X10: A High-Productivity Approach to Programming Multi-Core Systems

x10.sf.net

Vivek Sarkar

vsarkar@us.ibm.com

Senior Manager, Programming Technologies

IBM T.J. Watson Research Center

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

Acknowledgments

- **X10 Core Team**
  - Rajkishore Barik
  - Chris Donawa
  - Allan Kielstra
  - Nate Nystrom
  - Igor Peshansky
  - Christoph von Praun
  - Vijay Saraswat
  - Vivek Sarkar
  - Tong Wen
- **X10 Tools**
  - Philippe Charles
  - Robert Fuhrer
  - Stan Sutton
- **Emeritus**
  - Kemal Ebcioglu
  - Christian Grothoff

X10 Publications

1. "X10: An Object-Oriented Approach to Non-Uniform Cluster Computing", P. Charles, C. Donawa, K. Ebcioglu, C. Grothoff, A. Kielstra, C. von Praun, V. Saraswat, V. Sarkar. OOPSLA conference, October 2005.
2. "Concurrent Clustered Programming", V. Saraswat, R. Jagadeesan. CONCUR conference, August 2005.
3. "An Experiment in Measuring the Productivity of Three Parallel Programming Languages”, K. Ebcioglu, V. Sarkar, T. El-Ghazawi, J. Urbanic. P-PHEC workshop, February 2006.
4. "Experiences with an SMP Implementation for X10 based on the Java Concurrency Utilities”, R. Barik, V. Cave, C. Donawa, A. Kielstra, I. Peshansky, V. Sarkar, PMUP workshop, September 2006.
5. “May-Happen-in-Parallel Analysis of X10 programs”, S. Agarwal, R. Barik, V. Sarkar, R. Shyamasundar, PPoPP 2007 conference, March 2007 (to appear).
6. “Deadlock-Free Scheduling of X10 Computations with Bounded Resources”, S.Agarwal, R.Barik, D.Bonachea, V.Sarkar, R.Shyamasundar, K.Yelick, SPAA 2007 conference, June 2007 (to appear).

X10 tutorials

- PACT 2006, OOPSLA 2006, PPoPP 2007

Programming Technologies Research at IBM

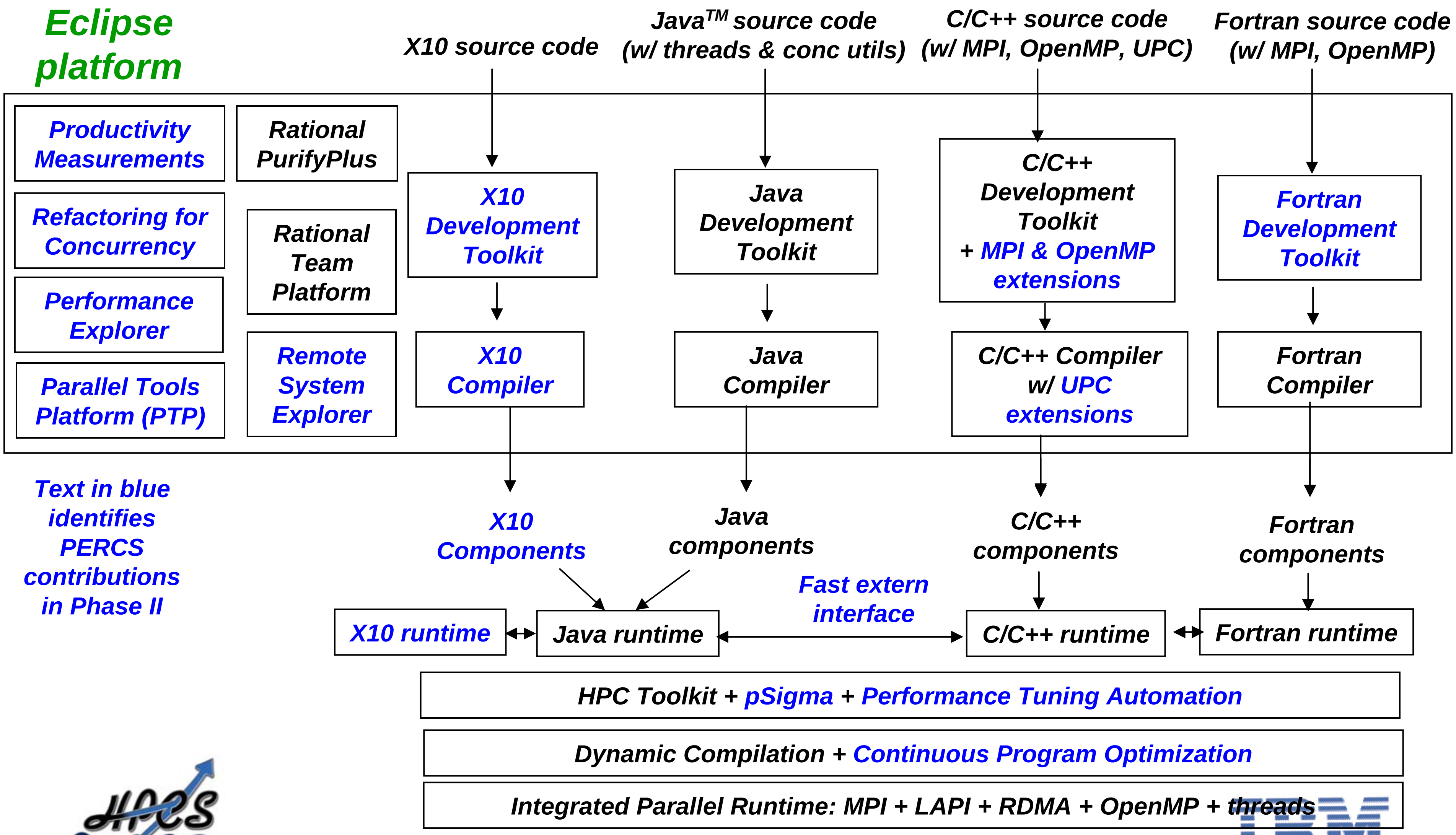
Goal: Focus our research on core technologies for development, deployment, and execution of programs and related software assets

Focus Research Areas and Current Projects:

λ

- ***Programming Models and Programming Language Design***
  - **Collage, DALI/XJ, X10**
- ***Development Tools***
  - **CSQ (includes SAFE, Security Analysis, Scripting Analysis), Parallel Tools, SAFARI**
- ***Deployment, Execution, Optimization***
  - **Dynamic Optimization, Jikes RVM, Metronome, PDS/Mirage,**

PERCS Programming Model, Tools and Compilers (Productive Easy-to-use Reliable Computer System)



Eclipse platform

Productivity Measurements

Rational PurifyPlus

Refactoring for Concurrency

Rational Team Platform

Performance Explorer

Remote System Explorer

Parallel Tools Platform (PTP)

X10 source code

Java™ source code (w/ threads & conc utils)

C/C++ source code (w/ MPI, OpenMP, UPC)

Fortran source code (w/ MPI, OpenMP)

X10 Development Toolkit

Java Development Toolkit

C/C++ Development Toolkit + MPI & OpenMP extensions

Fortran Development Toolkit

X10 Compiler

Java Compiler

C/C++ Compiler w/ UPC extensions

Fortran Compiler

X10 Components

Java components

C/C++ components

Fortran components

Fast extern interface

X10 runtime

Java runtime

C/C++ runtime

Fortran runtime

HPC Toolkit + pSigma + Performance Tuning Automation

Dynamic Compilation + Continuous Program Optimization

Integrated Parallel Runtime: MPI + LAPI + RDMA + OpenMP + threads

Text in blue identifies PERCS contributions in Phase II

Outline

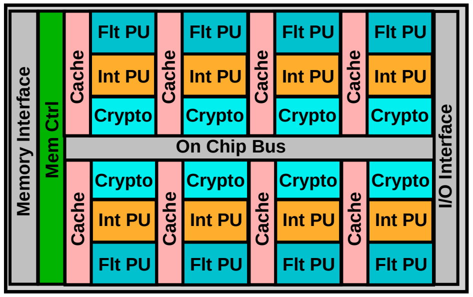
- **Software challenges for multi-core systems**
- X10 Programming Model and Language
- X10 Productivity Analysis
- X10 Implementation
- Conclusions

Future Multi-Core Systems: a new Era of Mainstream Parallel Processing

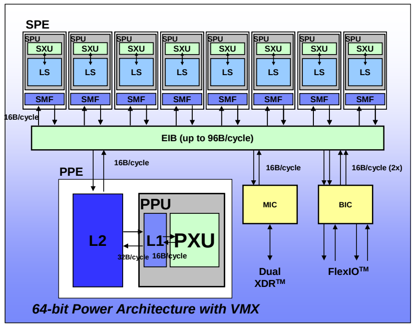
The Challenge:

Parallelism scaling replaces frequency scaling as foundation for increased performance ➔ Profound impact on future software

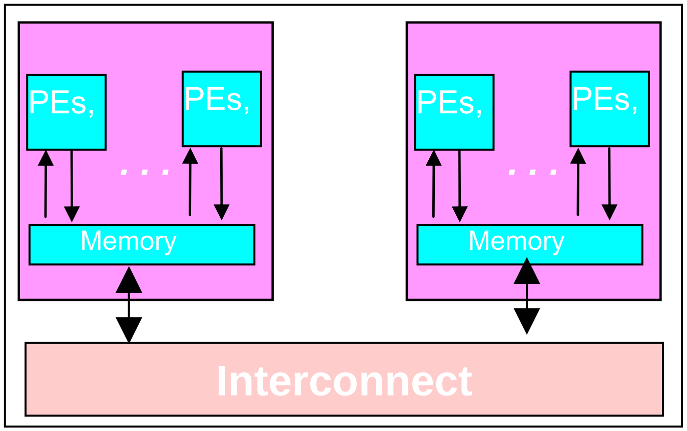
Homogeneous Multi-core



Heterogeneous Multi-core



Multi-Core Cluster



Our response:

Use X10 as a new language for parallel hardware that builds on existing tools, compilers, runtimes, virtual machines and libraries

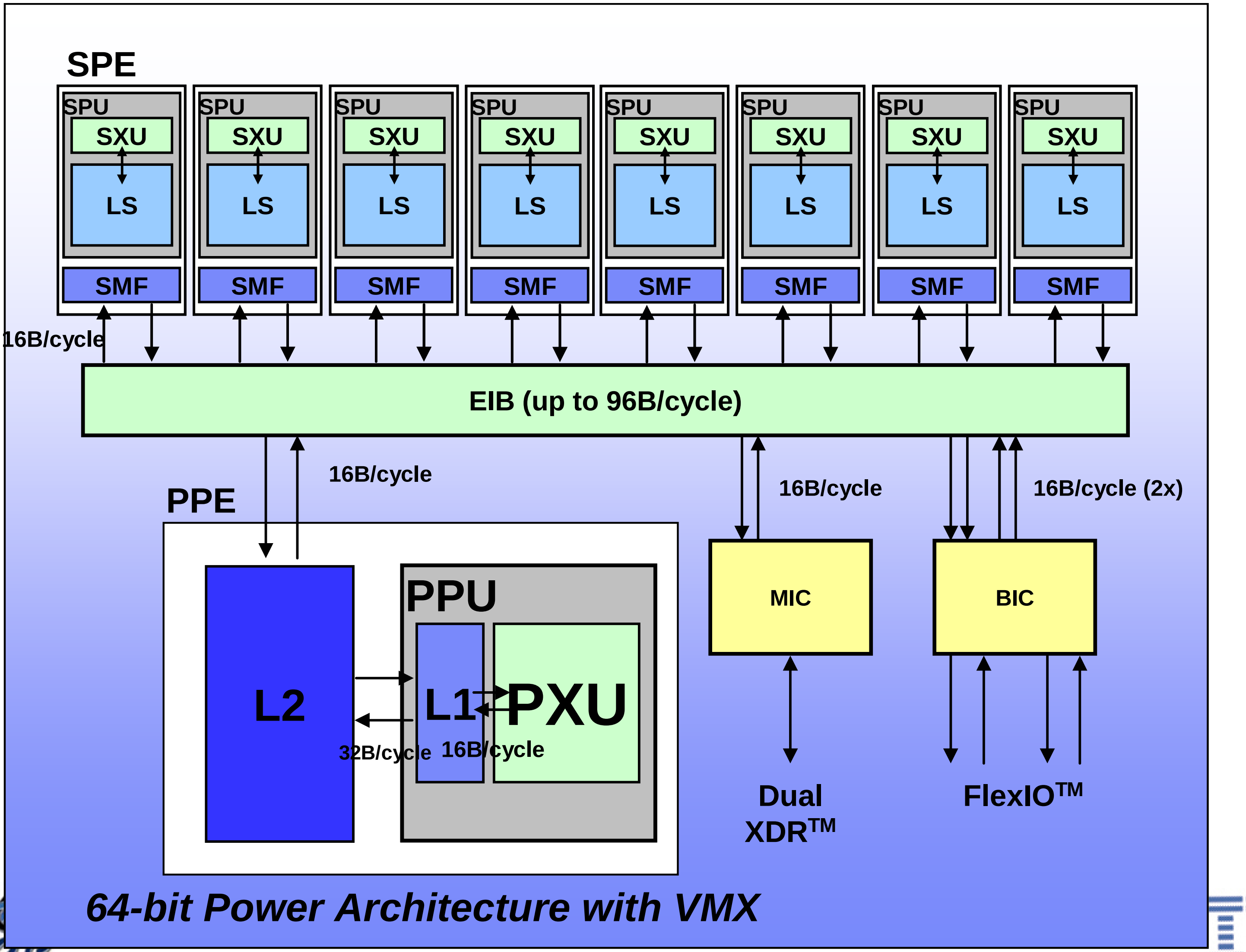
Current state of the art in Automatic Parallelization and Explicit Parallelism

- **Automatic parallelization technologies have been successful in SIMDization, Instruction scheduling, Data privatization, …**

  … but whole-program automatic parallelization still eludes us

- **Explicit parallel programming models have either focused on uniform shared-memory parallelism or message-passing parallelism, neither of which is appropriate for multi-core processors with a non-uniform shared memory model**
  - **Threaded models like OpenMP and Java have a uniform view of memory**
  - **Bulk-synchronous MPI model is SPMD (one process per node) with infrequent coarse-grained barriers and collective/two-sided communications**
  - **PGAS model (CAF, UPC, Titanium) is SPMD (one process per node) with coarse-grained barriers and fine-grained one-sided communications**

Cell Programming Model
(Heterogeneous Multi-Core Parallelism)



Disconnect between Uniform Memory Access model and Heterogeneous Multi-Core Parallelism

DMA Commands

Put - Transfer from Local Store to EA space
Puts - Transfer and Start SPU execution
Putr - Put Result - (Arch. Scarf into L2)
Putl - Put using DMA List in Local Store
Putrl - Put Result using DMA List in LS (Arch)
Get - Transfer from EA Space to Local Store
Gets - Transfer and Start SPU execution
Getl - Get using DMA List in Local Store
Sndsig - Send Signal to SPU
Command Modifiers: <**f**,**b**>
f: Embedded Tag Specific Fence
  Command will not start until all previous commands in same tag group have completed
b: Embedded Tag Specific Barrier
  Command and all subsiquent commands in same tag group will not start until previous commands in same tag group have completed

SL1 Cache Management Commands

sdcrt - Data cache region touch (DMA Get hint)
sdcrtst - Data cache region touch for store (DMA Put hint)
sdcrz - Data cache region zero
sdcrs - Data cache region store
sdcrf - Data cache region flush

Command Parameters

LSA - Local Store Address (32 bit)
EA - Effective Address (32 or 64 bit)
TS - Transfer Size (16 bytes to 16K bytes)
LS - DMA List Size (8 bytes to 16 K bytes)
TG - Tag Group(5 bit)
CL - Cache Management / Bandwidth Class

Synchronization Commands

Lockline (Atomic Update) Commands:
  getllar - DMA 128 bytes from EA to LS and set Reservation
  putllc - Conditionally DMA 128 bytes from LS to EA
  putlluc - Unconditionally DMA 128 bytes from LS to EA
barrier - all previous commands complete before subsiquent commands are started
mfcsync - Results of all previous commands in Tag group are remotely visible
mfceieio - Results of all preceding Puts commands in same group visible with respect to succeeding Get commands

What's Involved in Programming for Cell?

- Partition application into PPE and SPE portions
- Map application data onto Local Store for SPE parts
    - this typically requires both temporal and spatial concerns for data that needs to be streamed in and out of Local Store
- Code SPE functions to exploit SIMD processing capabilities

```
 vector float a,b,c,d;
a=(vector float){2.3,6.0,0.0,5.1};
d = spu_madd(a,b,c); // d=a*b+c
```

- Orchestrate the data streaming using MFC commands

```
vector float a[8];
spu_mfcdma32(a,p,128,tagnum,READ);
spu_mfcstat(ALL);
```

- Parallelize across multiple SPEs and 2 threads on the PPE

What’s Involved in Programming for Cell? (contd.)

- SIMDize if at all possible
  - reorder data, array-of-structures vs structure-of-arrays
  - either use intrinsics, or allow the compiler to do it automatically
  - Align SIMD data on 16 byte boundaries – go to any lengths !
- Try to have data in Local Store before it is needed
  - DMA latency is in the hundreds of cycles
  - SPEs are single threaded processors
- Its better to do DMA from the SPE side
  - more channels / less trouble synchronizing
- Try to reduce the amount of “branchy” code
- Using an SPE to run Scalar code is OK
  - its really another – application specific - processor

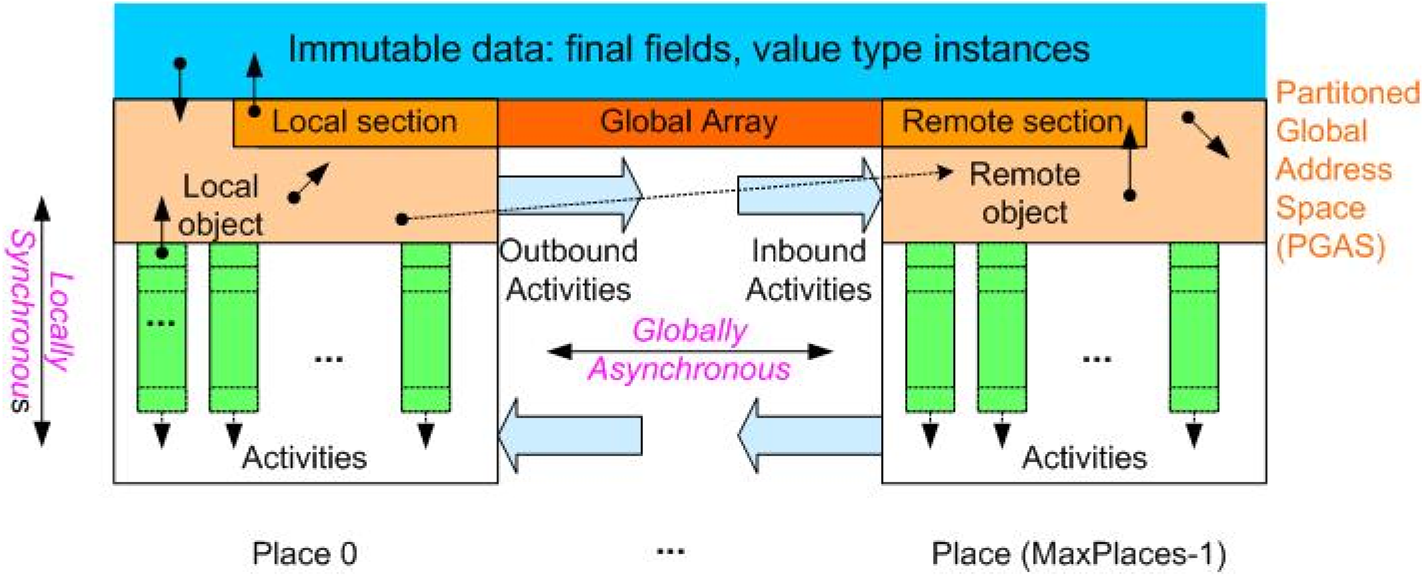
Outline

- Software challenges for multi-core systems
- **X10 Programming Model and Language**
- X10 Productivity Analysis
- X10 Implementation
- Conclusions

X10 Approach

- Unified abstractions of asynchrony and concurrency for use in
  - Multi-core SMP Parallelism
  - Messaging and Cluster Parallelism
- Productivity
  - High Level Language designed for portability and safety
  - X10 Development Toolkit for Eclipse
- Performance
  - Extend VM+JIT model for high performance
  - Performance transparency – don’t lock out the performance expert!
    - expert programmer should have controls to tune optimizations and tailor distributions & communications to actual deployment
- Build on sequential subset of Java language
  - Retain core values of Java --- productivity, ubiquity, maturity, security
  - Target adoption by mainstream developers with Java/C/C++ skills

X10 Programming Model



Storage classes:

- **Activity-local**
- **Place-local**
- **Partitioned global**
- **Immutable**

- Dynamic parallelism with a *Partitioned Global Address Space*
- *Places* encapsulate binding of activities and globally addressable data
- All concurrency is expressed as *asynchronous activities* – subsumes threads, structured parallelism, messaging, DMA transfers (beyond SPMD)
- *Atomic sections* enforce mutual exclusion of co-located data
  - No place-remote accesses permitted in atomic section
- *Immutable* data offers opportunity for single-assignment parallelism

Deadlock safety: any X10 program written with async, atomic, finish, foreach, ateach, and clocks can never deadlock

X10 Language

- async [(*Place*)] [clocked(c…)] *Stm*
  - Run Stm asynchronously at Place
- finish *Stm*
  - Execute s, wait for all asyncs to terminate (generalizes join)
- foreach (point *P* : *Reg*) *Stm*
  - Run Stm asynchronously for each point in region
- ateach (point *P* : *Dist*) *Stm*
  - Run Stm asynchronously for each point in dist, in its place.
- atomic *Stm*
  - Execute Stm atomically
- new T
  - Allocate object at this place (here)
- new T[d] / new T value [d]
  - Array of base type T and distribution d

- Region
  - Collection of index points, e.g.

    region r = [1:N,1:M];
- Distribution
  - Mapping from region to places, e.g.
    - dist d = block(r);
- next
  - suspend till all clocks that the current activity is registered with can advance
  - Clocks are a generalization of barriers and MPI communicators
- future [(*Place*)] [clocked(c…)] *Expr*
  - Compute Expr asynchronously at Place
- F. force()
  - Block until future F has been computed
- extern
  - Lightweight interface to native code

Deadlock safety: any X10 program written with above constructs excluding future can never deadlock

- Can be extended to restricted cases of using future

X10 Arrays, Regions, Distributions

ArrayExpr:

new ArrayType (Formal) { Stm }

Distribution Expr -- Lifting

ArrayExpr [*Region*] -- Section

ArrayExpr | *Distribution* -- Restriction

ArrayExpr || *ArrayExpr* -- Union

ArrayExpr.overlay(*ArrayExpr*) -- Update

ArrayExpr. scan(*[fun [, ArgList]*)

ArrayExpr. reduce(*[fun [, ArgList]*)

ArrayExpr.lift(*[fun [, ArgList]*)

ArrayType:

Type [Kind] [ ]

Type [Kind] [region(N)]

Type [Kind] [*Region*]

Type [Kind] [*Distribution*]

Region:

Expr : Expr -- 1-D region

[*Range, …, Range*] -- Multidimensional Region

Region && *Region* -- Intersection

Region || *Region* -- Union

Region – *Region* -- Set difference

BuiltinRegion

Dist:

Region -> Place -- Constant distribution

Distribution | *Place* -- Restriction

Distribution | *Region* -- Restriction

Distribution || *Distribution* -- Union

Distribution – *Distribution* -- Set difference

Distribution.overlay (*Distribution*)

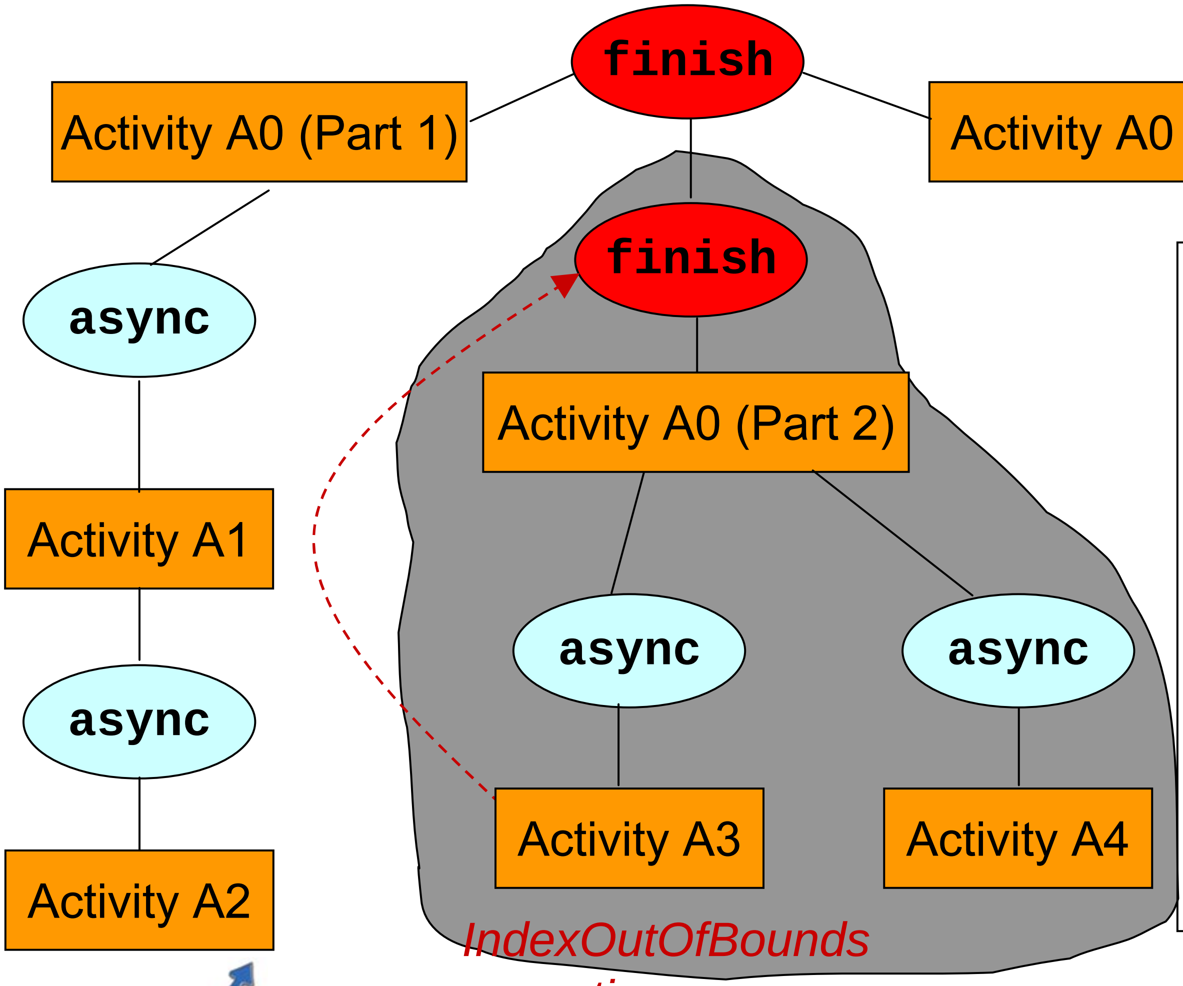
BuiltinDistribution

Language supports type safety, memory safety, place safety, clock safety.

X10 Language Constructs: Examples

```
1)  finish { // Intra-place parallelism
       final int x = … , y = … ;
       async a.foo(x); // Initiate two activities at same
       async b.bar(y); // place as parent activity
    } // Wait for both activities to complete

2) finish { // Inter-place parallelism
       final int x = … , y = … ;
       async (a) a.foo(x); // Execute at a’s place
       async (b) b.bar(y); // Execute at b’s place
   }

3) // Implicit and explicit versions of remote fetch-and-add
   a) a.x += b.y ;
   b) async (b) {final int v = b.y; async (a) atomic a.x += v; }
```

X10 Dynamic Activity Invocation Tree



IndexOutOfBounds exception

```
// X10 pseudo code
main(){ // implicit finish
  Activity A0 (Part 1);
  async {A1; async A2;}
  try {
    finish {
      Activity A0 (Part 2);
      async A3;
      async A4;
    }
  catch (…) { … }
  Activity A0 (Part 3);
}
```

X10 Language Constructs: Examples (contd.)

```
4) future<int> F = future(a) { a.baz() }; // returns immediately
   . . .
   int i = F.force(); // block until return value is obtained

5) // A is a local 1-D array, B is a distributed 2-D array
   int[.] A = new int[[0:N-1]];
   int[.] B = new int[dist.blockRows([0:M-1,0:N-1])];
   . . .
   // serial pointwise for loop
   for (point[j] : [1:N-1]) A[j] = f(A[j-1]);
   . . .
   // intra-place pointwise parallel loop
   foreach (point[j] : A.region) A[j] = g(A[j]);
   . . .
   // inter-place pointwise parallel loop
   ateach (point[i,j] : B.distribution) B[i,j] = h(B[i,j]);
```

Explicit vs. Implicit Syntax for Places

- **Explicit syntax** – target place specified explicitly for remote activity
  - async (a) { a.z = expr ; a.foo(x); . . . }
  - BadPlaceException thrown if operation is performed on remote reference
- **Implicit syntax** – freely access global address space, and let compiler insert the target places
  - { a.z = expr ; . . . }
  - ➔ { final int T = expr; finish async(a) a.z = T; … }
  - More convenient to write code, but harder to control and debug performance
- **X10 approach**
  - Allow combination of implicit and explicit syntax
  - Extend type system with dependent types to statically identify local operations

Example: Monte Carlo Calculations

Explicit language concurrency greatly simplifies threading related constructs

Multi-Threaded Java

Single-Threaded Java

```
initTasks() { tasks = new ToTask[nRunsMC]; … }

public void runSerial() {
  results = new Vector(nRunsMC);
  // Now do the computation.
  PriceStock ps;
  for( int iRun=0; iRun < nRunsMC; iRun++ ) {
    ps = new PriceStock();
    ps.setInitAllTasks(initAllTasks);
    ps.setTask(tasks[iRun]);
    ps.run();
    results.addElement(ps.getResult());
  }
}
```

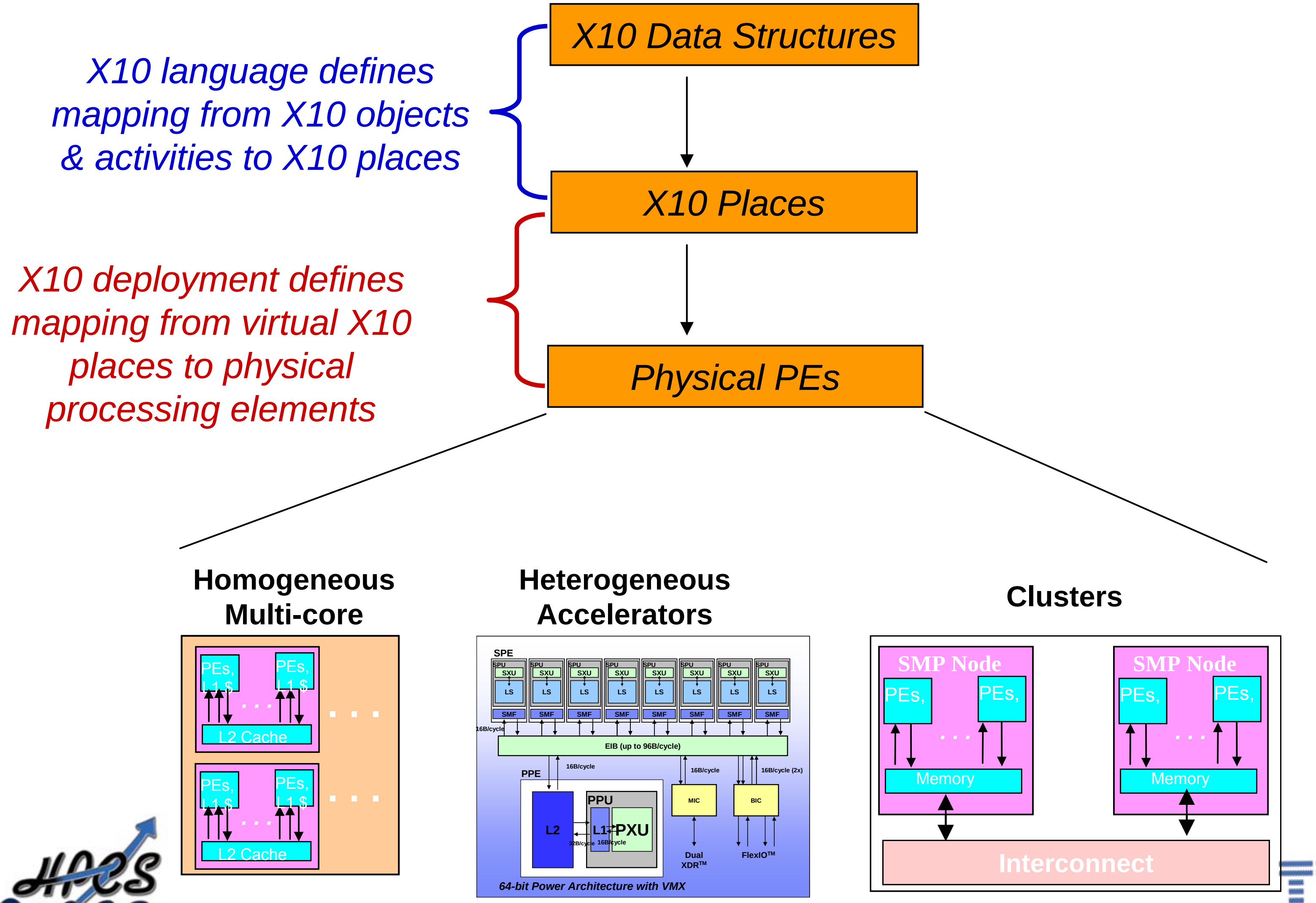
```
;
nnable [JGFM
MonteCarloBe
re work
nch.nthreads
hread(i,nRun
s[i]);


ad 0
ead(0,nRunsM


nch.nthreads
(Interrupted
  }
}
class AppDemoThread implements Runnable {
  ... // initialization code
  public void run() {
    PriceStock ps;
    int ilow, iupper, slice;
    slice = (nRunsMC+JGFMonteCarloBench.nthreads-1)
               / JGFMonteCarloBench.nthreads;
    ilow = id*slice;
    iupper = Math.min((id+1)*slice, nRunsMC);
    for( int iRun=ilow; iRun < iupper; iRun++ ) {
      ps = new PriceStock();
      ps.setInitAllTasks(AppDemo.initAllTasks);
      ps.setTask(AppDemo.tasks[iRun]);
      ps.run();
      AppDemo.results.addElement(ps.getResult());
    }
  } // run()
}
```

Multi-Threaded X10

```
initTasks() { tasks = new ToTask[dist.block([0:nRunsMC-1])]; … }

public void runDistributed()
{
  results = new x10Vector(nRunsMC);
  // Now do the computation
  finish ateach ( point[iRun] : tasks.distribution ) {
    PriceStock ps = new PriceStock();
    ps.setInitAllTasks((ToInitAllTasks) initAllTasks);
    ps.setTask(tasks[iRun]);
    ps.run();
    final ToResult r = ps.getResult(); // ToResult is a value type
    async(results) atomic results.v.addElement(r);
  }
}
```

Source: http://www.epcc.ed.ac.uk/javagrande/javag.html - The Java Grande Forum Benchmark Suite

X10 Deployment



X10 language defines mapping from X10 objects & activities to X10 places

X10 deployment defines mapping from virtual X10 places to physical processing elements

X10 Data Structures

X10 Places

Physical PEs

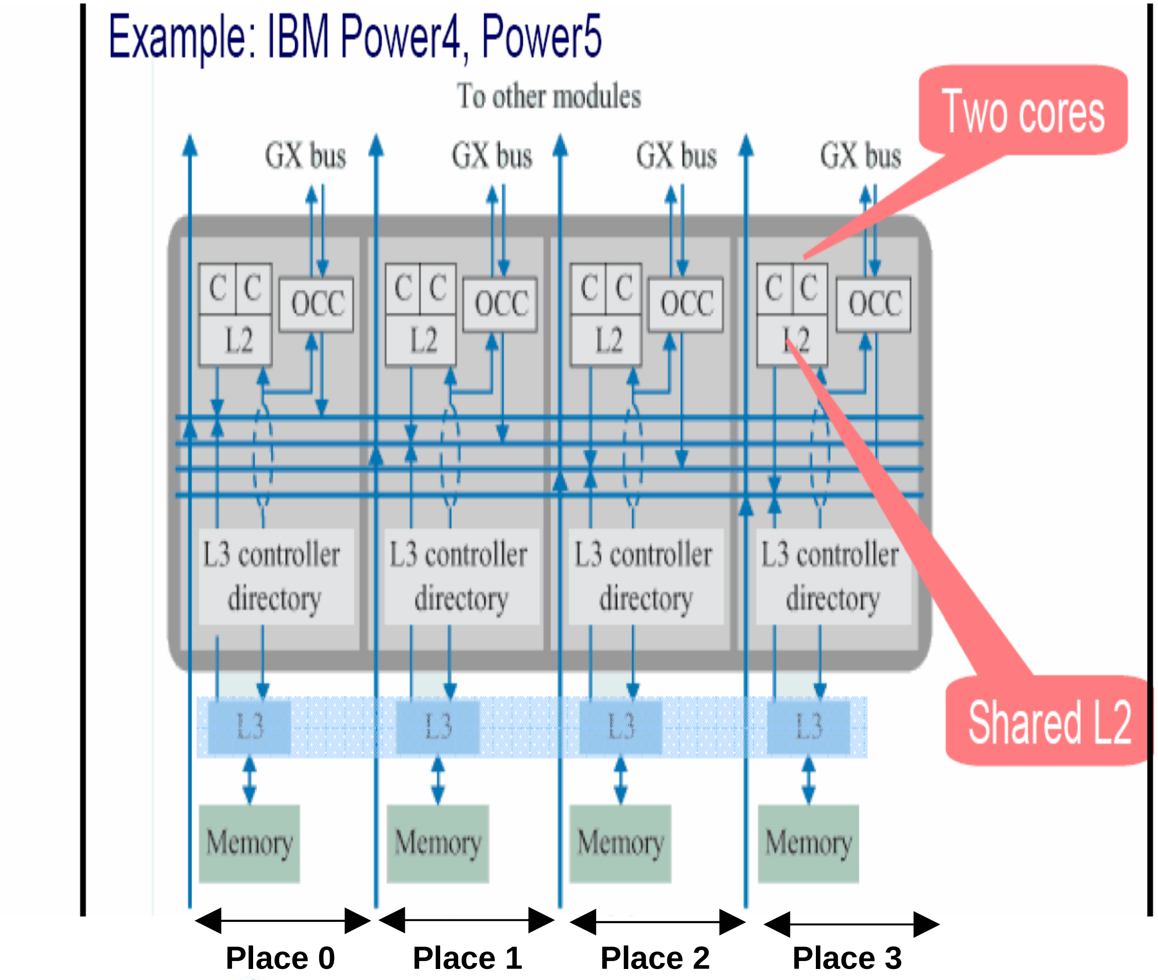
Homogeneous Multi-core

Heterogeneous Accelerators

Clusters

64-bit Power Architecture with VMX

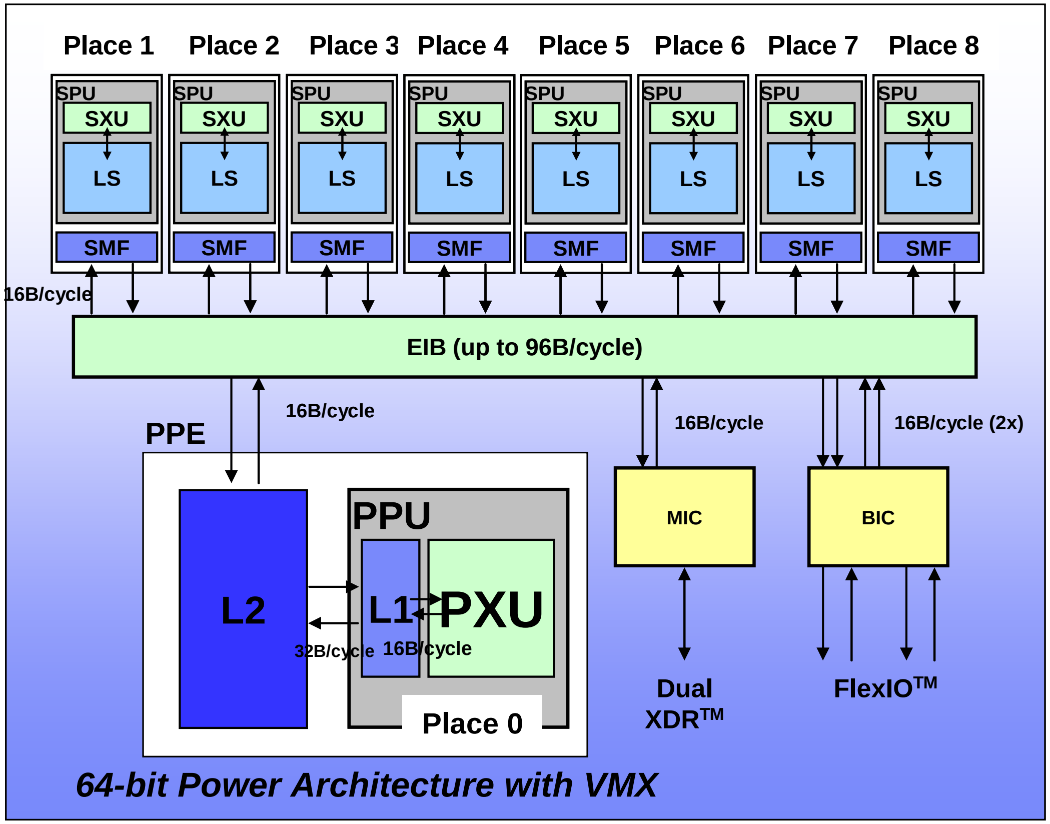
X10 Deployment on an SMP



- Basic Approach -- partition X10 heap into multiple place-local heaps
- Each X10 object is allocated in a designated place
- Each X10 activity is created (and pinned) at a designated place
- Allow an X10 activity to synchronously access data at remote places outside of atomic sections (implicit syntax)

➔ Thus, places serve as affinity hints for intra-SMP locality

Possible X10 Deployment for Cell (under discussion)



- Basic Approach:
  - map 9 places on to PPE + eight SPEs
  - Use finish & async's as high-level representation of DMAs
- Opportunities:
  - Exploit dynamic compilation and code specialization
- Challenges:
  - Weak PPE
  - SIMDization is critical
  - Lack of hardware support for coherence
  - Limited memory on SPE's
  - Limited performance of code with frequent conditional or indirect branches
  - Different ISA's for PPE and SPE.

Outline

- Software challenges for multi-core systems
- X10 Programming Model and Language
- **X10 Productivity Analysis**
- X10 Implementation
- Conclusions

X10 Productivity Analysis

Scenario: parallelization of serial X10/Java code

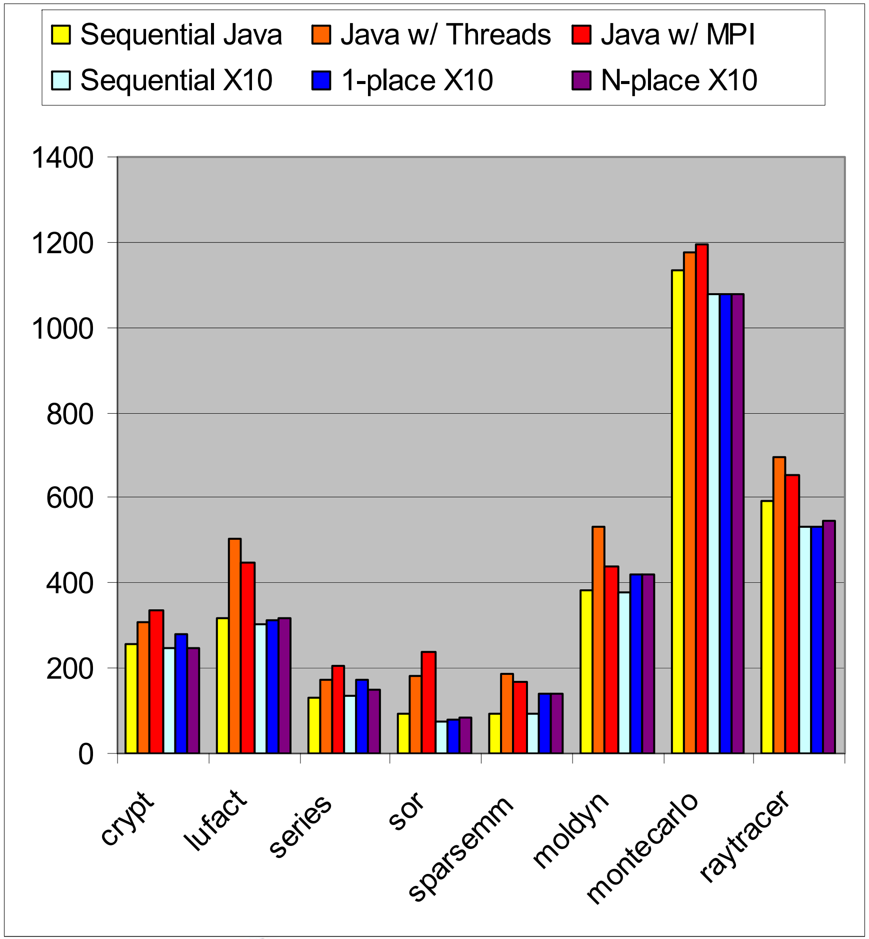
Two sets of results

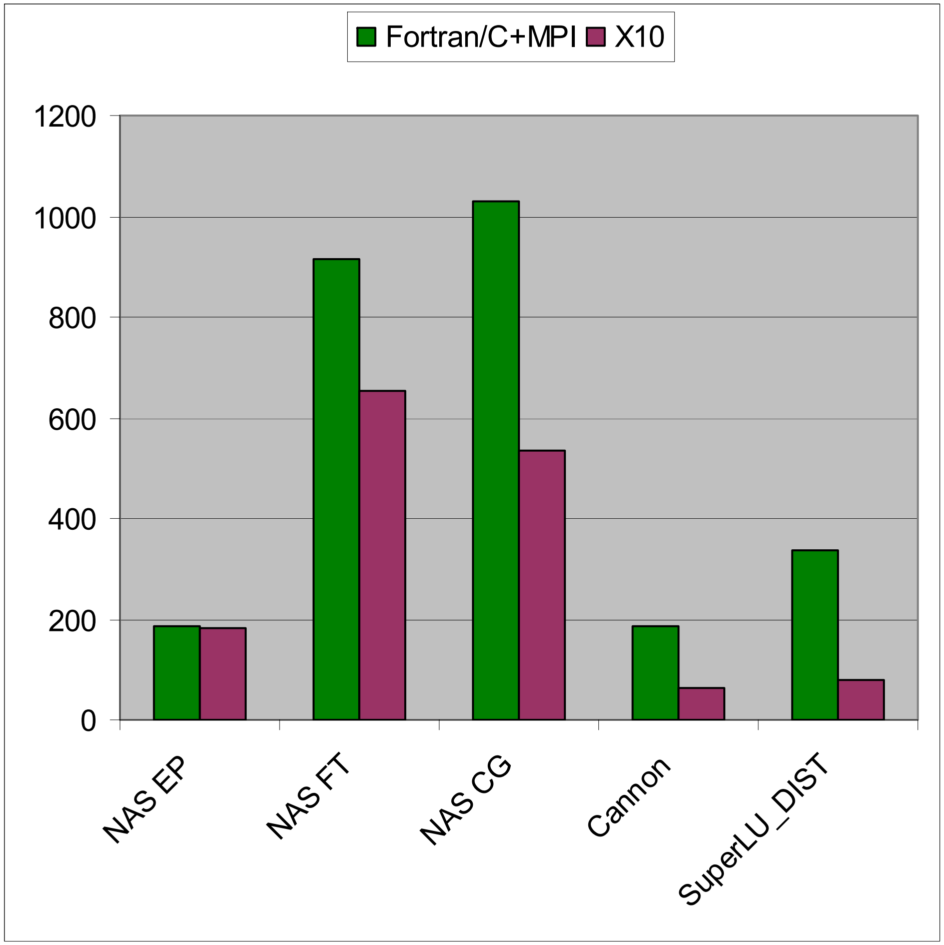
1. *Code size comparison* of serial, multi-threaded, and distributed versions of Java Grande benchmarks in Java and X10
   - Details in OOPSLA paper, “X10: An Object-Oriented Approach to Non-Uniform Cluster Computing”, (Section 5)
2. *Human productivity study* comparing *time to first correct parallel version* for C+MPI, UPC, and X10
   - Summary in P-PHEC 2006 workshop paper, "An Experiment in Measuring the Productivity of Three Parallel Programming Languages.
   - Acknowledgments for productivity study
     - Pittsburgh Supercomputing Center: Nick Nystrom, John Urbanic, Deborah Weisser
     - IBM Research Social Computing Group: Catalina Danis, Christine Halverson, Wendy Kellogg

Code Size Comparison: Java Grande Benchmarks

Language	**Code size metric**	**Serial version**	**Multithreaded version**	**Distributed version**
Java	Classes/SLOC	50/3254	64/4028	50/3973
+ threads	***SLOC ratio***		***1.24***	***1.22***
+ MPI	SSC	2592	3212	3133
	SSC ratio		***1.24***	***1.21***
	# stmts changed		1108	781
	Change ratio		***0.43***	***0.30***
X10	Classes/SLOC	49/3107	49/3212	49/3243
	SLOC ratio		***1.03***	***1.04***
	SSC		2594	2649
	SSC ratio		***1.04***	***1.07***
	# stmts changed		363	510
	Change ratio		***0.15***	***0.21***

Benchmark SLOC





OOPSLA 05 Onwards!

Fraguela, et al (UIUC).

Human Productivity Study (Comparison of MPI, UPC, X10)

- **Goals**
  - Contrast productivity of X10, UPC, and MPI for a statistically significant subject sample on a programming task relevant to HPCS Mission Partners
  - Validate the PERCS Productivity Methodology to obtain quantitative results that, given specific populations and computational domains, will be of immediate and direct relevance to HPCS.
- **Overview**
  - 4.5 days: May 23-27, 2005 at the Pittsburgh Supercomputing Center (PSC)
  - Pool of 27 comparable student subjects
  - Programming task: Parallelizing the alignment portion of Smith-Waterman algorithm (SSCA#1)
  - 3 language programming model combinations (X10, UPC, or C + MPI)
  - Equal environment as near as possible (e.g. pick of 3 editors, simple println stmts for debugging)
  - Provided expert training and support for each language

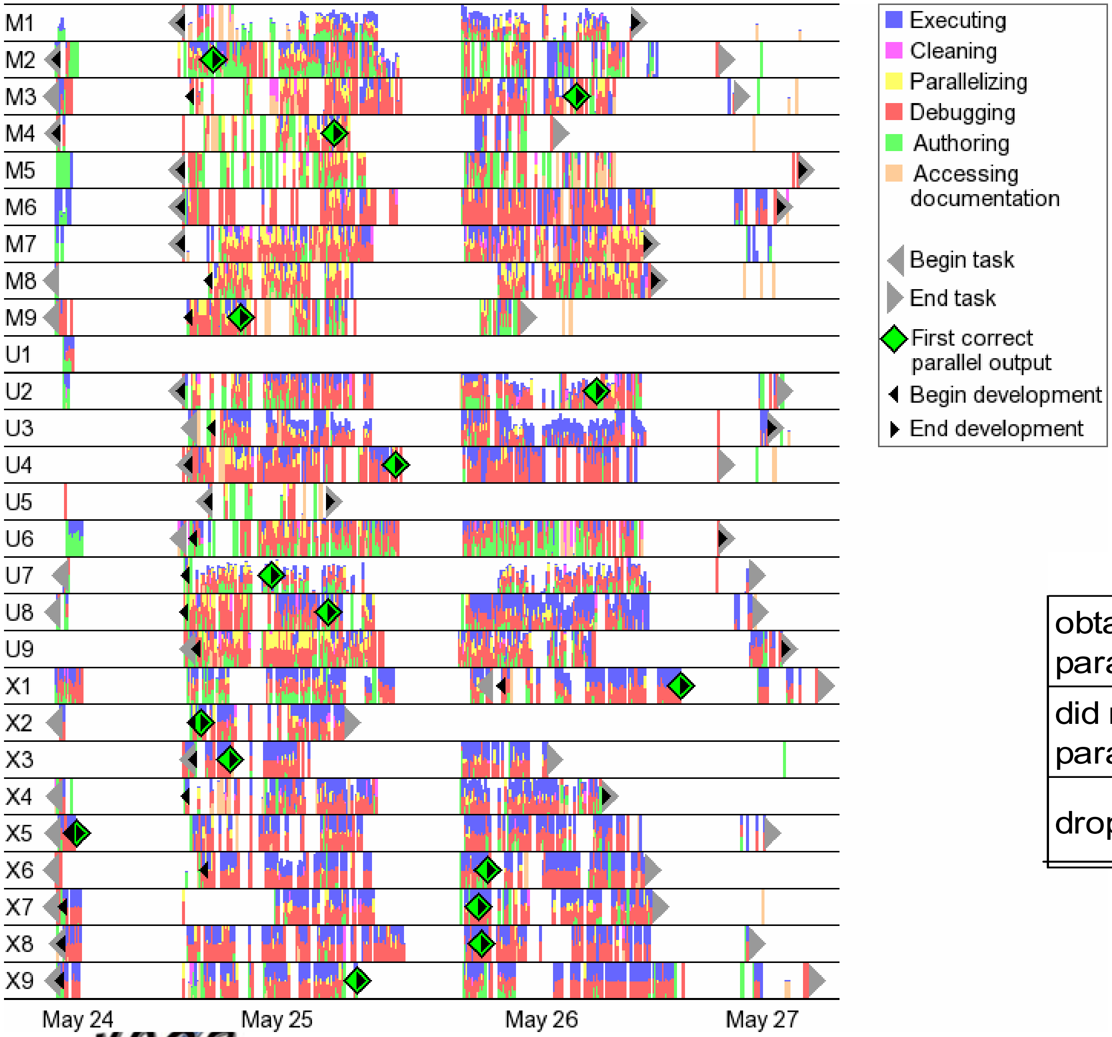
Data Summary

- 180,524 source, source diff, compiler, batch, shell, web, and window events were recorded for the 27 subjects.
- Each event contains detailed information for subsequent contextual and temporal analysis
- Example: compiler component
  - experiment and subject IDs
  - timestamp
  - compiler name
  - command line
  - number of errors and warnings
  - compiler output
  - links to source and batch records

Instrumented Raw Events for Full Experiment

user_id	source	src_diff	compiler	batch	shell	web	window	total
M1	132	353	179	168	1,305	9,775	2,450	14,362
M2	155	366	151	463	1,975	346	2,259	5,715
M3	237	465	330	107	1,386	2,736	2,483	7,744
M4	69	134	81	30	432	3,801	1,664	6,211
M5	119	271	139	24	608	299	1,458	2,918
M6	327	313	287	79	1,235	366	2,658	5,265
M7	409	1,067	427	77	1,920	1,300	3,691	8,891
M8	258	766	342	73	1,355	3,250	2,504	8,548
M9	116	247	160	59	875	913	1,224	3,594
U1	129	145	254	138	1,485	20	416	2,587
U2	224	525	256	240	1,669	733	2,222	5,869
U3	236	449	268	427	8,053	1,014	4,204	14,651
U4	316	420	162	298	1,301	580	1,478	4,555
U5	82	63	20	24	274	486	653	1,602
U6	297	388	179	227	1,479	389	1,691	4,650
U7	207	661	419	104	1,625	7,396	1,550	11,962
U8	244	500	238	303	7,529	645	1,563	11,022
U9	422	847	402	342	2,492	1,793	2,268	8,566
X1	767	354	645	0	2,104	987	2,056	6,913
X2	162	228	404	0	1,109	30	687	2,620
X3	766	329	432	0	1,421	91	1,419	4,458
X4	236	420	455	0	1,341	1,007	3,518	6,977
X5	680	251	669	0	1,809	844	1,753	6,006
X6	291	348	663	0	1,809	1,446	1,661	6,218
X7	238	484	595	0	1,731	307	1,396	4,751
X8	405	452	582	0	1,727	888	2,465	6,519
X9	217	319	887	0	2,634	1,025	2,268	7,350
total	7,741	11,165	9,626	3,183	52,683	42,467	53,659	180,524

Data Summary (contd.)

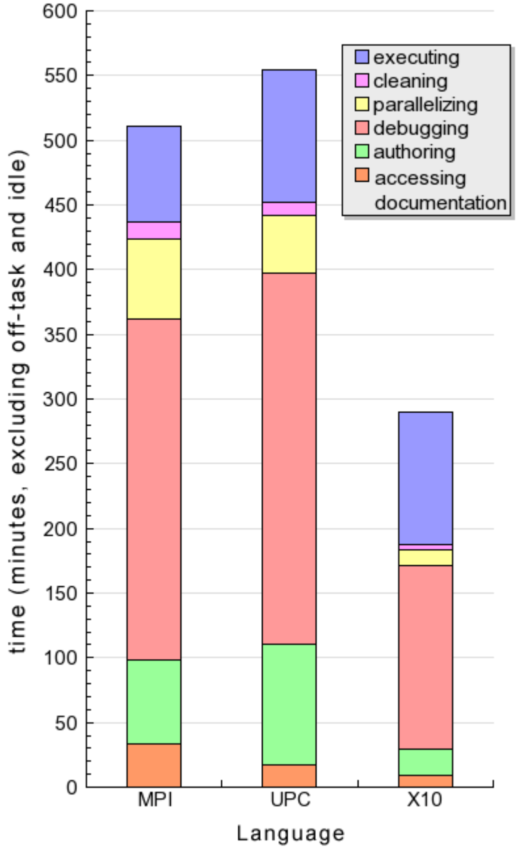


- Each thin vertical bar depicts 5 minutes of development time, colored by the distribution of activities within the interval.
- Development milestones bound intervals for statistical analysis:
  - begin/end task
  - begin/end development
  - first correct parallel output

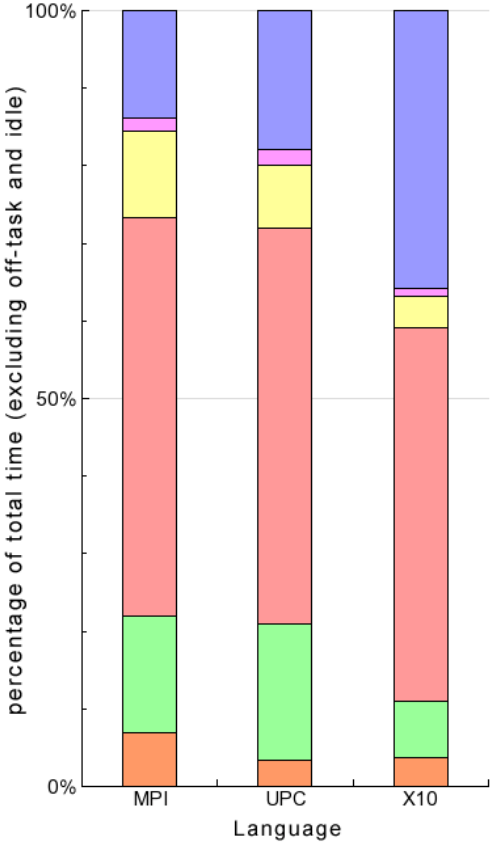
	MPI	UPC	X10
obtained correct parallel output	4	4	8
did not obtain correct parallel output	5	3	1
dropped out	0	2	0

Average Development Time by Language

Absolute Time



Percentage of Total



Comparing average development times between languages, several observations are clear:

- Average development time for subjects using X10 was significantly lower than that for subjects using UPC and MPI.
- The relative time debugging was approximately the same for all languages.
- X10 programmers spent relatively more time executing code and relatively less time authoring and tidying code.
- Subjects using MPI spent more time accessing documentation (tutorials were online; more documentation is available).
- A batch environment was used in this study --- use of an interactive environment like Eclipse will probably have a significant.impact on development time results

Outline

- Software challenges for multi-core systems
- X10 Programming Model and Language
- X10 Productivity Analysis
- **X10 Implementation**
- Conclusions

X10 Implementation Status

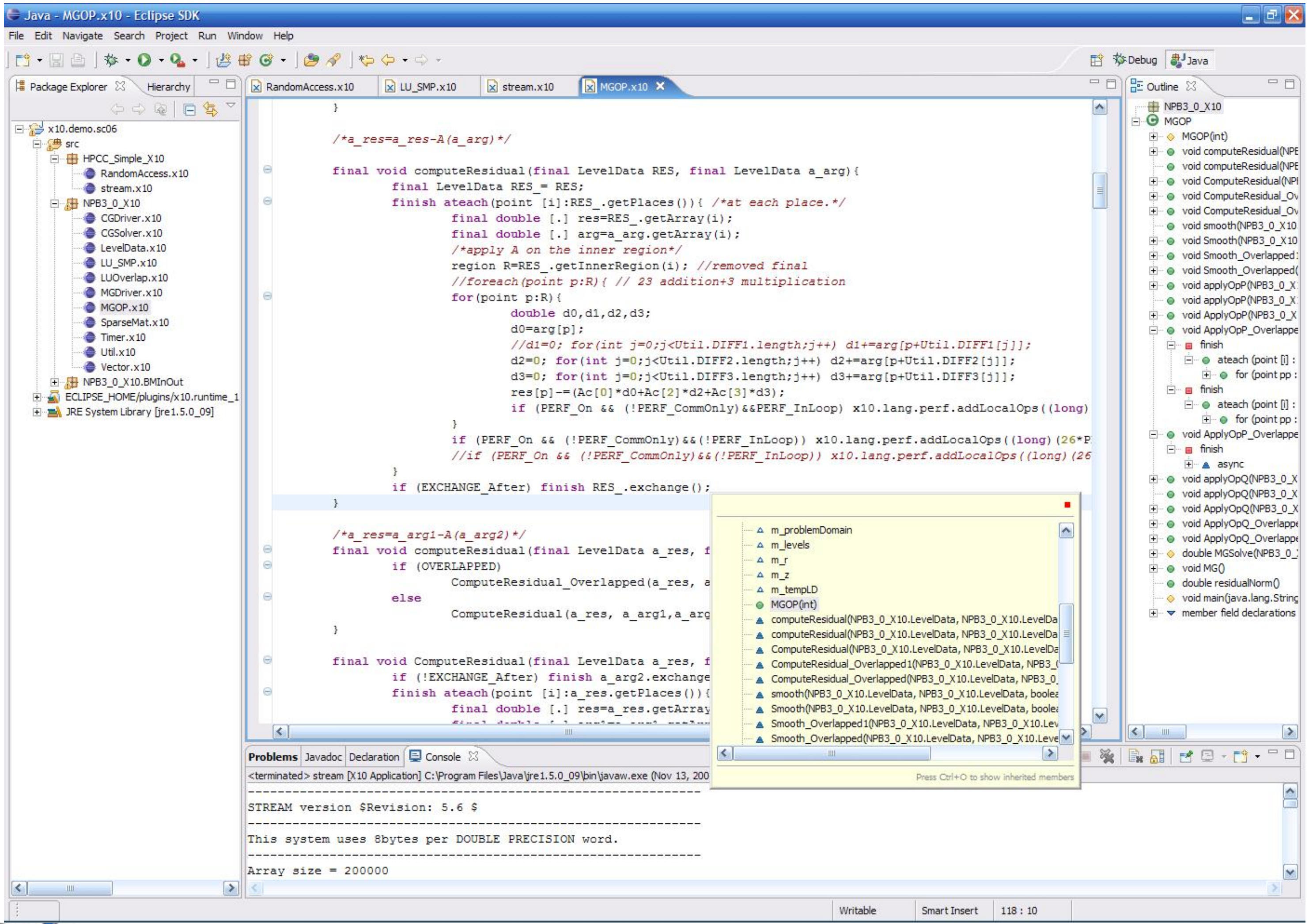
Open source project on sourceforge (x10.sf.net)

- **Reference X10 implementation on a single SMP**
  - CVS repository hosted on
  - Nightly regression tests (~ 600 unit tests)
  - X10 application set starting to grow
  - Basis for Optimized SMP Implementation
- **X10 Development Toolkit (X10DT)**
  - Eclipse tools with basic X10 language support
  - X10-specific *refactorings* in progress
    - Extract Async
    - Introduce atomic sections
  - Built on meta-tooling framework (SAFARI)

Efforts under way this year:

- **X10 Libraries for C/C++ users on cluster of SMP nodes**
- **C/C++ code generation from X10**
- **Static Analysis and Ahead-Of-Time Optimization**
  - Concurrency analysis
  - Optimization of BadPlaceException checks and redundant async/finish ops
  - Use of static analysis to enhance X10-specific refactorings

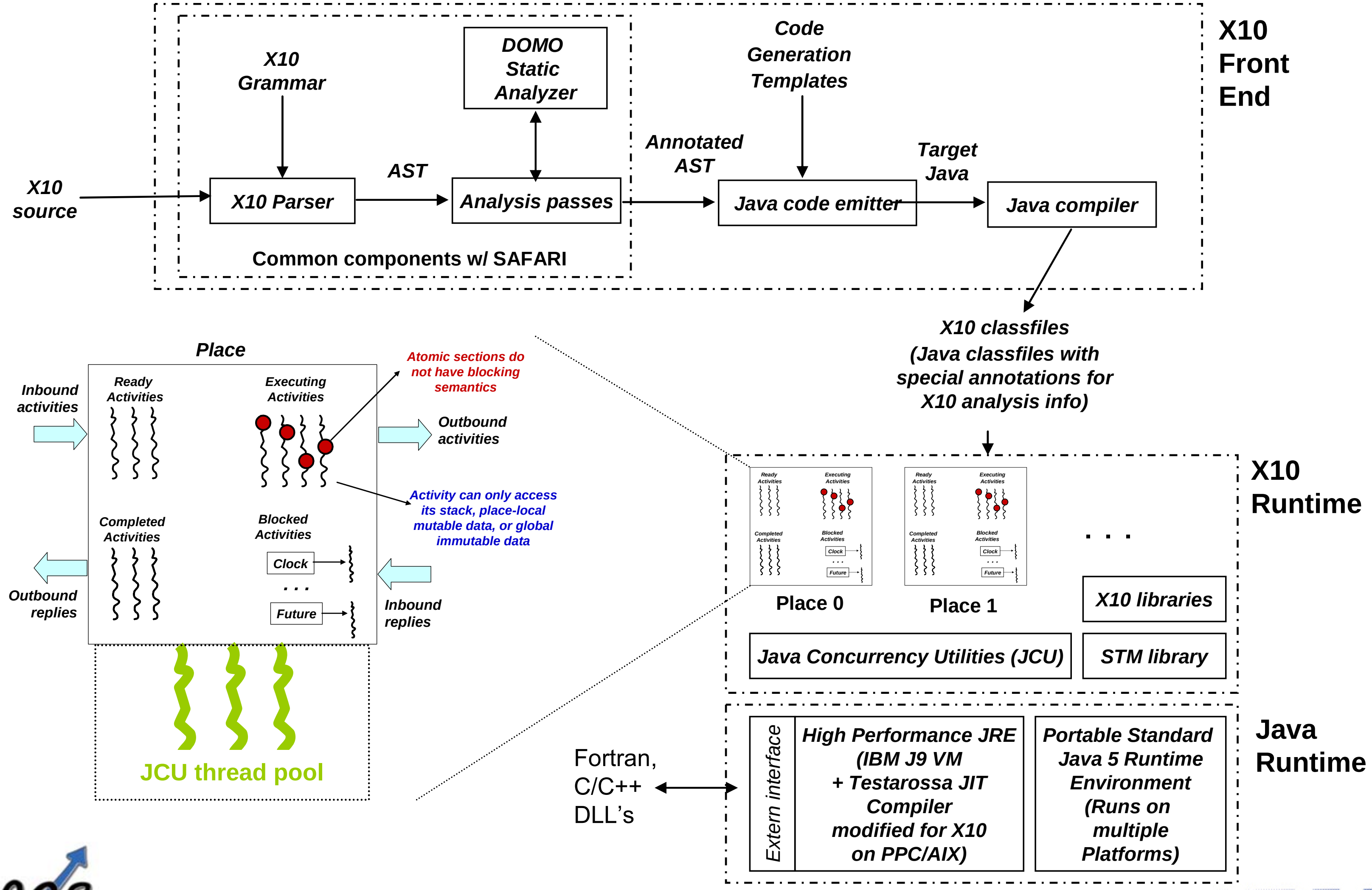
X10 Eclipse Development Toolkit



```
        }

        /*a_res=a_res-A(a_arg)*/

        final void computeResidual(final LevelData RES, final LevelData a_arg){
                final LevelData RES_= RES;
                finish ateach(point [i]:RES_.getPlaces()){ /*at each place.*/
                        final double [.] res=RES_.getArray(i);
                        final double [.] arg=a_arg.getArray(i);
                        /*apply A on the inner region*/
                        region R=RES_.getInnerRegion(i); //removed final
                        //foreach(point p:R){ // 23 addition+3 multiplication
                        for(point p:R){
                                double d0,d1,d2,d3;
                                d0=arg[p];
                                //d1=0; for(int j=0;j<Util.DIFF1.length;j++) d1+=arg[p+Util.DIFF1[j]];
                                d2=0; for(int j=0;j<Util.DIFF2.length;j++) d2+=arg[p+Util.DIFF2[j]];
                                d3=0; for(int j=0;j<Util.DIFF3.length;j++) d3+=arg[p+Util.DIFF3[j]];
                                res[p]-=(Ac[0]*d0+Ac[2]*d2+Ac[3]*d3);
                                if (PERF_On && (!PERF_CommOnly)&&PERF_InLoop) x10.lang.perf.addLocalOps((long)
                        }
                        if (PERF_On && (!PERF_CommOnly)&&(!PERF_InLoop)) x10.lang.perf.addLocalOps((long)(26*P
                        //if (PERF_On && (!PERF_CommOnly)&&(!PERF_InLoop)) x10.lang.perf.addLocalOps((long)(26
                }
                if (EXCHANGE_After) finish RES_.exchange();
        }

        /*a_res=a_arg1-A(a_arg2)*/
        final void computeResidual(final LevelData a_res, f
                if (OVERLAPPED)
                        ComputeResidual_Overlapped(a_res, a
                else
                        ComputeResidual(a_res, a_arg1,a_arg
        }

        final void ComputeResidual(final LevelData a_res, f
                if (!EXCHANGE_After) finish a_arg2.exchange
                finish ateach(point [i]:a_res.getPlaces()){
                        final double [.] res=a_res.getArray
```

```
<terminated> stream [X10 Application] C:\Program Files\Java\jre1.5.0_09\bin\javaw.exe (Nov 13, 200
-----------------------------------------------------------------
STREAM version $Revision: 5.6 $
-----------------------------------------------------------------
This system uses 8bytes per DOUBLE PRECISION word.
-----------------------------------------------------------------
Array size = 200000
```

Current Status: Multi-core SMP Implementation for X10



X10 Front End

X10 source

X10 Grammar

DOMO Static Analyzer

X10 Parser

AST

Analysis passes

Common components w/ SAFARI

Annotated AST

Code Generation Templates

Java code emitter

Target Java

Java compiler

X10 classfiles (Java classfiles with special annotations for X10 analysis info)

Place

Inbound activities

Ready Activities

Executing Activities

Atomic sections do not have blocking semantics

Outbound activities

Activity can only access its stack, place-local mutable data, or global immutable data

Completed Activities

Blocked Activities

Clock

. . .

Future

Outbound replies

Inbound replies

JCU thread pool

X10 Runtime

Place 0

Place 1

. . .

X10 libraries

Java Concurrency Utilities (JCU)

STM library

Java Runtime

Fortran, C/C++ DLL's

Extern interface

High Performance JRE (IBM J9 VM + Testarossa JIT Compiler modified for X10 on PPC/AIX)

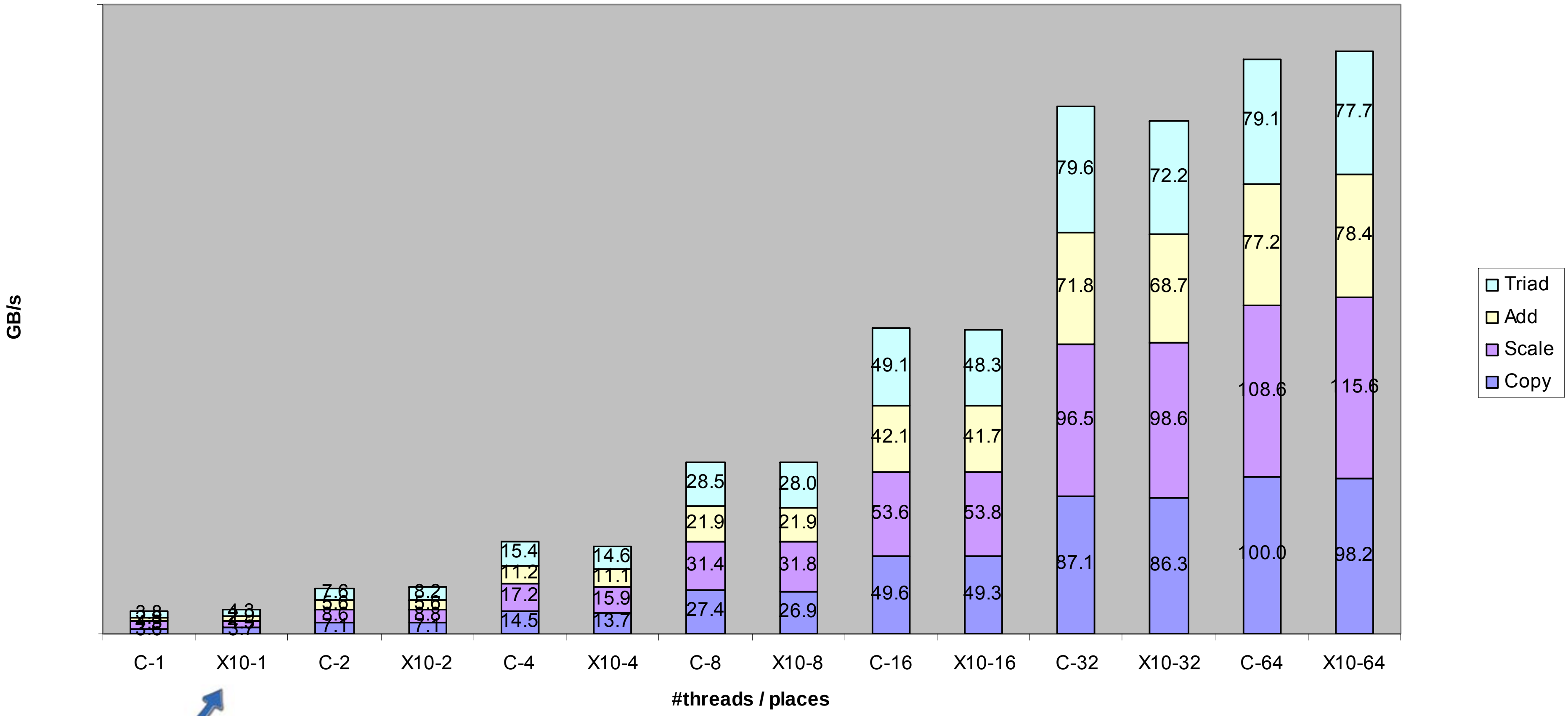
Portable Standard Java 5 Runtime Environment (Runs on multiple Platforms)

System Configuration used for Performance Results

- Hardware
  - STREAM (C/OpenMP & X10), RandomAccess (C/OpenMP & X10), FFT (X10)
    - 64-core POWER5+, p595+, 2.3 GHz, 512 GB (r28n01.pbm.ihost.com)
  - FFT (Cilk version)
    - 16-core POWER5+, p570, 1.9 GHz
  - All runs performed with page size = 4KB and SMT turned off
- Operating System
  - AIX v5.3
- Compiler
  - xlc v7.0.0.5 w/ -O3 option (also qsmp=omp for OpenMP compilation)
- X10
  - Dynamic compilation options: -J-Xjit:count=0,optLevel=veryHot
  - X10 activities use serial libraries written in C and linked with X10 runtime
  - Data size limitation: current X10 runtime is limited to a max heap size of 2GB
- All results reported are for runs that passed validation
  - Caveat: these results should *not* be treated as official benchmark measurements of the above systems

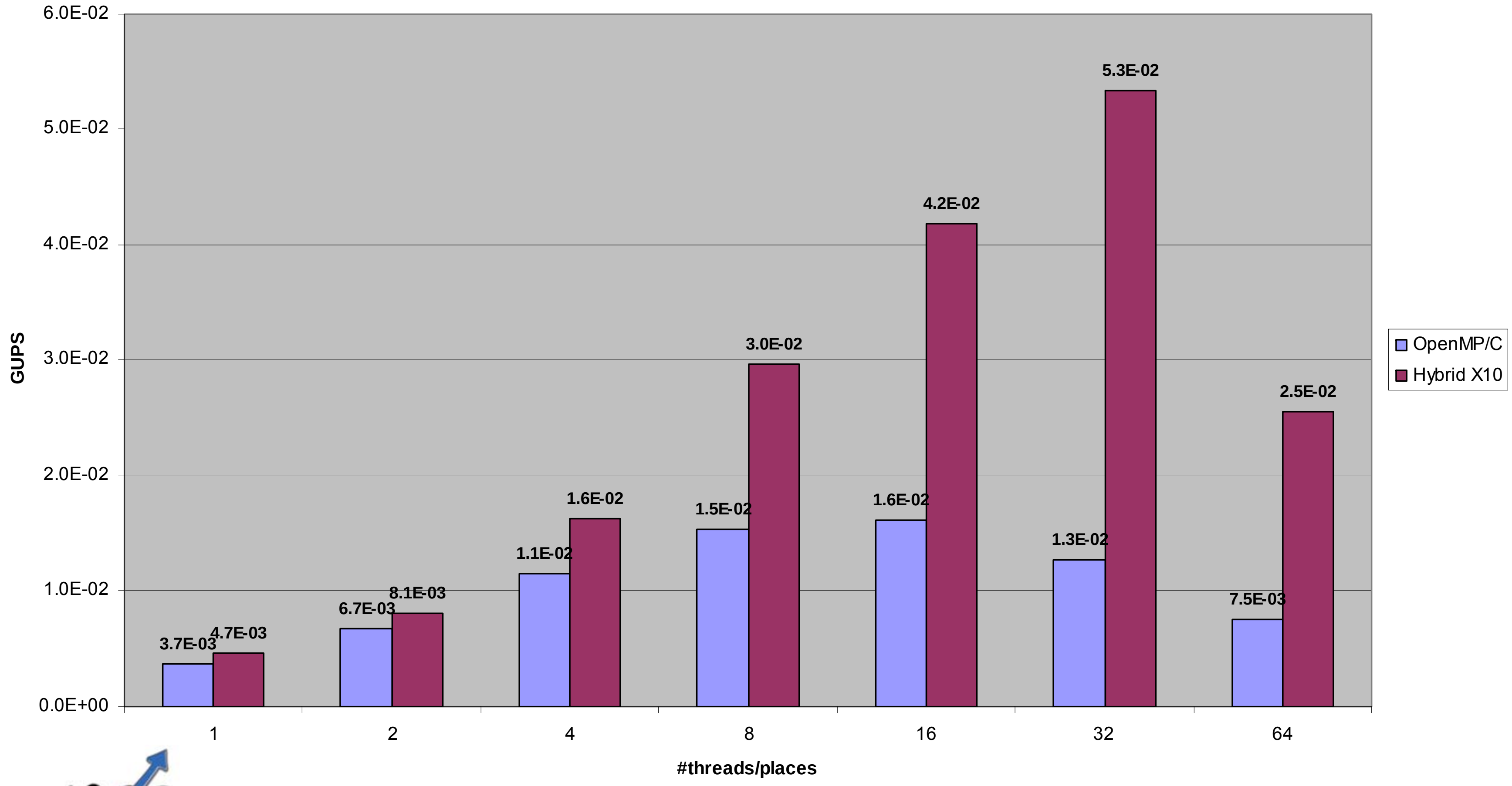
Performance Results for STREAM

Array size = 2^{26} elements
Combined memory for 3 arrays = 1.5GB

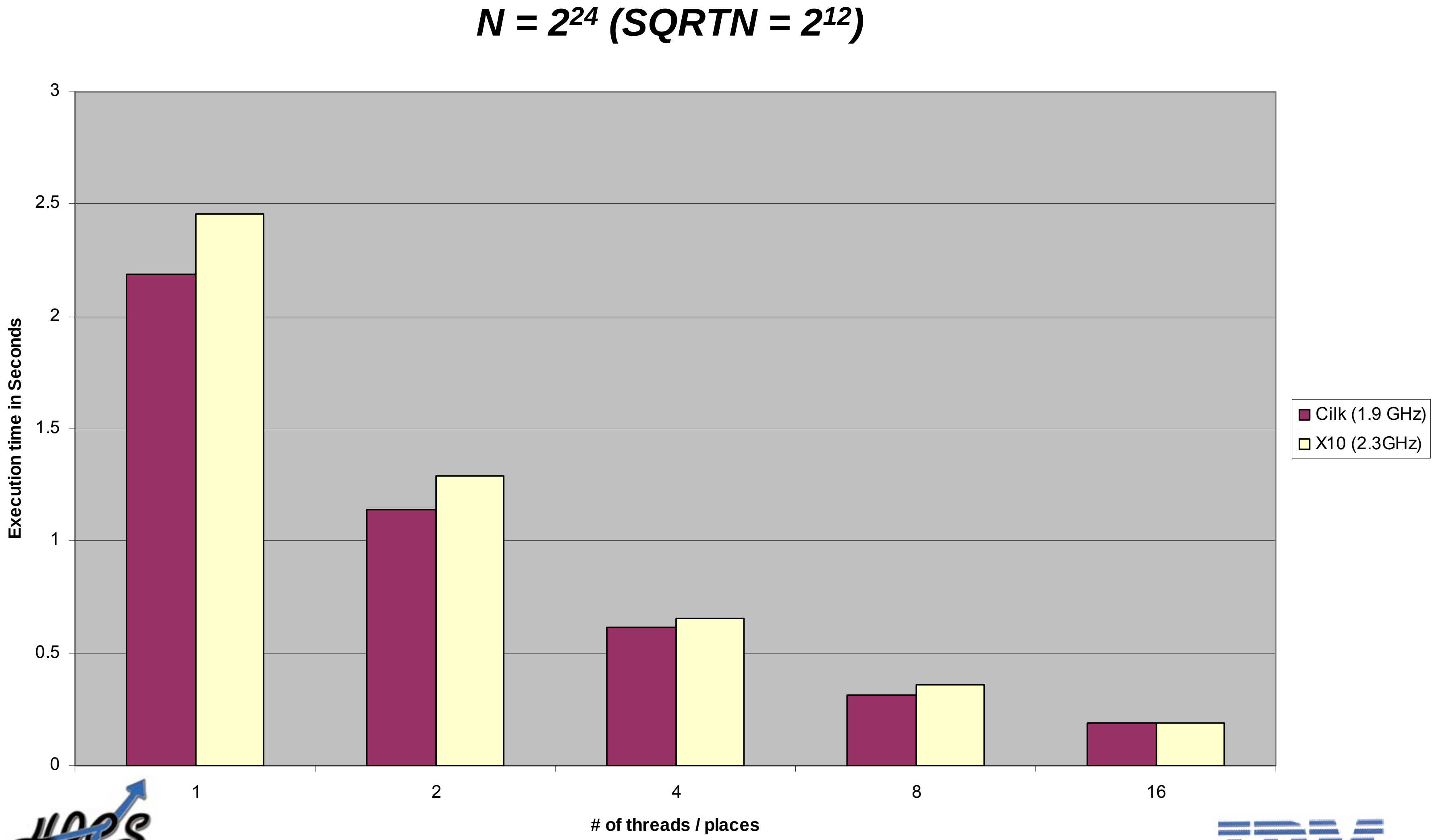


Performance Results for RandomAccess

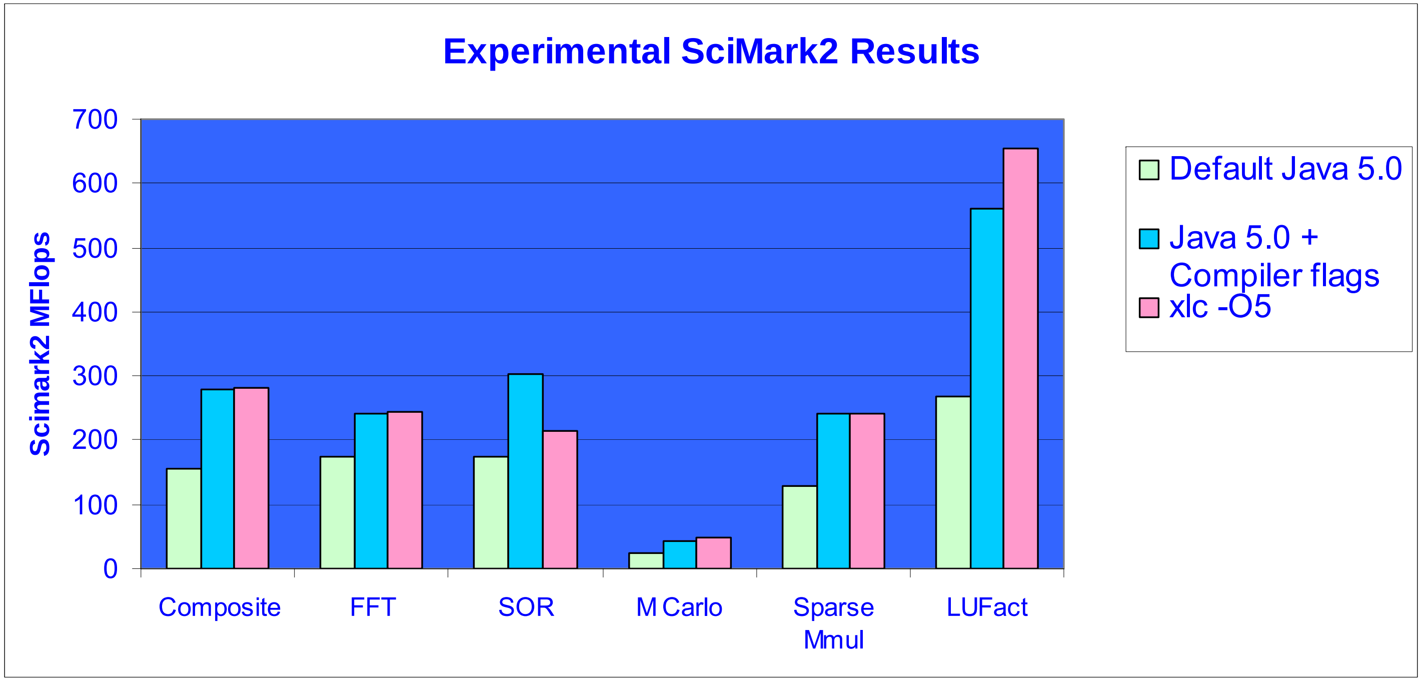
Array size = 1.8GB



Performance Results for FFT
(w/ memoized sine/cosine twiddle factors)



Studying the Java vs. C performance gap (1.1GHz Power4 system, xlc v7.0.0.1 xlc)



Details on "Compiler flags"

•Force higher optimization level on first execution of methods

•Enable generation of Fused Multiply-Add machine instructions

•Simulate X10 Static Analyzer's ability to remove most null- and bounds-checks by removing all such checks from selected methods.

•Also remove extra code and register allocation restrictions related to memory management (GC) in selected methods

Some Challenges in Optimization of X10 programs

- **Improving Single Activity Performance**
- **Analysis and optimization of explicitly parallel programs**
  - Proposed approach: use Parallel Program Graph (PPG) representation
- **Analysis and optimization of remote data accesses**
  - Proposed approach: perform data access aggregation and elimination using heap-based SSA framework
- **Optimized implementation of Atomic Sections**
  - Simple cases that can be supported by hardware
  - Analyzable atomic sections
  - General case
- **Load-balancing**
  - Dynamic, adaptive migration of places across nodes in deployment
- **Efficient invocation of components in other languages**
  - C, Fortran
- **Garbage collection across multiple places**

Outline

- Software challenges for multi-core systems
- X10 Programming Model and Language
- X10 Productivity Analysis
- X10 Implementation
- **Conclusions**

Related Work

- Single Program Multiple Data (SPMD) languages with Partition Global Address Space (PGAS)
  - Unified Parallel C, Co-Array Fortran, Titanium
  - X10 generalizes PGAS to a “threaded-PGAS” model (beyond SPMD)
- Hierarchical fork-join parallelism
  - Cilk (ultra-lightweight threads, work-stealing scheduling, …)
  - X10 generalizes Cilk by adding places, distributions, futures, …
- X10 has similarities with other languages in DARPA HPCS program --- Chapel (Cray) and Fortress (Sun) --- but there are also key differences
  - Chapel allows object migration and data redistribution, which could makes it harder to use for scalable parallelism (compared to X10)
  - Fortress has a major focus on new type system and user-viewable program representations (single-threaded advances that are complementary to X10)

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

Summary: Advantages of X10 Programming Model

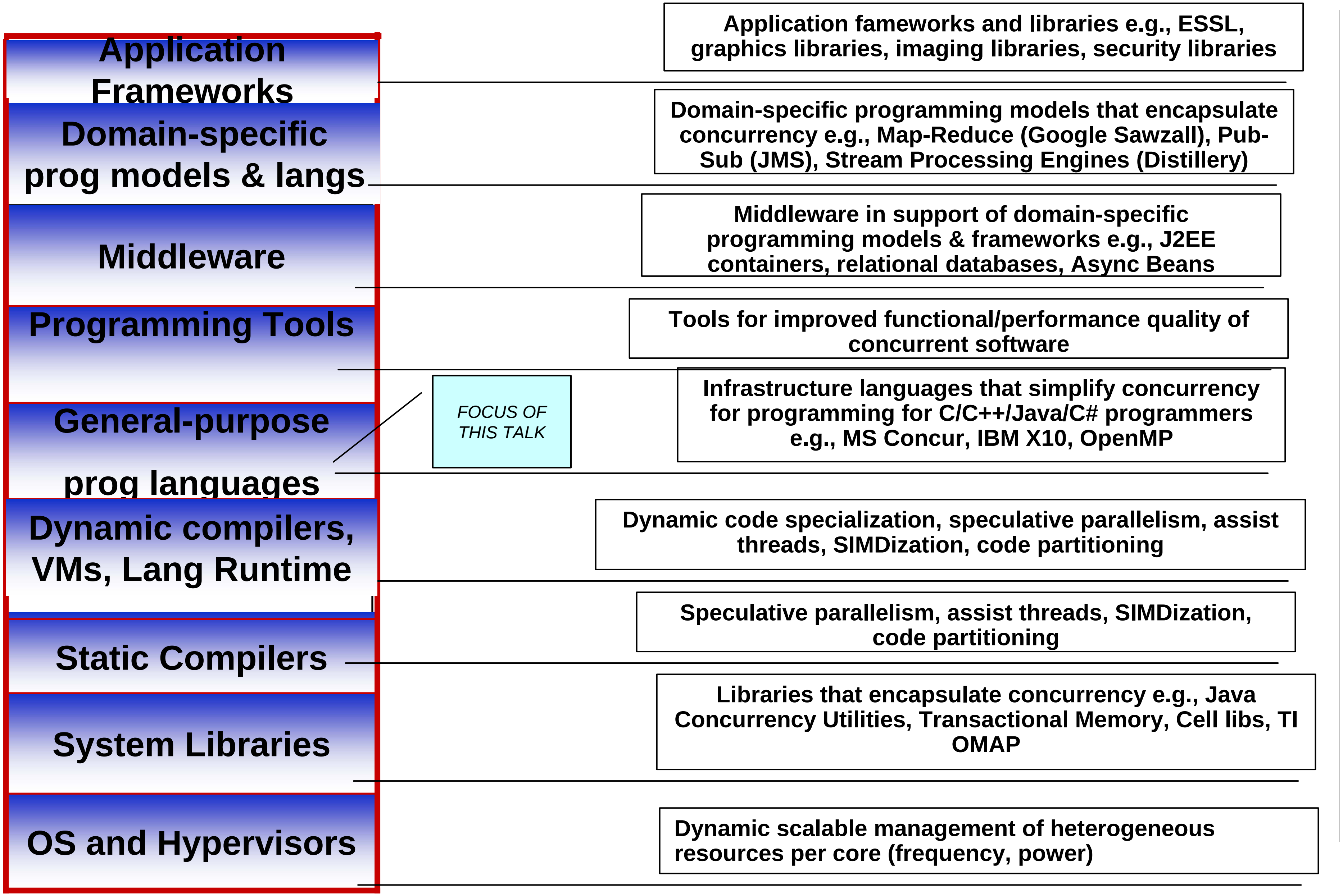
- Any program written with atomic, async, finish, foreach, ateach, and clock parallel constructs *will never deadlock*
  - future-force needs disciplined usage to guarantee absence of deadlock e.g., force should be performed by activity that created future (or an ancestor of that activity)
- Inter-node and intra-node parallelism integrated in a single model
- Remote activity invocation subsumes one-sided data transfer, remote atomic operations, active messages, . . .
- Finish subsumes point-to-point and team synchronization
- All remote data accesses are performed as activities ➔ rules for ordering of remote accesses follows simply from concurrency model
- Can be easily mapped to multiple levels of parallel hardware (SIMD, SMT, coprocessors, cache prefetch, SMP, clusters, …)

Conclusions and Future Work

- X10 programming model provides core concurrency and distribution constructs for new era of mainstream parallel processing
- Two primary opportunities for X10 adoption:
  - DARPA High Programming Language Systems
  - Mainstream language for multi-core
- We'd welcome collaboration on X10
  - Applications to evaluate X10 in different domains
  - X10 subprojects for different hardware platforms
    - Multi-core, Clusters, Co-processors, Grid
  - Participation in productivity experiments
  - Participation in X10 open source project on SourceForge (x10.sf.net)

BACKUP SLIDES START HERE

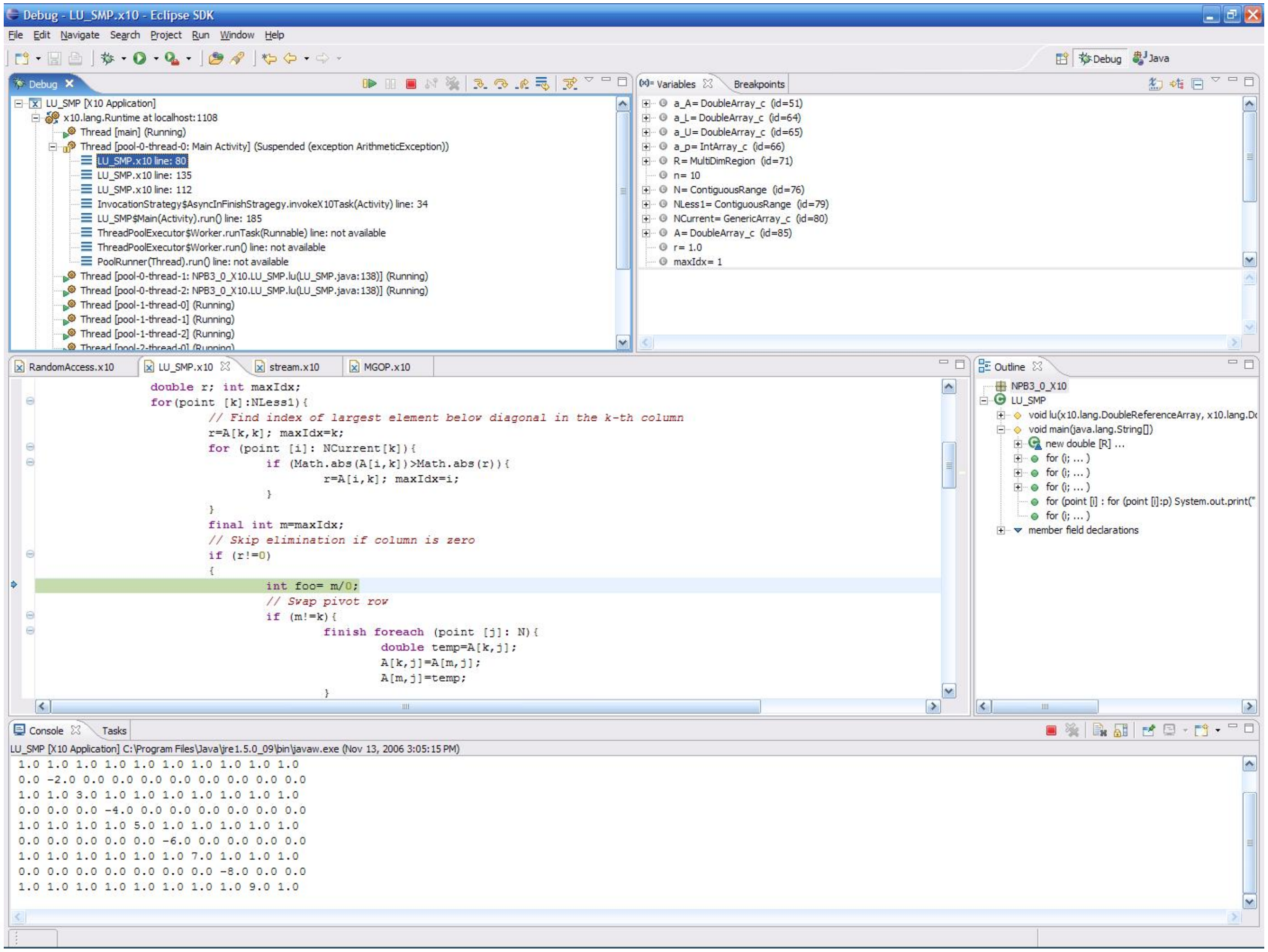
Need to look holistically at entire Software Stack for Multi-Core Enablement



Application Frameworks	Application fameworks and libraries e.g., ESSL, graphics libraries, imaging libraries, security libraries
Domain-specific prog models & langs	Domain-specific programming models that encapsulate concurrency e.g., Map-Reduce (Google Sawzall), Pub-Sub (JMS), Stream Processing Engines (Distillery)
Middleware	Middleware in support of domain-specific programming models & frameworks e.g., J2EE containers, relational databases, Async Beans
Programming Tools	Tools for improved functional/performance quality of concurrent software
General-purpose prog languages	Infrastructure languages that simplify concurrency for programming for C/C++/Java/C# programmers e.g., MS Concur, IBM X10, OpenMP
Dynamic compilers, VMs, Lang Runtime	Dynamic code specialization, speculative parallelism, assist threads, SIMDization, code partitioning
Static Compilers	Speculative parallelism, assist threads, SIMDization, code partitioning
System Libraries	Libraries that encapsulate concurrency e.g., Java Concurrency Utilities, Transactional Memory, Cell libs, TI OMAP
OS and Hypervisors	Dynamic scalable management of heterogeneous resources per core (frequency, power)

FOCUS OF THIS TALK

X10 Eclipse Debugging Toolkit



```
LU_SMP [X10 Application]
  x10.lang.Runtime at localhost:1108
    Thread [main] (Running)
    Thread [pool-0-thread-0: Main Activity] (Suspended (exception ArithmeticException))
      LU_SMP.x10 line: 80
      LU_SMP.x10 line: 135
      LU_SMP.x10 line: 112
      InvocationStrategy$AsyncInFinishStragegy.invokeX10Task(Activity) line: 34
      LU_SMP$Main(Activity).run() line: 185
      ThreadPoolExecutor$Worker.runTask(Runnable) line: not available
      ThreadPoolExecutor$Worker.run() line: not available
      PoolRunner(Thread).run() line: not available
    Thread [pool-0-thread-1: NPB3_0_X10.LU_SMP.lu(LU_SMP.java:138)] (Running)
    Thread [pool-0-thread-2: NPB3_0_X10.LU_SMP.lu(LU_SMP.java:138)] (Running)
    Thread [pool-1-thread-0] (Running)
    Thread [pool-1-thread-1] (Running)
    Thread [pool-1-thread-2] (Running)
    Thread [pool-2-thread-0] (Running)
```

```
a_A= DoubleArray_c  (id=51)
a_L= DoubleArray_c  (id=64)
a_U= DoubleArray_c  (id=65)
a_p= IntArray_c  (id=66)
R= MultiDimRegion  (id=71)
n= 10
N= ContiguousRange  (id=76)
NLess1= ContiguousRange  (id=79)
NCurrent= GenericArray_c  (id=80)
A= DoubleArray_c  (id=85)
r= 1.0
maxIdx= 1
```

```
double r; int maxIdx;
for(point [k]:NLess1){
        // Find index of largest element below diagonal in the k-th column
        r=A[k,k]; maxIdx=k;
        for (point [i]: NCurrent[k]){
                if (Math.abs(A[i,k])>Math.abs(r)){
                        r=A[i,k]; maxIdx=i;
                }
        }
        final int m=maxIdx;
        // Skip elimination if column is zero
        if (r!=0)
        {
                int foo= m/0;
                // Swap pivot row
                if (m!=k){
                        finish foreach (point [j]: N){
                                double temp=A[k,j];
                                A[k,j]=A[m,j];
                                A[m,j]=temp;
                        }
```

```
NPB3_0_X10
LU_SMP
  void lu(x10.lang.DoubleReferenceArray, x10.lang.Do
  void main(java.lang.String[])
    new double [R] ...
    for (i; ... )
    for (i; ... )
    for (i; ... )
    for (point [i] : for (point [i]:p) System.out.print("
    for (i; ... )
  member field declarations
```

LU_SMP [X10 Application] C:\Program Files\Java\jre1.5.0_09\bin\javaw.exe (Nov 13, 2006 3:05:15 PM)

```
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
0.0 -2.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 1.0 3.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
0.0 0.0 0.0 -4.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 1.0 1.0 1.0 5.0 1.0 1.0 1.0 1.0 1.0
0.0 0.0 0.0 0.0 0.0 -6.0 0.0 0.0 0.0 0.0
1.0 1.0 1.0 1.0 1.0 1.0 7.0 1.0 1.0 1.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 -8.0 0.0 0.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 9.0 1.0
```

STREAM

OpenMP / C version

```
#pragma omp parallel for
for (j=0; j<N; j++) {
   b[j] = scalar*c[j];
}
```

Hybrid X10 + Serial C version

```
finish ateach(point p : dist.factory.unique()) {
    final region myR = (D | here).region;
    scale(b,scalar,c,myR.rank(0).low(),myR.rank(0).high()+1);
}
```

STREAM

OpenMP / C version

```
#pragma omp parallel for
for (j=0; j<N; j++) {
   b[j] = scalar*c[j];
}
```

Traversing array region can be error-prone

Implicitly assumes Uniform Memory Access model (no distributed arrays)

SLOC counts are comparable

Hybrid X10 + Serial C version

```
finish ateach(point p : dist.factory.unique()) {
    final region myR = (D | here).region;
    scale(b,scalar,c,myR.rank(0).low(),myR.rank(0).high()+1);
}
```

Multi-place version designed to run unchanged on an SMP or a cluster

Restrict operator simplifies computation of local region

scale( ) is a sequential C function

RandomAccess

OpenMP / C version

```
#define NUPDATE (4 * TableSize)
for (i=0; i<NUPDATE/128; i++) {
#pragma omp parallel for
   for (j=0; j<128; j++) {
      ran[j] = (ran[j] << 1) ^ ((s64Int) ran[j] < 0 ? POLY : 0);
     Table[ran[j] & (TableSize-1)] ^= ran[j];
   }
}
```

Hybrid X10 + Serial C version

```
finish ateach(point p : dist.factory.unique()) {
 final region myR = (D | here).region;
 for (int i=0; i<(4 * TableSize)/W; i++) {
   innerLoop(Table,TableSize,ran,myR.rank(0).low(),myR.rank(0).high()+1);
 }
}
```

RandomAccess

OpenMP / C version

```
#define NUPDATE (4 * TableSize)
for (i=0; i<NUPDATE/128; i++) {
#pragma omp parallel for
   for (j=0; j<128; j++) {
      ran[j] = (ran[j] << 1) ^ ((s64Int) ran[j] < 0 ? POLY : 0);
     Table[ran[j] & (TableSize-1)] ^= ran[j];
   }
}
```

Inner parallel loop is a source of inefficiency in OpenMP version

SLOC counts are comparable

Multi-place version designed to run unchanged on an SMP or a cluster

Hybrid X10 + Serial C version

```
finish ateach(point p : dist.factory.unique()) {
 final region myR = (D | here).region;
 for (int i=0; i<(4 * TableSize)/W; i++) {
   innerLoop(Table,TableSize,ran,myR.rank(0).low(),myR.rank(0).high()+1);
 }
}
```

innerLoop() is a sequential C function

Restrict operator simplifies computation of local region

FFT: Transpose example

Cilk / C version (Recursive version)

```
#define SUB(A, i, j) (A)[(i)*SQRTN+(j)]

cilk void transpose(fftw_complex *A, int n)
{
     if (n > 1) {
       int n2 = n/2;
       spawn transpose(A, n2);
       spawn transpose(&SUB(A, n2, n2), n-n2);
       spawn transpose_and_swap(A, 0, n2, n2, n);
     } else {
       /* 1x1 transpose is a NOP */
     }
}
```

Implicit sync at function boundary

Hybrid X10 + Serial C version (Non-recursive version)

```
int nBlocks = SQRTN / bSize;
int p = 0;
finish for (int r = 0; r < nBlocks; ++r) {
    for (int c = r; c < nBlocks; ++c) { // Triangular loop
        final int topLefta_r = (bSize * r);
        final int topLefta_c = (bSize * c);
        final int topLeftb_r = (bSize * c);
        final int topLeftb_c = (bSize * r);
           async (place.factory.place(p++))
              transpose_and_swap(A, topLefta_r, topLefta_c, topLeftb_r, topLeftb_c, bSize);
     }
}
```

“finish” operator is used to wait for termination of all subactivities (async’s)

transpose_and_swap( ) is a sequential C function