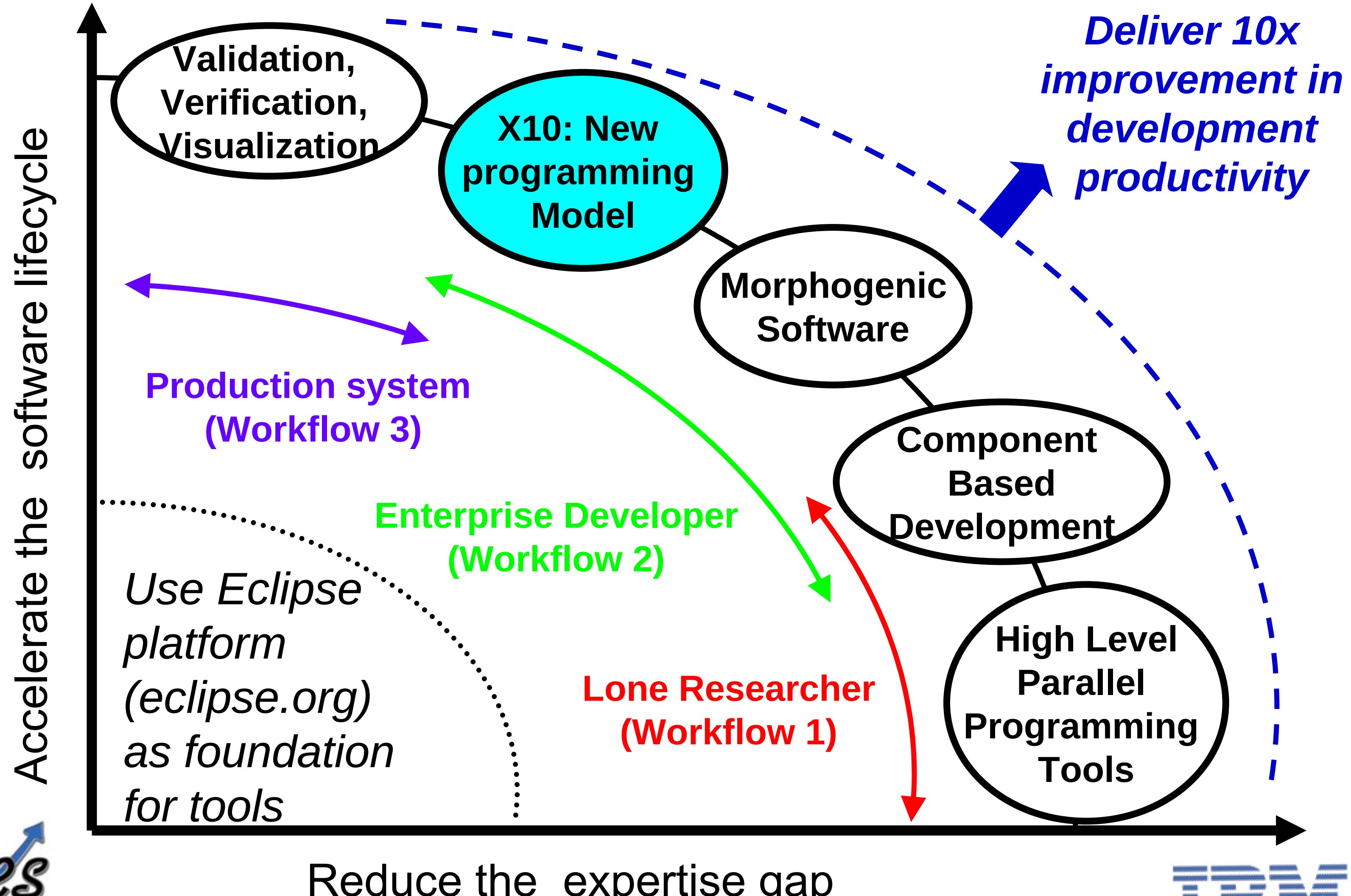
- **Two classes of programming skills:**
  - **Expert knowledge of concurrent programming**
    - top-gun knowledge of system s/w and h/w
    - find today's programming models time-consuming to use
  - **Others**
    - Scientists
    - Domain experts
    - . . .



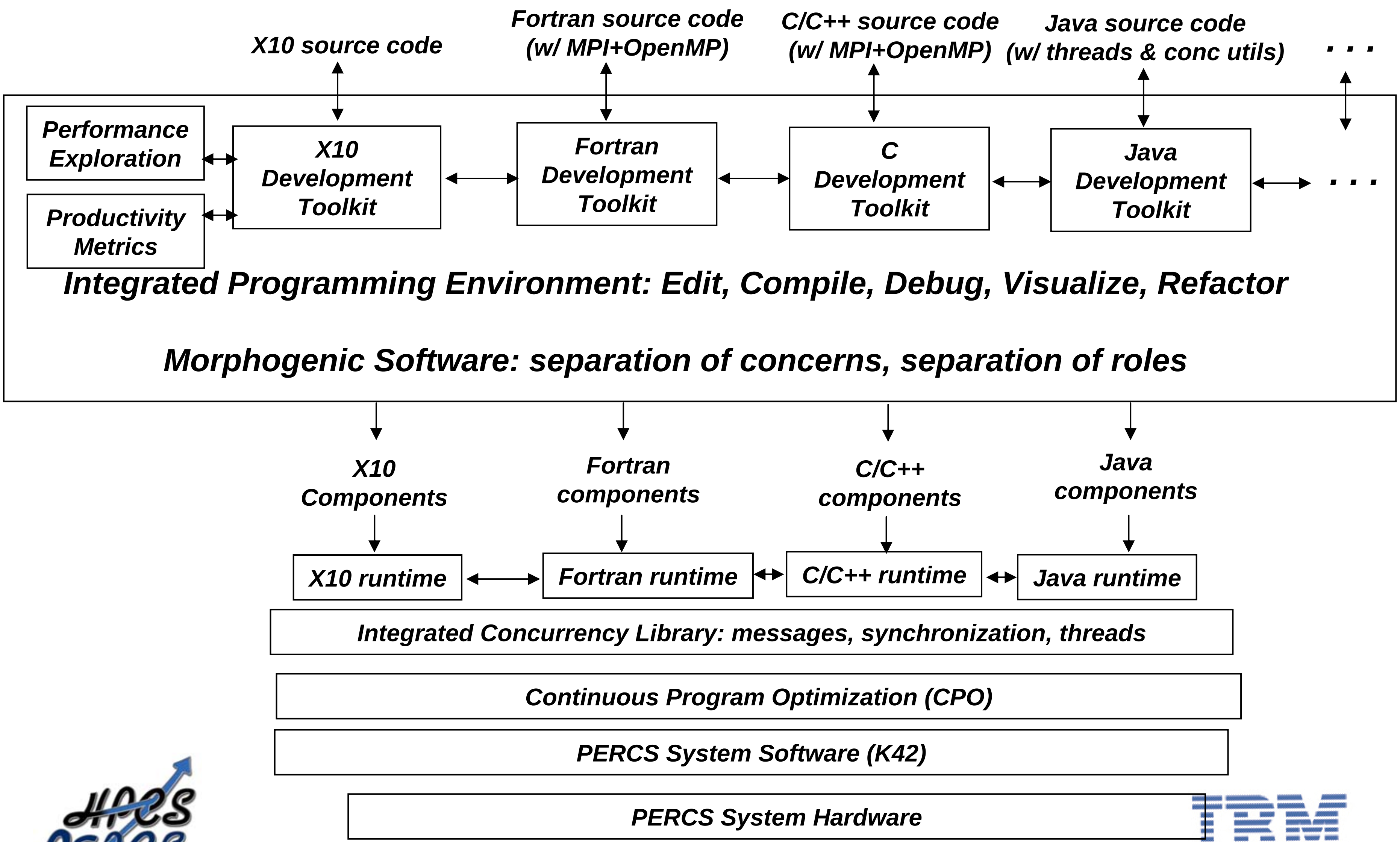
Application development productivity on current HPC systems:



IBM PERCS project: Programming Model and Tools



X10 Programming and Runtime Environments



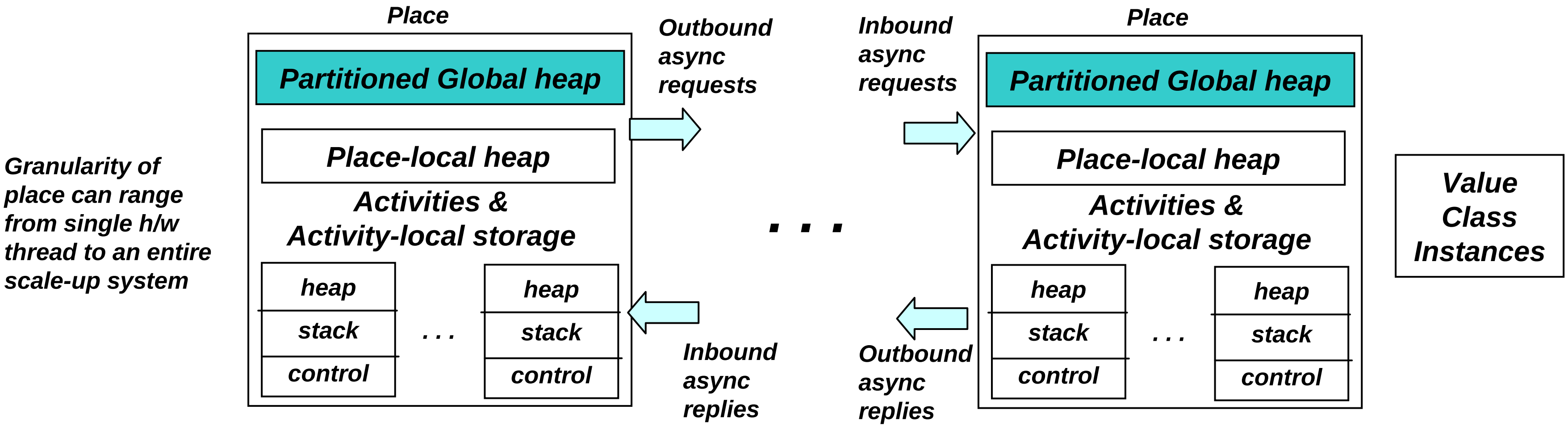
Outline

1. Motivation
2. X10 language
3. Compiler challenges and opportunities
4. Runtime system challenges and opportunities
5. Conclusions

X10 Design Guidelines: Design for Productivity & Compiler/Runtime-driven Performance

- **Start with state-of-the-art OO language primitives as foundation**
  - No gratuitous changes
  - Build on existing skills
- **Raise level of abstraction for constructs that should be amenable to optimized implementation**
  - Monitors → atomic sections
  - Threads → async activities
  - 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
  - Simple concurrency model & memory model
- **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
    - Can be copied/migrated freely
- Activities can be created by
  - *async statements* (one-way msgs)
  - *future expressions*
  - *foreach & ateach constructs*
- Activities are coordinated by
  - *Unconditional atomic sections*
  - *Conditional atomic sections*
  - *Clocks* (generalization of barriers)
  - *Force* (for result of future)

X10, in comparison with Java…

- Removes
  - Primitive arithmetic data types
  - Threads, lock-level synchronization
  - Single global heap
  - Arrays
  - JNI
- Adds
  - User-defined value types
  - Asynchronous activities, with atomic sections
  - Places specifying affinity between data and computation
  - True, distributed, multi-dimensional arrays
  - New efficient native code invocation mechanisms

X10, in comparison with MPI+OpenMP …

- Removes
  - Processes
  - Programmer-managed global data structures
  - Message passing w/ programmer-managed marshalling
    - Includes reductions
  - Low-level message envelopes
    - <source, destination, tag, communicator>
  - Barriers
  - OpenMP threads
  - Locks, critical sections
  - Affinity directives
  - INDEPENDENT directive

- Adds
  - Places
  - Partitioned Global Address Space
  - Asynchronous activities w/ objects and futures
    - Includes reductions
  - Strongly-typed invocations and return values (futures)
  - Clocks
  - Asynchronous activities
  - Atomic sections
  - “at” clauses
  - foreach, ateach statements

Async activities: unified abstraction of threads and messages

- **Async statement (active message)**
  - `async(P){S}:` run S at place P
  - `async(D){S}:` run S at place containing datum **D**
  - S may contain local atomic operations or additional async activities for same/different places.
- *Example:*

```
public void put(K key, V value) {
  int hash = key.hashCode()% D.size;
  async (D[hash]) {
       for (_ b = buckets[hash]; b != null; b = b.next) {
         if (b.k.equals(key)) {
           b.v = value;
           return;
         }
       }
       buckets[hash] =
         new Bucket<K,V>(key, value, buckets[hash]);
  };
}
```

- **Async expression (future)**
  - `F = future(P){E}`, **or** `F = future(D){E}`: Return the value of expression E, evaluated in place P (or the place containing datum D)
  - `force F` **or** `!F`: suspend until value is known
- *Example:*

```
public ^V get(K key) {
    int hash = key.hashCode()% D.size;
    return future (D[hash]) {
      for (_ b = buckets[hash]; b != null; b = b.next) {
        if (b.k.equals(key)) {
          return b.v;
        }
      }
      return new V();
    }
  }
```

Distributed hash-table example

Clocks: abstraction of barriers

- **Operations:**

  ```
  clock c = new clock();
  ```

  ```
  now(c){S}
  ```

  - Require S to terminate before clock can progress.

  ```
  continue c;
  ```

  - Signals completion of work by activity in this clock phase.

  next $c_1,\ldots,c_n$;

  - Suspend until clocks can advance. Implicitly continues all clocks. $c_1,\ldots,c_n$ names all clocks for activity.

  ```
  drop c;
  ```

  - No further operations on c..

- **Semantics**
  - Clock c can advance only when all activities registered with the clock have executed continue c..
- **Clocked final**
  - clocked(c) final int l = r;
  - Variable is "final" (immutable) until next phase

RandomAccess (GUPS) example

```
public void run(int a[] blocked, int seed[] cyclic,
            int value smallTable[]) {
    ateach (start : seed) clock(c) {
       int ran = start;
       for (int count : 1.. N_UPDATES/place.MAX_PLACES) {
           ran = Math.random(ran);
           int j = F(ran); // function F() can be in C/Fortran
           int k = smallTable[g(ran)];
           async (a[j]) atomic {a[j]^=k;}
       } // for
    } // ateach
   next c;
}
```

Regions and Distributions

- **Regions**
  - The domain of some array; a collection of array indices
  - region R = [0..99];
  - region R2 = [0..99,0..199];
- **Region operators**
  - region Intersect = R3 && R4;
  - region Union = R3 || R4;
  - Etc.
- **Distributions**
  - Map region elements to places
    - distribution D = cyclic(R);
  - Domain and range restriction:
    - distribution D2 = D | R;
    - distribution D3 = D | P;
- Regions/Distributions can be used like type and place parameters
  - <region R, distribution D> void m(...)

ArrayCopy example: example of high-level optimizations of async activities

Version 1 (orginal):

```
<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 (optimized):

```
<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[i] = async b[i];
     next c; // Advance clock
   }
```

Version 3 (further optimized):

```
<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
   }
```

Uniform treatment of Arrays & Loops and Collections & Iterators

- **Arrays**
  - Map region elements to values (therefore multidimensional)
  - Declared with a given distribution
  - int[D] array;
- **Loops**
  - ateach (D[R]) { ... }
  - ateach (array) { ... }
  - foreach (i : R) { ... }
  - foreach (i : D) { ... }
  - foreach (i : array) { ... }
  - sequential variants of foreach are available as for loops
- **Distributed Collections**
  - Map collection elements to places
  - Collection<D,E> identifies a collection with distribution D and element type E
- **Parallel iterators**
  - foreach (e : C) { … }
  - ateach (C) { … here … }
- **Sequential iterator**
  - for (e : C)

Unstructured Mesh Transport Example (UMT2K)

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

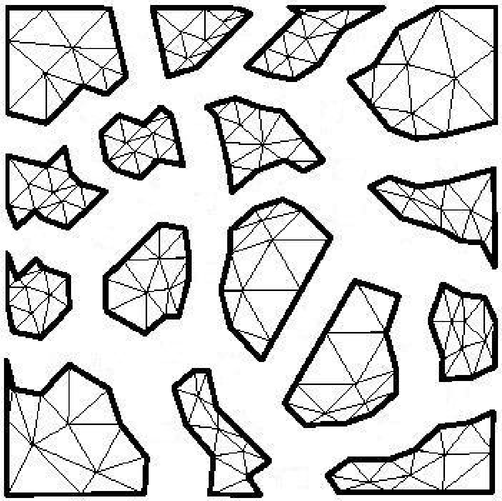


Figure source: Modified from Mathis and Kerbyson, IPDPS 2004

Communication Structure

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

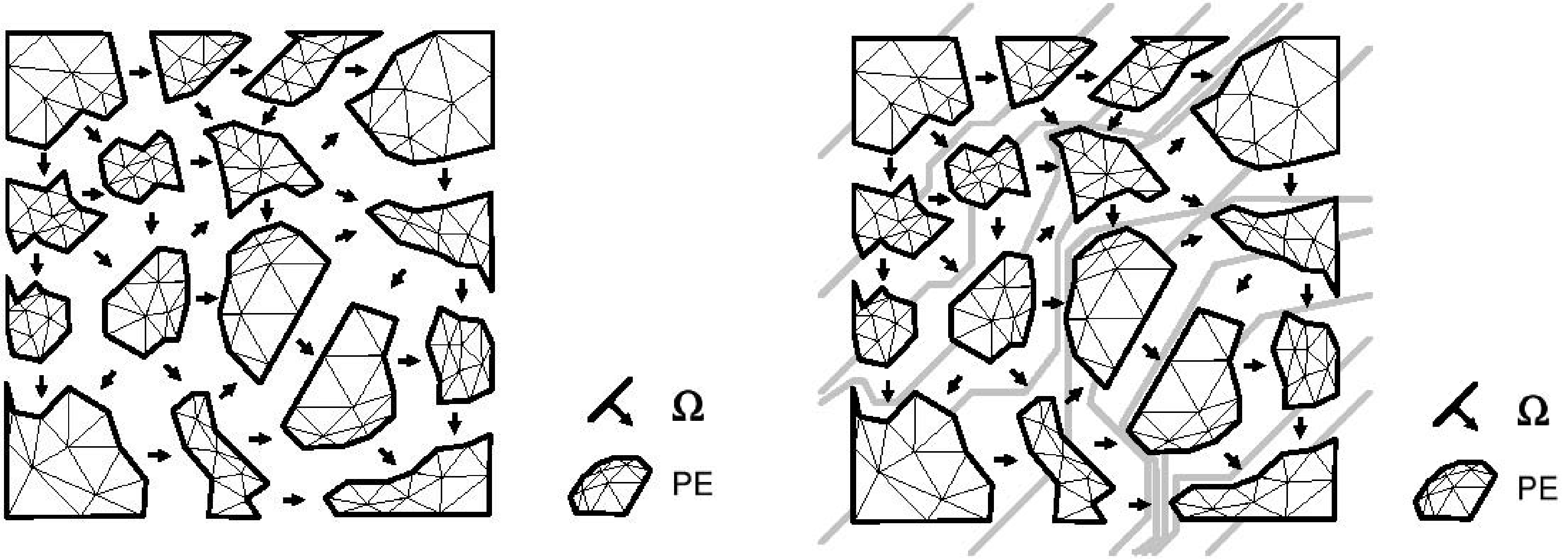


Figure source: Modified from Mathis and Kerbyson, IPDPS 2004

UMT2k in X10: example of hierarchical heterogeneous parallelism

```
do {
  now ( c ) {
    ateach ( n : nodes ) {  // Cluster-level parallelism

        foreach ( s : Sweeps ) clock(d)    { // SMP parallelism
              // receive inputs
              flows = new Flux[R] (k) { // SMT parallelism
                  async (…) inputs[s][k].receive();
              }
              // implicit force for each element of array constructor
             // 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

Reduction and Scan Operators

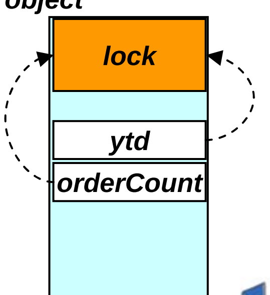
- Reduction operator over type T
  - Static method with signature: T(T,T)
  - Virtual method in class T with signature T(T)
  - Operator is expected to be associative and commutative
- Reduction operation: A >> foo() returns value of type T, where
  - A is an array over base type T
  - A>>foo() performs reductions over all elements of A to obtain a single result of type T
- Scan operation: A || foo() returns array, B, of base type T, where
  - B[i] = A[0..i]>>foo()

Example of Unconditional 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; …} …
}
```

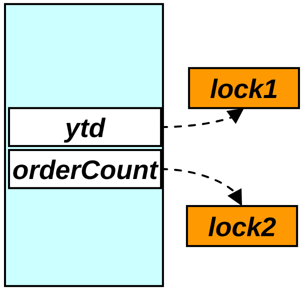
Layout of a "Stock" object



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, X10 implementation can choose to execute these two methods in parallel

Atomic Sections are deadlock-free!

Example of Conditional Atomic Section

- **Conditional Atomic Sections are similar to Conditional Critical Regions (CCRs)**
  - **Powerful construct, misuse can lead to deadlock**
  - **Need to identify special cases that are most useful in practice**

```
class OneBuffer<value T> {
   ?Box<T> datum = null;
   public void send(T v) {
       when (this.datum == null) {
          this.datum := new Box<T>(datum);
        }
   }
   public T receive() {
      when (this.datum !=null) {
         T v = datum.datum;
         value := null;
         return v;
      }
   }
}
```

Outline

1. Motivation
2. X10 language
3. <u>Compiler challenges and opportunities</u>
   - Focus topic: opportunities for exploiting immutability properties
4. Runtime system challenges and opportunities
5. Conclusions

X10 Type System: Features relevant to Compiler Optimization

- Unified type system
  - All data items are objects
- Value classes and clocked final
  - Immutable --- no updatable fields
- Type parameters
  - Places, distributions,
- Nullable
  - All types are non-null by default, need to explicitly declare a variable as nullable
  - For any type T, the type ?T (read: “nullable T”) contains all the values of type T, and a special null value, unless T already contains null.
- Support for both rectangular multidimensional arrays (matrices) and nested arrays

Migrating Applications 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

Relating optimizations for past programming paradigms to X10 optimizations

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

Compiler Framework for Optimization of Explicitly Parallel X10 programs

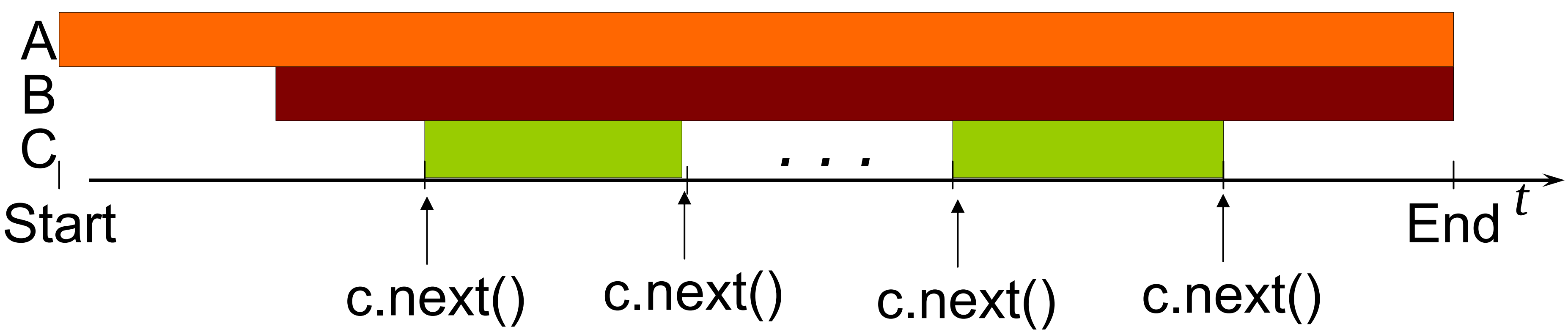
- Parallel Program Graph representation
  - Generalization of program dependence graph
  - Supports optimization of explicitly parallel programs
  - Assume weak memory model (Location Consistency)
- Array SSA Form
  - Generalization of scalar SSA form to arrays and objects
  - Use as foundation for data flow analysis at array/object level
- Continuous Program Optimization (CPO)
  - Extend scope of adaptive on-line optimizations to whole program spanning multiple places

Motivation for Value Classes & Clocked Final in X10: identifying immutable locations

- Immutability information can be used to enhance:
  - Load elimination and register allocation
    - Load of immutable value cannot be changed across a procedure call
  - Array dependence analysis, pointer alias analysis
    - Target of a store instruction cannot be aliased with target of a load instruction
  - Value Numbering / CSE / PRE
    - Load of an immutable value can be treated similarly to read of an unmodified local variable to enable optimization of derived expressions (including null pointer, type checks, array bounds checks)
  - Data transformations
    - object inlining, splitting, replication, caching
  - Parallelization
    - Immutable locations cannot interfere with parallelization
    - No consistency operations need to be performed on immutable locations

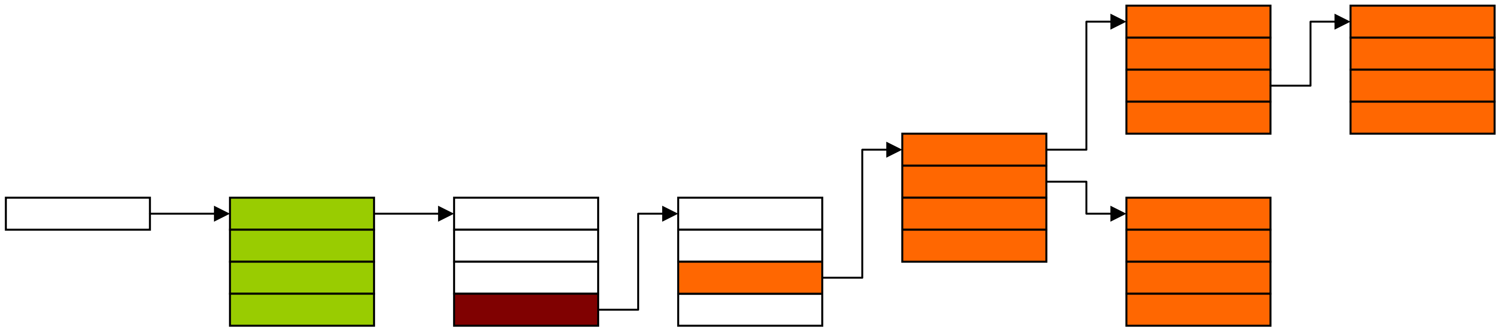
“Lifetime” Dimensions of Immutability in X10

- whole program (config value class)
- after object has been initialized (value class)
- between two next() operations on a clock (clocked final)



Reachability Dimensions of Immutability in X10

- single reference (= `final`)
- single object (shallow immutability)
- full reachability (deep immutability)



Immutability in existing programs: Limit Study

- **Define *Immutability Ratio* as**

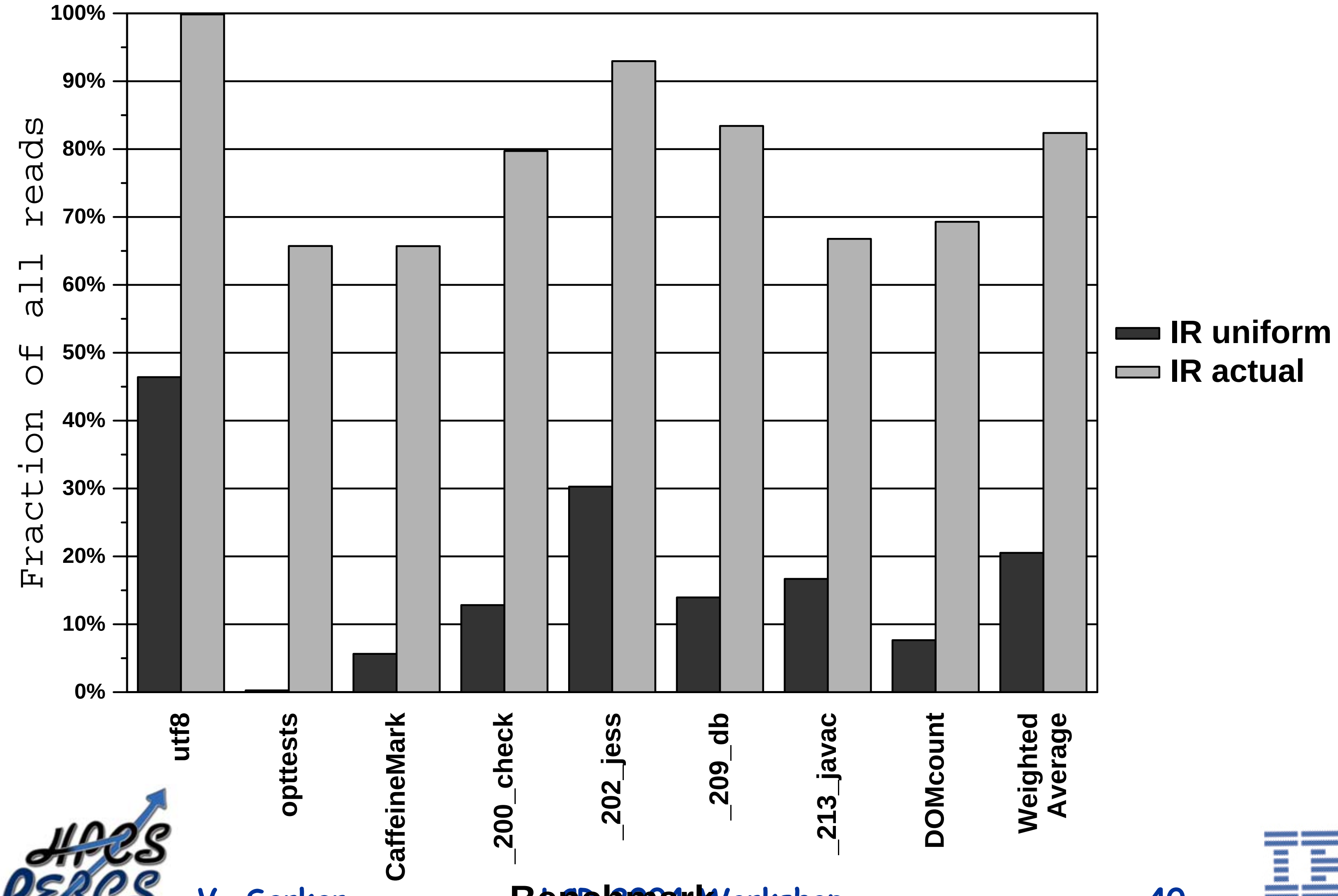
$$IR = \frac{\text{\# of read operations after last write}}{\text{total \# of read operations}}$$

- **IR actual**
  - **Obtained by counting last write separately for each dynamic object instance**
- **IR uniform**
  - **Obtained by assuming that writes are uniformly distributed among reads**
    - **Hypothetical "expected" value of IR**

Limit Study: Experimental Setup

- Instrument Jikes RVM to generate traces
  - all read and write accesses
- Benchmarks
  - Jikes RVM regression tests
    - bytecodeTests, reflect, threads, utf8, opttests
  - CaffeineMark
  - SPECjvm98 (input size = 10%)
    - _200_check, _202_jess, _209_db, _213_javac
  - Xerces (DomCount)
- Goal: measure *Immutability Ratio* for benchmarks

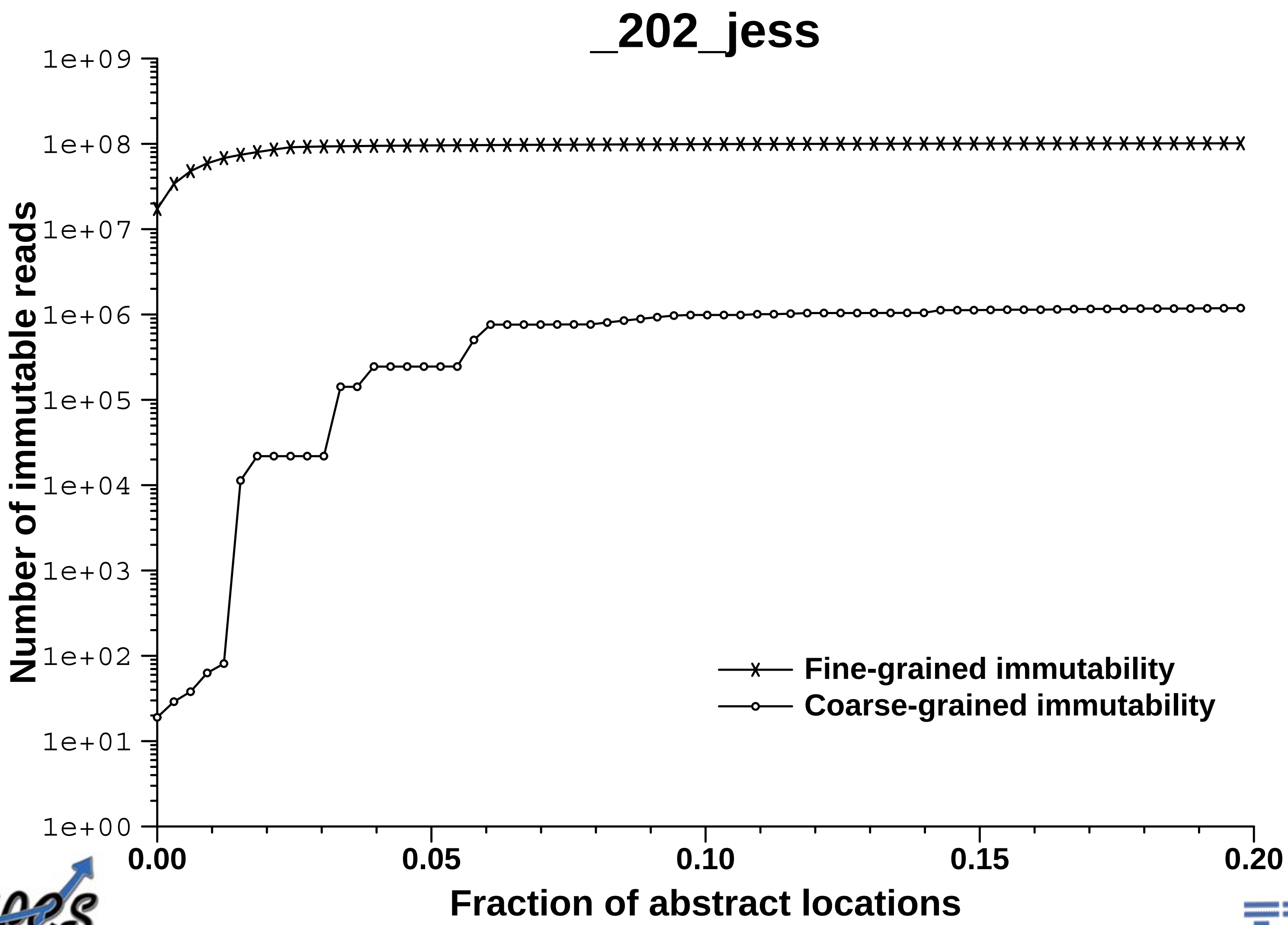
Immutability Ratios



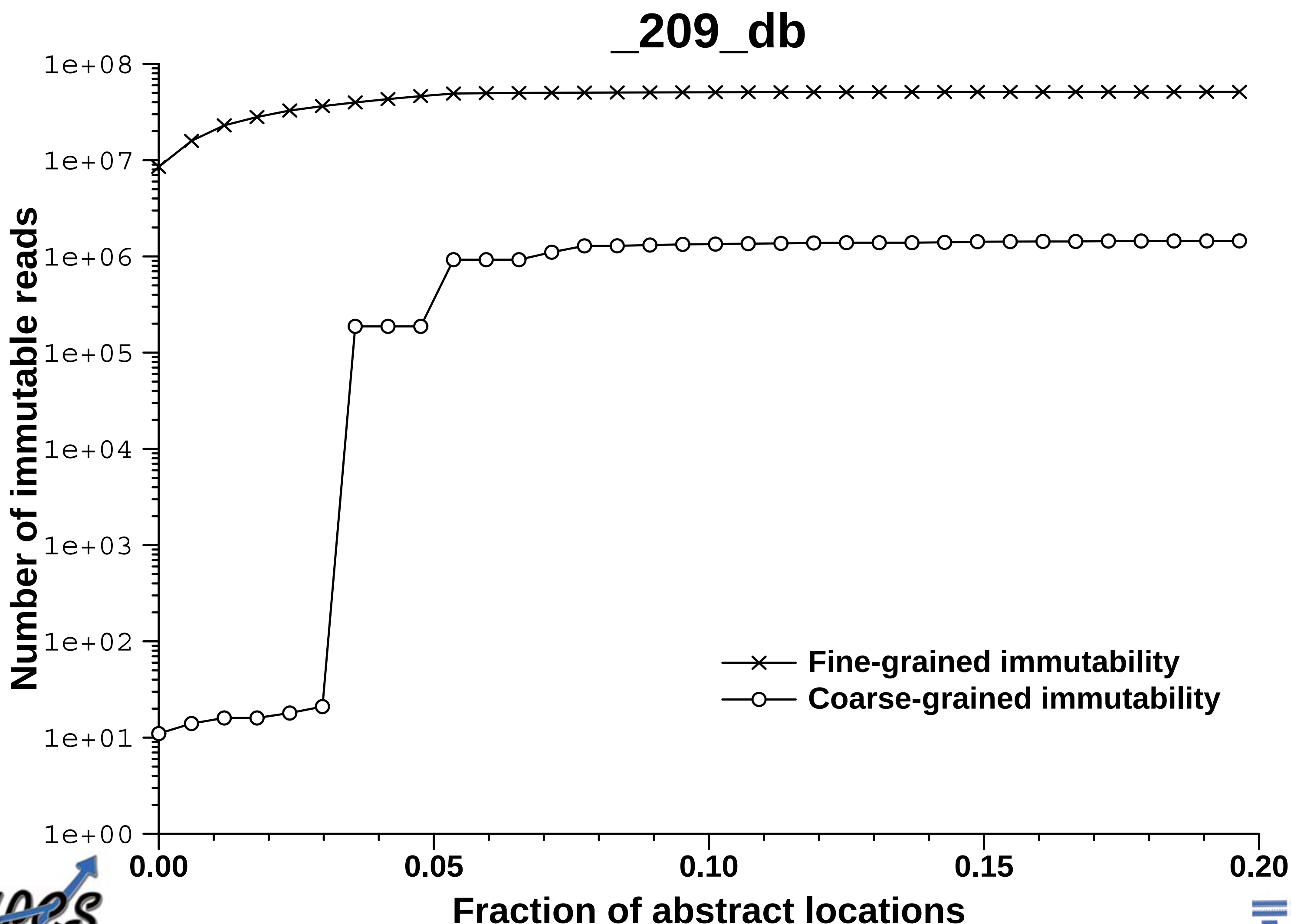
Limit Study: Abstract Locations

- **Abstract location = static representative for set of dynamic locations**
  - **Each declared *field* is a distinct abstract location**
  - **Each declared *array type* is a distinct abstract location**
- **Coarse-grained immutability: measured by merging all dynamic instances of the same abstract location**
- **Goals:**
  - **Measure gap between fine-grained and coarse-grained immutability**
  - **Determine how immutable reads are distributed across abstract locations**

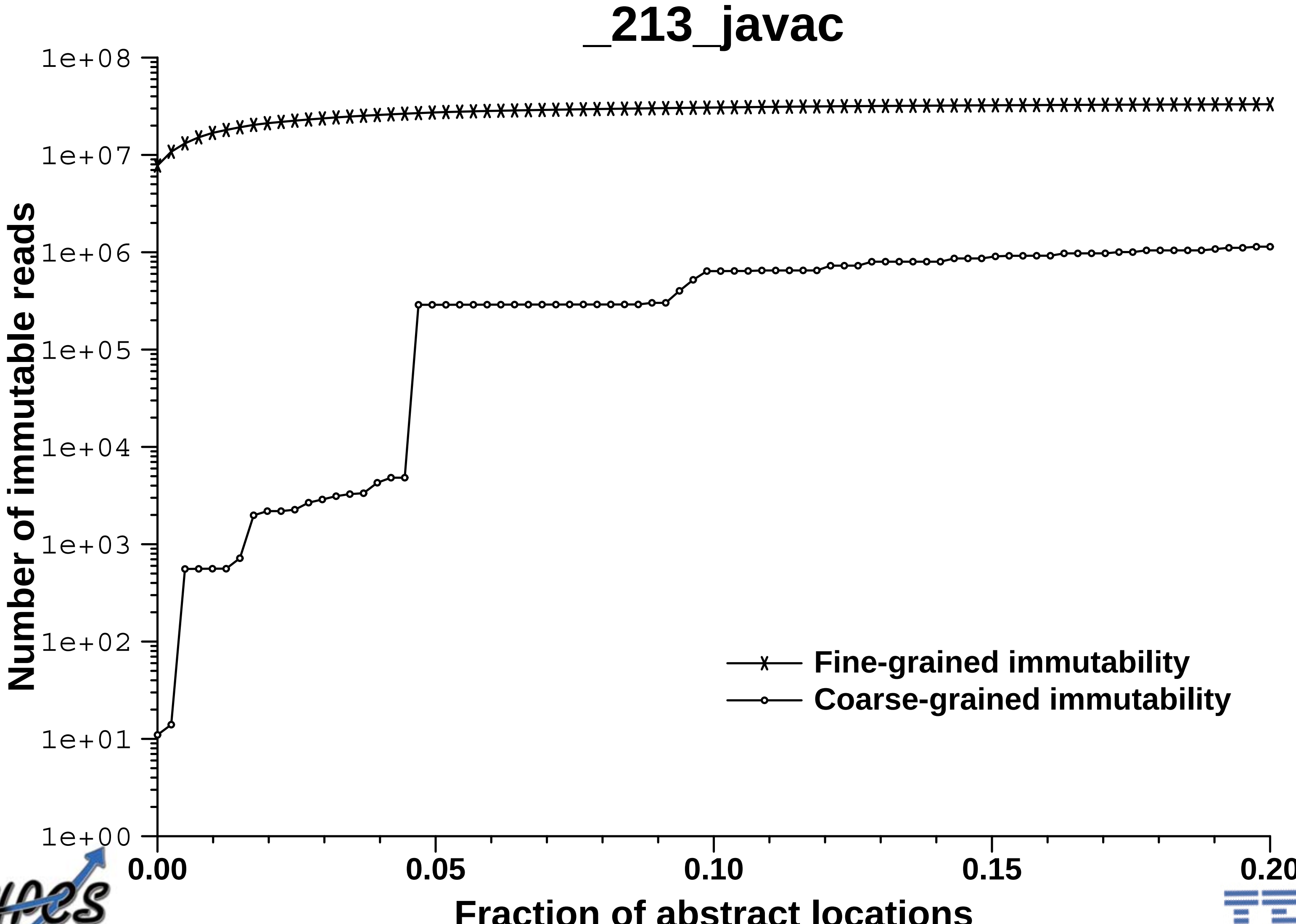
Distribution of immutable reads across abstract locations: _202_jess



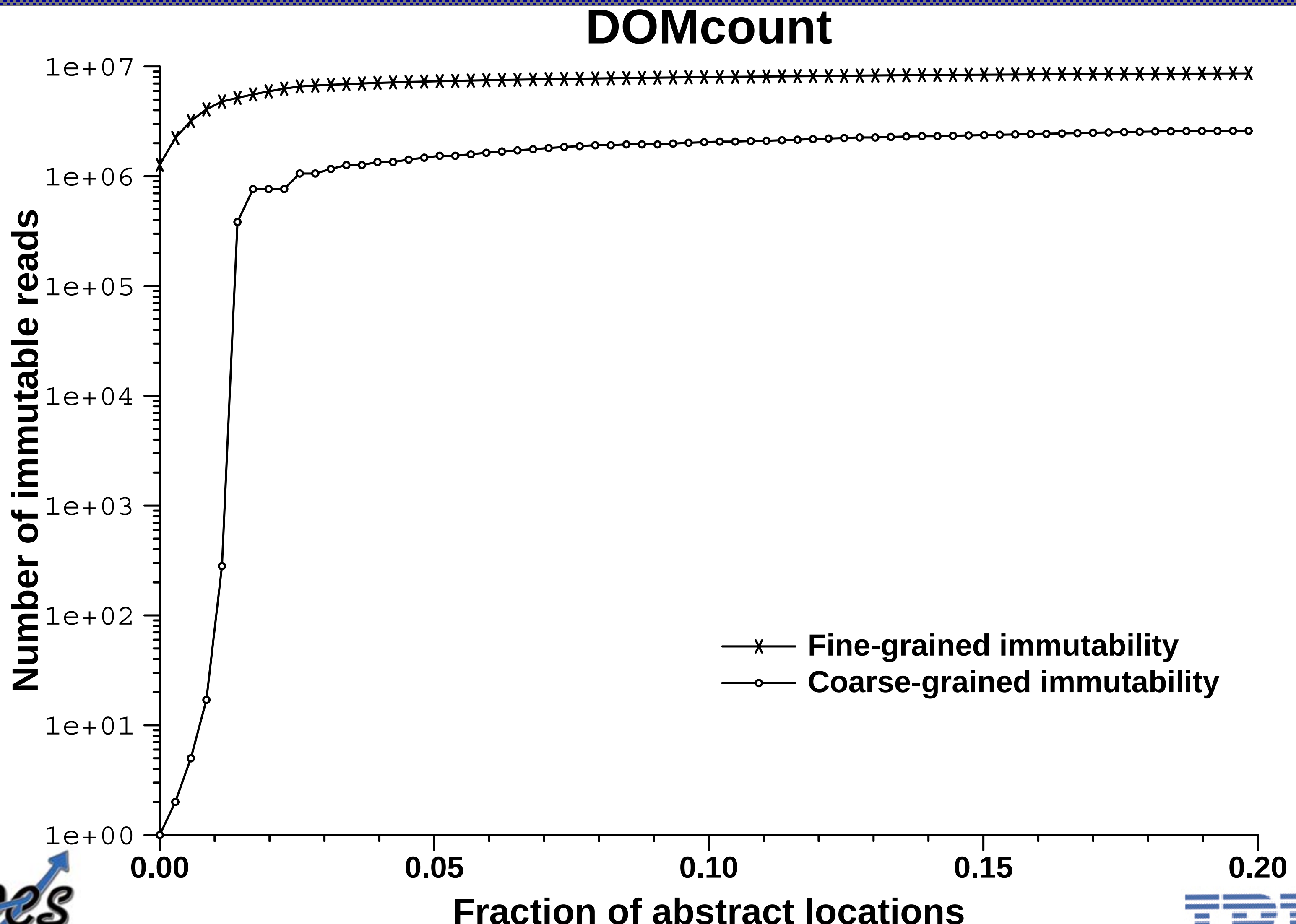
Distribution of immutable reads across abstract locations: _209_db



Distribution of immutable reads across abstract locations: _213_javac



Distribution of immutable reads across abstract locations: DOMcount



Outline

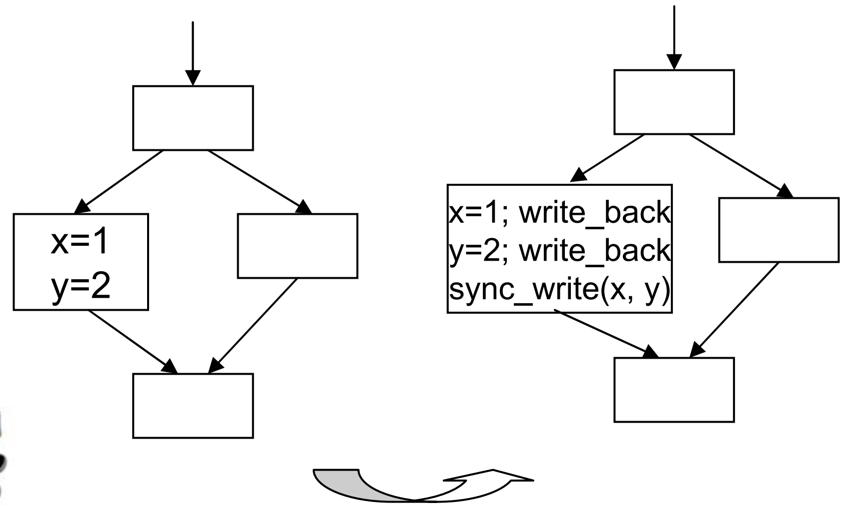
1. Motivation
2. X10 language
3. Compiler challenges and opportunities
4. **Runtime system challenges and opportunities**
   - **Focus topic: continuous program optimization**
5. Conclusions

X10 Runtime design issues

- Places
  - Typically, map one place per SMP node
  - Scenarios where multiple places/node could be useful
    - Virtual partitions
    - Hierarchical places
- Local async/future operations
  - Similar to lightweight threads
- Remote async/future operations
  - Similar to active messages
  - Runtime system needs to marshall/unmarshall parameters and return values
- Possible implementation strategies for atomic sections
  - Only execute one atomic section at a time in a place
  - Analyzable atomic setcions
  - Transactional semantics

Analyzable Atomic Sections

- Definition: An atomic section is fully analyzable if the addresses of all shared locations that it accesses are computable on entry to the section
- Lock assignment problem: assign a set of locks to each atomic section so as to guarantee mutual exclusion and avoid deadlock
- Consistency optimization problem: PRE of consistency operations (refresh, writeback) necessary to support memory consistency of data accesses in an atomic section



"Analyzable Atomic Sections: Integrating Fine Grained Synchronization and Weak Consistency Models for Scalable Parallelism", V.Sarkar, G.Gao, U. Delaware CAPSL Technical Memo 52, Feb 2004.

Memory Model

- X10 focus is on data-race-free applications
- Programmer uses atomic / clock / force operations to avoid data races
  - X10 programming environment also includes data race detection tool
- Weak memory model for defining consistency of unsynchronized accesses
  - Based on Location Consistency memory model
  - Akin to weak ordering guarantees of messages in MPI

X10 Managed Runtime

Benefits of managed runtime systems and virtual machines are well understood …

- Safety
- Productivity
- Portability
- Interoperability
- Isolation
- Virtualization

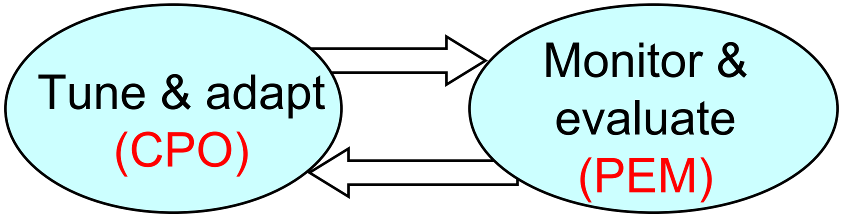
… but, are managed runtime systems appropriate for addressing performance challenges facing future large-scale parallel systems?

➔ Yes, because they enable *continuous program optimization*

Continuous Program Optimization (CPO) through Performance & Environment Monitoring (PEM)

- Continuous Program Optimization (CPO) increases programmer productivity by automating the laborious and challenging performance tuning effort
- CPO aims at tuning application by optimally
  - **adapting the application to its behavior and environment**
  - **adapting environment resources to application behavior**
- CPO is made possible through continuous whole-system Performance and Environment Monitoring (PEM)

CPO/PEM contact:
Evelyn Duesterwald, IBM



Continuous optimization loop

A Catalogue of CPO Scenarios

Adapting applications to execution behavior and environment

— Library Tuning

— Adaptive MPI tuning

— Tuning Just-In-Time (JIT) compiler heuristics

— Interactive tuning environment to support algorithmic tuning

— Adpative mapping of X10 places to physical nodes

Adapting the execution environment to the application's needs:

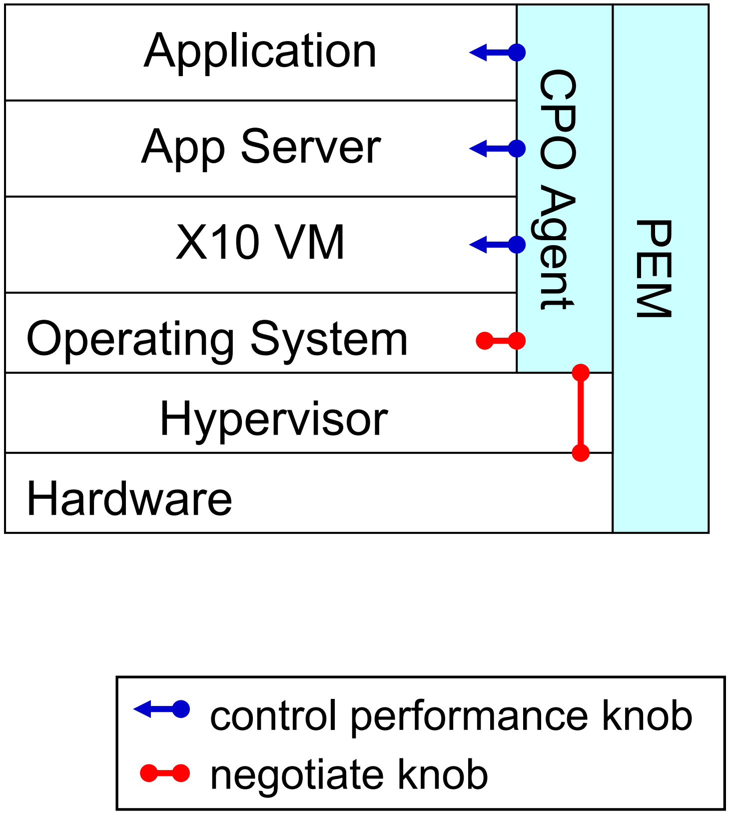
— Tuning the Virtual Machine (heap size, GC policies)

— Tuning the operating environment/OS (adaptive page size, adaptive file cache management (FCM))

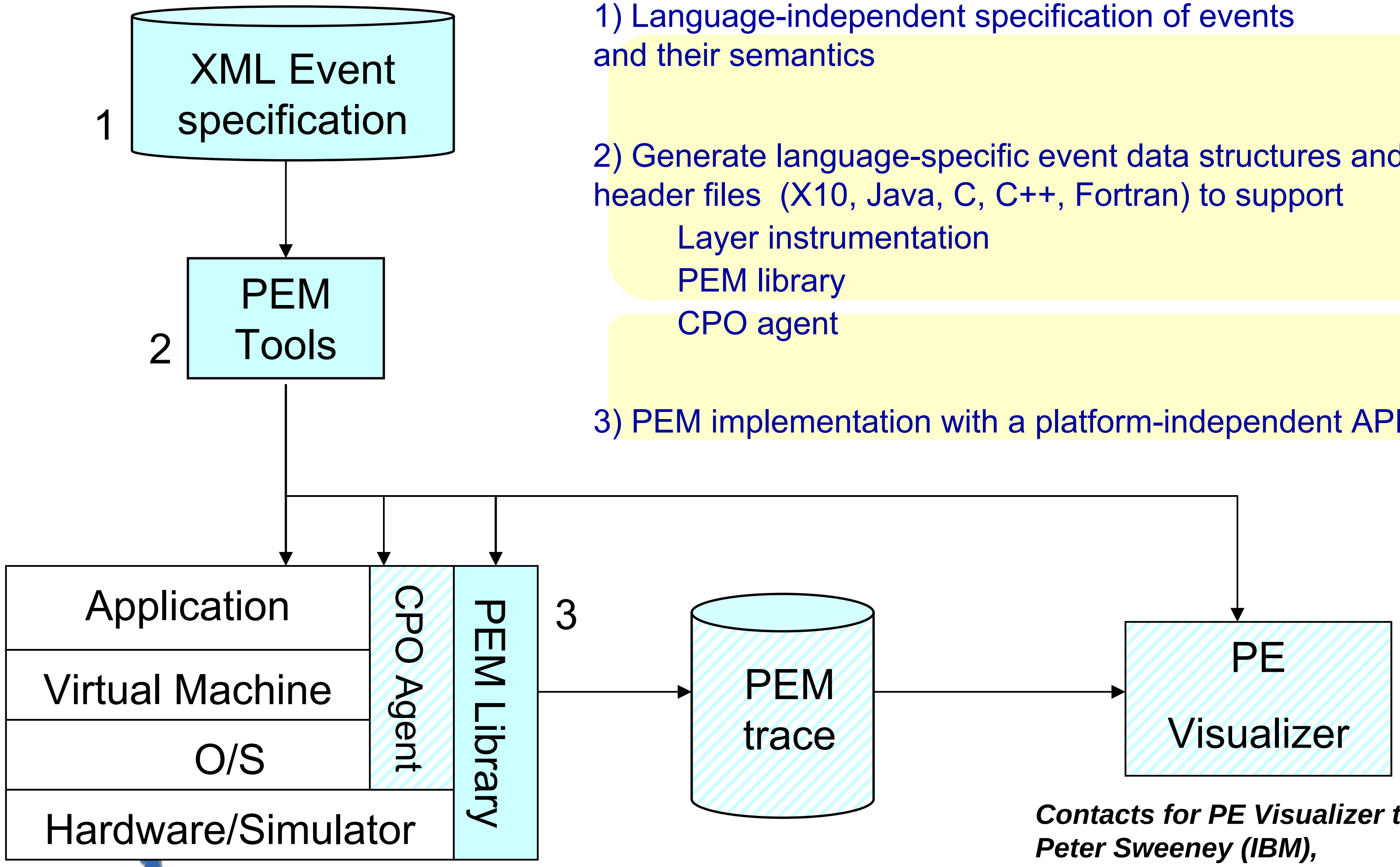
CPO Architecture

2 Components to implement CPO scenarios:

- **PEM: Vertical Performance and Environment Monitoring**
  - **Implements programmable monitoring functionality across execution layers**
  - **Provides integrated whole system view**
- **CPO Agent** (Specific to CPO scenario)
  - **programs PEM to provide desired monitoring information**
  - **Implements specific scenario optimization**
- **There may be several CPO Agents active in the system, interconnected through PEM**



PEM Infrastructure



1) Language-independent specification of events and their semantics

2) Generate language-specific event data structures and header files (X10, Java, C, C++, Fortran) to support
- Layer instrumentation
- PEM library
- CPO agent

3) PEM implementation with a platform-independent API

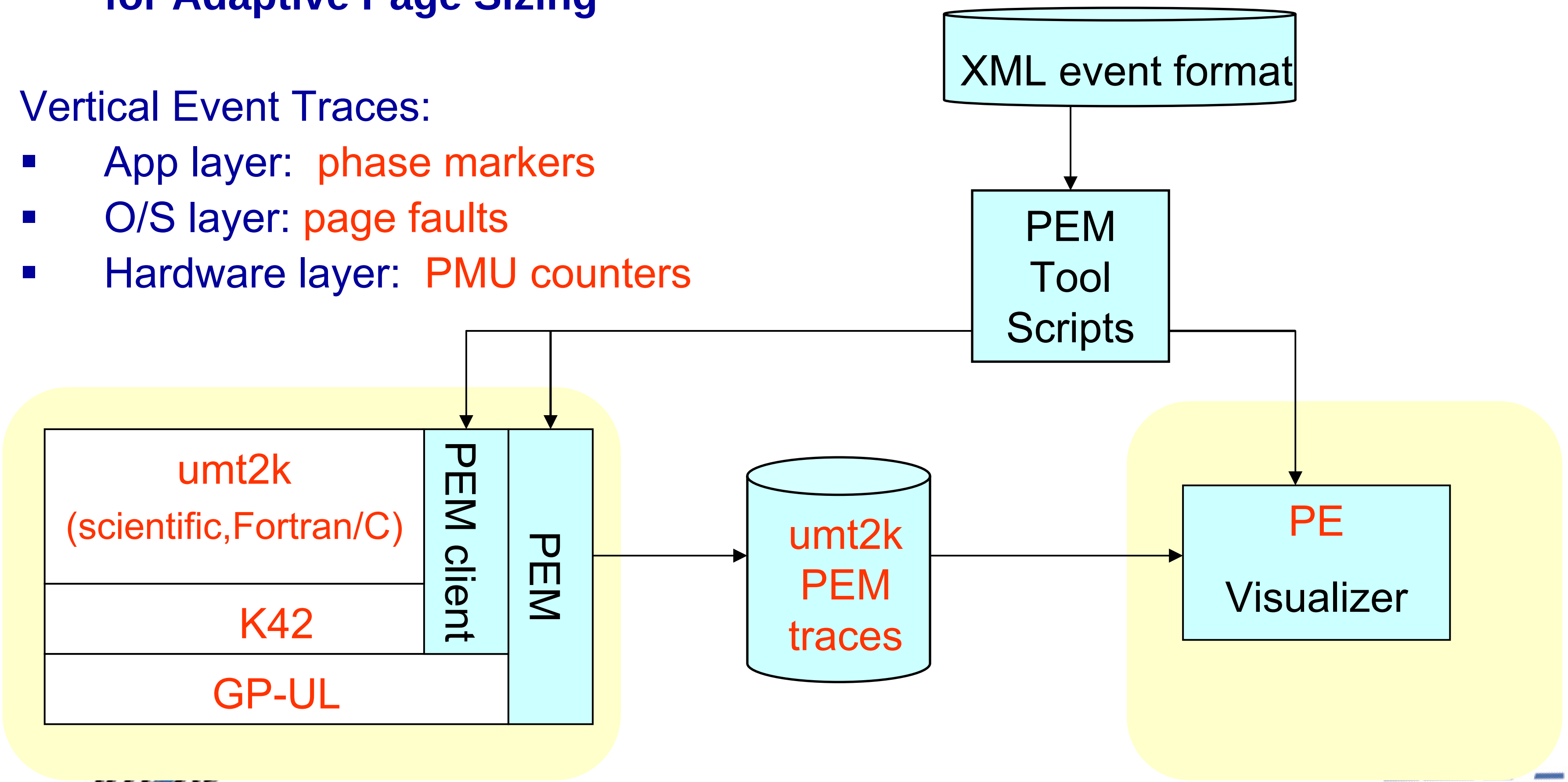
Contacts for PE Visualizer tool:
Peter Sweeney (IBM),
Matthias Hauswirth (U. Colorado)

PEM Scenario: Exploring the Performance Impact of Large Pages

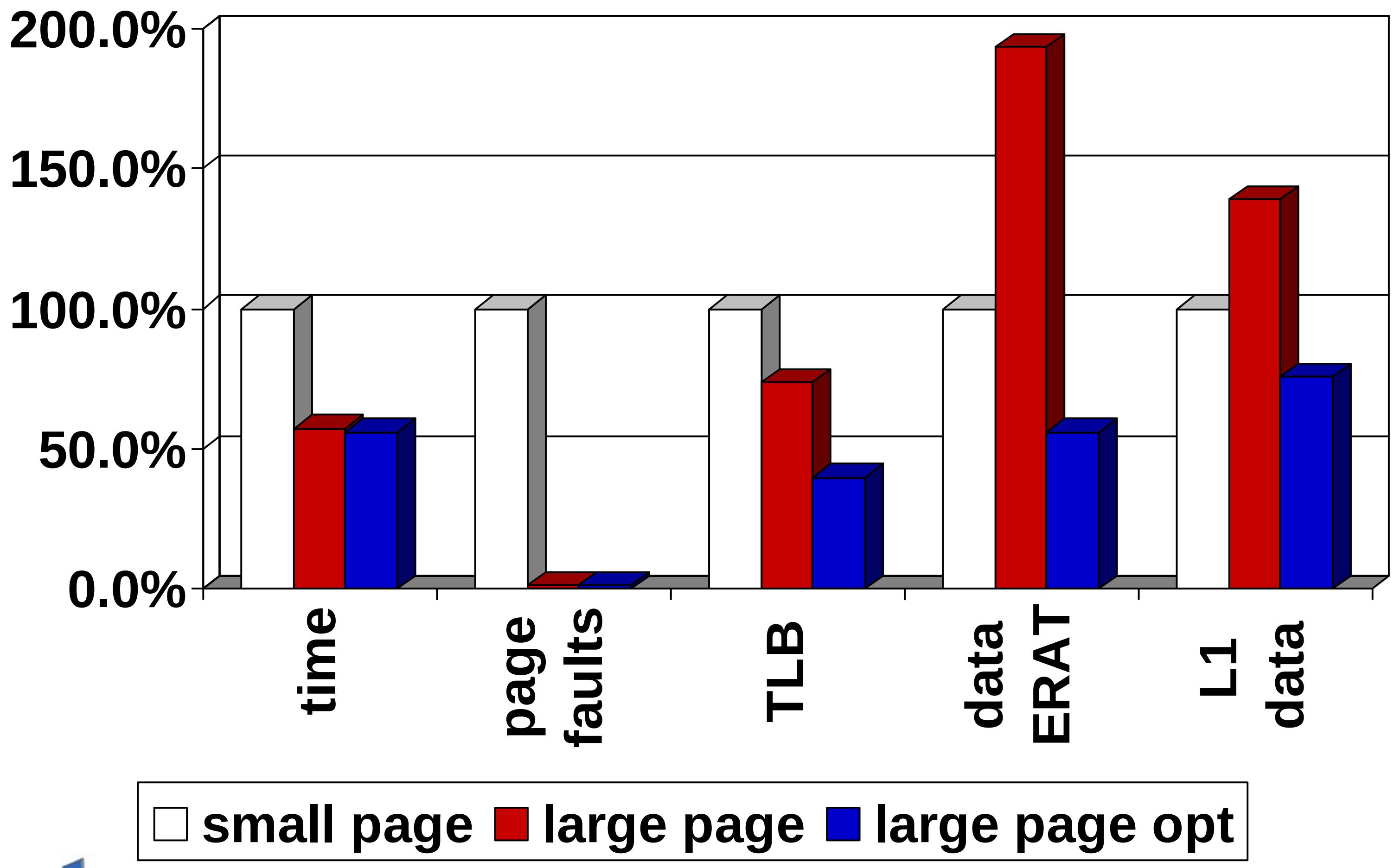
Preliminary Work for building a CPO Agent for Adaptive Page Sizing

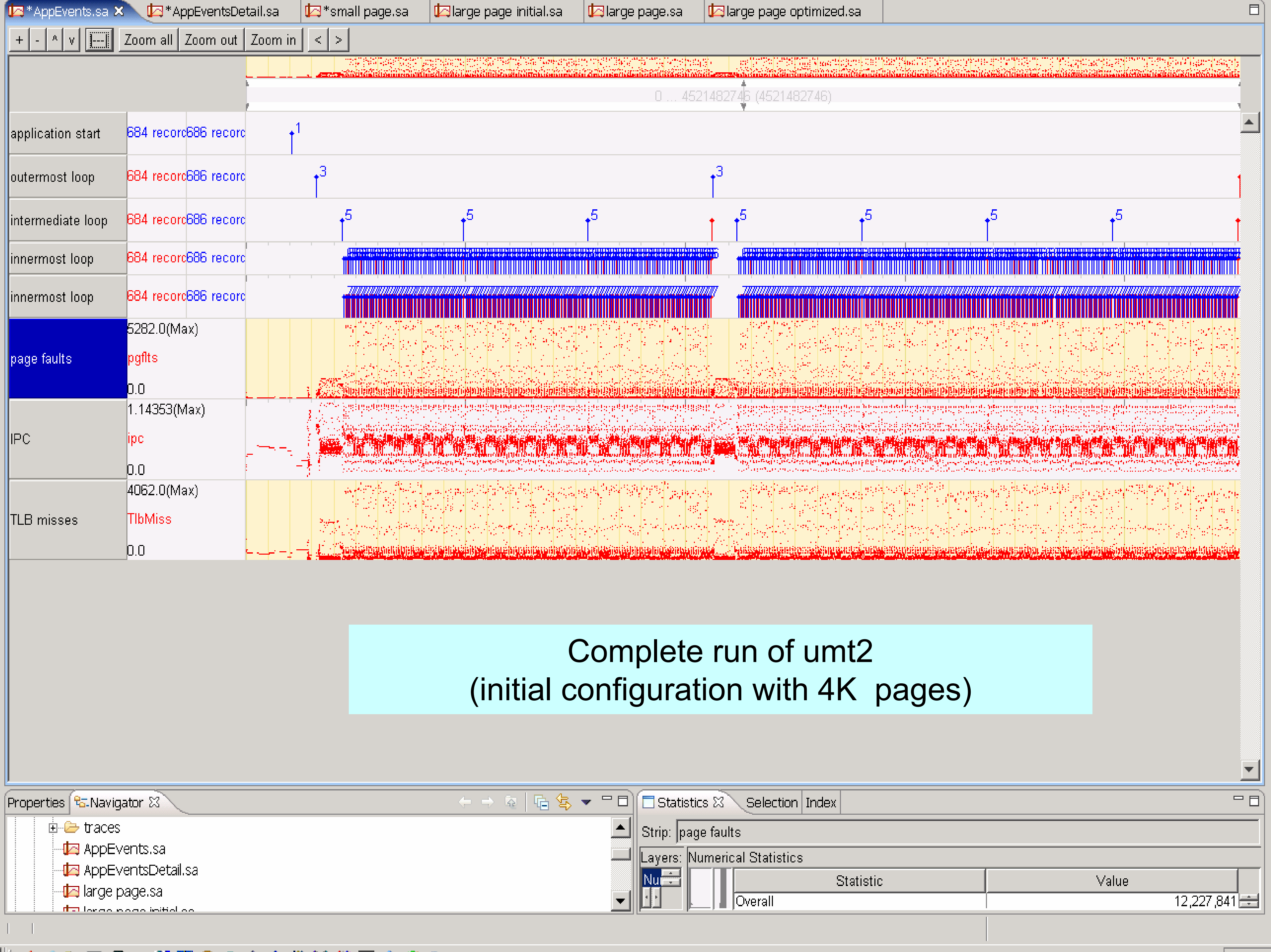
Vertical Event Traces:

- App layer: phase markers
- O/S layer: page faults
- Hardware layer: PMU counters

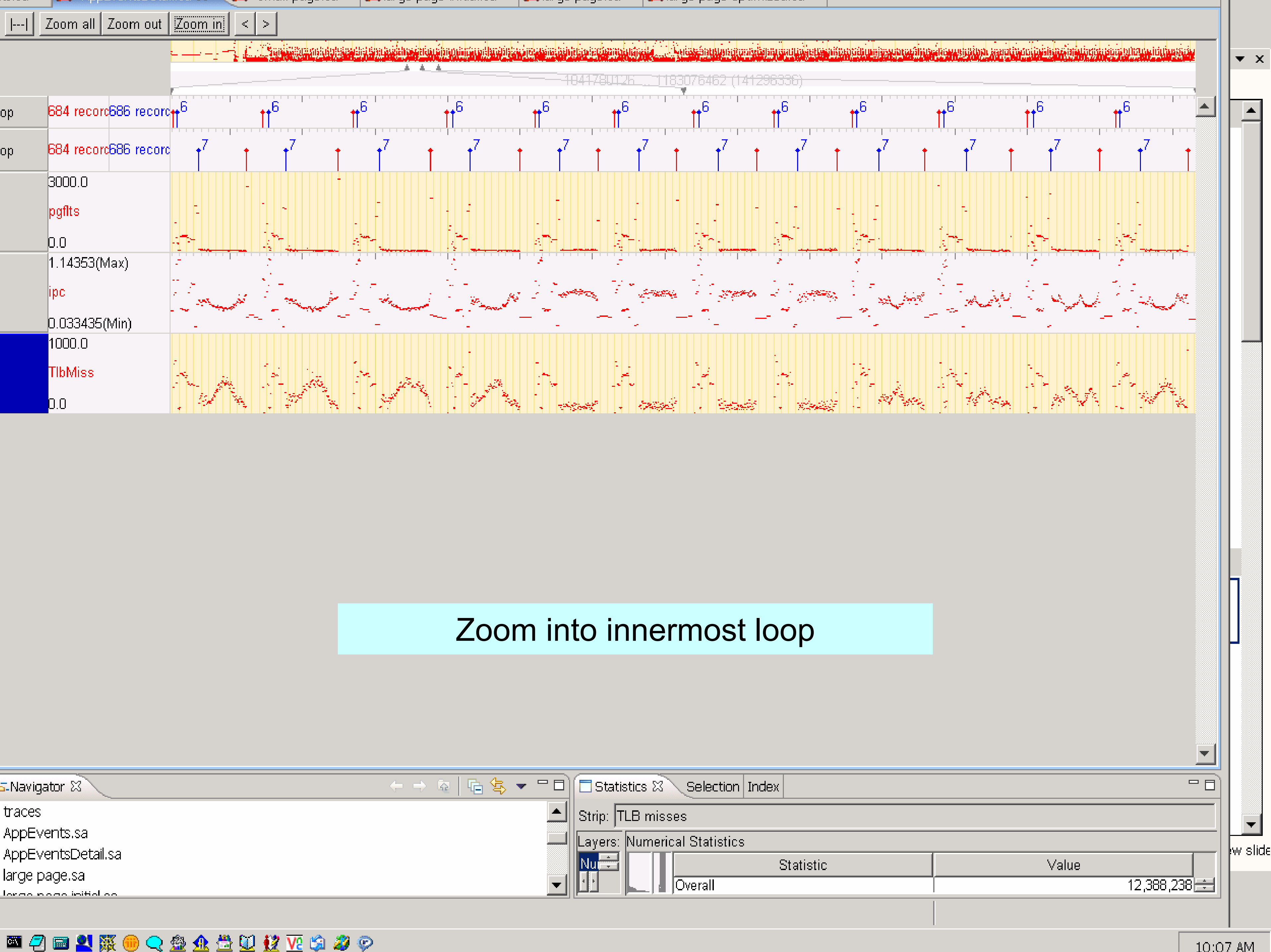


Summary of Performance Exploration





Complete run of umt2
(initial configuration with 4K pages)



|--|
Zoom all
Zoom out
Zoom in
<
>
1041780126 ... 1183076462 (141296336)
684 record
686 record
3000.0
pgflts
0.0
1.14353(Max)
ipc
0.033435(Min)
1000.0
TlbMiss
0.0
Zoom into innermost loop
Navigator
traces
AppEvents.sa
AppEventsDetail.sa
large page.sa
Statistics
Selection
Index
Strip:
TLB misses
Layers:
Numerical Statistics
Statistic
Value
Overall
12,388,238
10:07 AM

```
/* Set angular fluxes for reflected angles */

snxyzref(&nout, &npart, &nbelem, &ndim, &nbsets, &ndir,
     &m, &mtmp_ref(1, thnum), ix1, ix2, lcx,
     quadwt, omega, &tmp_ref(1, thnum), psib,
     &abdym_ref(1, thnum), A_bdy);

/* Sweep the mesh, calculating PSIC for each corner; the boundary */
/* flux array PSIB is also updated here.                          */

    TraceAPPAppPhaseStart( 6, __LINE__, __FILE__);

snswp3d(&npart, &nelem, &nelempad, &ncornr, &nfaces, &nzones,
    &nsides, &ndim, &nbelem, &m, &mpsi, &npsi, &ndir, ipath,
    &next_ref(1, m), konnect, kktc, kktf,
    kkcz, ksbdy, kktzcf,
    faces_corner, listcf, corner_zone,
    &omega_ref(1, m), A_fep, A_pez, A_fpz,
    &abdym_ref(1, thnum), sigvol, &sosf_ref(1, 1, thnum),
    &sosz_ref(1, 1, thnum), &qc_ref(1, 1, thnum),
    facewt, zonewt, tetwt, &adotm_ref(1, 1, thnum),
    &tpsic_ref(1, 1, thnum), psic, psib,
    &psi_inc_ref(1, 1, thnum), 8L);

    TraceAPPAppPhaseEnd( 6, __LINE__, __FILE__);

/* Add this angle's contribution to the flux moments */
snmmnt(&npart, &ncornr, &m, &ndir, &nmomt, &quadwt_ref(m),
       ynm, &tpsic_ref(1, 1, thnum), &tphic_ref(1, 1, thnum));

}
```

Application instrumentation to trace phase marker events

Properties | Navigator | Statistics | Selection | Index

main.f
rtmainsn.f
snflwxyz.c
snswp3d.c

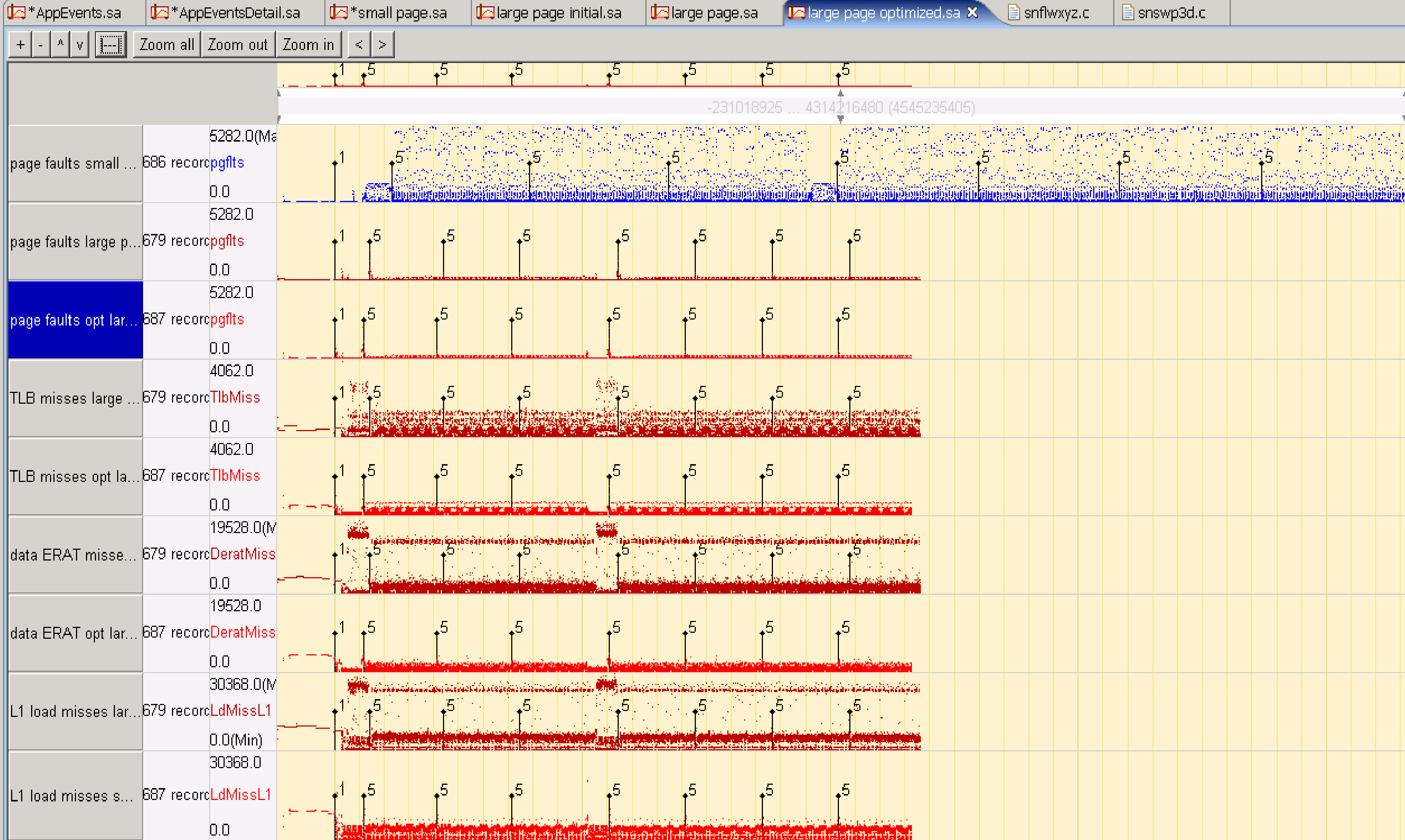
Strip: TLB misses

Layers: Numerical Statistics

Statistic	Value
Overall	12,388,238

Writable | Insert | 501 : 53

10:08 AM



Blue – 4k pages

Brown – Initial Large Page Mapping: each structure aligned at large page boundary

Red - Optimized large page mapping: Offset each data structure to avoid conflicts

Outline

1. Motivation
2. X10 language
3. Compiler challenges and opportunities
4. Runtime system challenges and opportunities
5. Conclusions

X10 Status and Plans

- Draft Language Design Report available internally w/ set of sample programs
- Implementation begun on X10 Prototype #1 for 1/2005
  - Functional reference implementation of language subset, not optimized for performance
  - Support for calls to single-threaded native code (C, Fortran)
- Productivity experiments planned for 7/2005
  - Use prototype #1 and related tools (PE, refactoring) to compare X10 w/ MPI, UPC
  - Revise language based on feedback from productivity experiments
- Prototype #2 planned for 12/2005
  - Includes design & prototype implementation of selected optimizations for parallelism, synchronization and locality in X10 programs
  - Revise language based on feedback from design evaluation

Summary: X10 addresses Important Productivity Attributes for an HPC Language

1. **Safety** -- eliminate entire classes of errors through static & dynamic checks e.g., Type errors, Initialization errors, Memory errors, Concurrency errors, Consistency errors, …

2. **Portability** – across multiple platforms, multiple system generations, and multiple application domains

3. **Design for Optimized Implementation** --- give compiler & runtime system freedom to manage resources

4. **Integration ---** with existing Languages, Environment, Libraries, and Tools

Conclusions and Future Work

- Future Large-scale Parallel Systems will be accompanied by severe productivity and performance challenges
  - ➔ Opportunity for Languages, Compilers, and Runtime technologies to have even greater impact on scalable systems than before
- Summarized X10 language approach in PERCS project, with a focus on next steps:
  - Use applications and productivity studies to refine design decisions in X10
  - Prototype solutions to address implementation challenges
- Future work (beyond 2005)
  - Community effort to build consensus on standardized "high productivity" languages for HPC systems in the 2010 timeframe
  - Explore integration of X10 ideas with other research language efforts under way in IBM
    - XJ, BPEL, …