
Microsoft® Research
Faculty Summit
10
YEAR ANNIVERSARY

Microsoft Research

Universal Parallel Computing Research Center at Illinois

Making parallel programming synonymous with programming

Wen-mei Hwu
Co-Director
UPCRC Illinois

The UPCRC@ Illinois Team

Microsoft Research

Microsoft Research

UPCRC Illinois

background

Microsoft Research

PARALLEL@ILLINOIS

Illiac

UPCRC Illinois
Universal Parallel Computing Research Center

NVIDIA
CUDA Center of Excellence

Illinois Cloud Computing Testbed

Extreme Scale Computing

BLUE WATERS
SUSTAINED PETASCALE COMPUTING

OpenSPARC
Center of Excellence

Moore's Law Pre 2004

Microsoft Research

- Number of transistors per chip doubles every 18 months
- Performance of single thread increases
- New generation hardware provides better user experience on existing applications or support new applications that cannot run on old hardware
- People buy new PC every three years

Moore's Law Post 2004

Microsoft Research

- Number of transistors per chip doubles every 18 months
- Thread performance does not improve; number of cores per chip doubles
  - power/clock constraints and diminishing returns on new microprocessor features
- New generation hardware provides better user experience on existing application or support new applications that cannot run on old hardware – *only if these applications run in parallel and scale automatically*
- Parallel Software is essential to maintaining the current business model of chip and system vendors

Goals

Microsoft Research

- Create opportunity
  - New client applications that require high performance and can leverage high levels of parallelism
- Create SW to exploit opportunity
  - Languages, tools, environments, processes that enable the large number of client application programmers to develop good parallel code
- Create HW to exploit opportunity
  - Architectures that can scale to 100's of cores and provide best use of silicon a decade from now

Microsoft Research

UPCRC Illinois

applications

Applications Strategy

Microsoft Research

- Identify application types that
  - are likely to execute on clients
  - require much more performance than now available on a client
  - can run in parallel
- Develop enabling parallel code (core libraries, application prototypes) for such application types
  - hard to identify the killer app; easier to work in its “vicinity”
  - doing so gives us an understanding of the apps requirements; demonstrates feasibility
  - and leads to the creation of reusable software

Client Application Drivers

Microsoft Research

- Intelligent user interfaces require high performance on the client side (!)
  - graphics, vision, NLP
- Private information will be kept on the client side (?)
  - concerns for privacy and security
  - fewer efficiencies to be achieved on server side, because of limited sharing
  - NLP, data mining, search
- High-availability services require client performance and adaptation
  - Provide “best answer”, given current connectivity
  - Adaptive applications (NLP)
- More powerful client reduces app development time
  - Games, browser

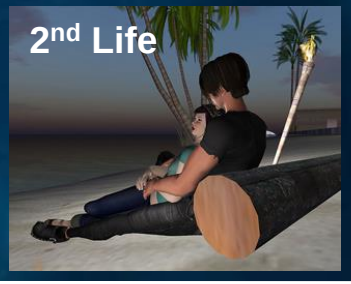
Graphics – Motivation

Microsoft Research

Game Engineering
Faster, Cheaper

Multicore
Visual Apps

Online
Social
Spaces
More
Realistic



Realism

(Time, $$$)

(Precomputation)

Flexibility

(John Hart)

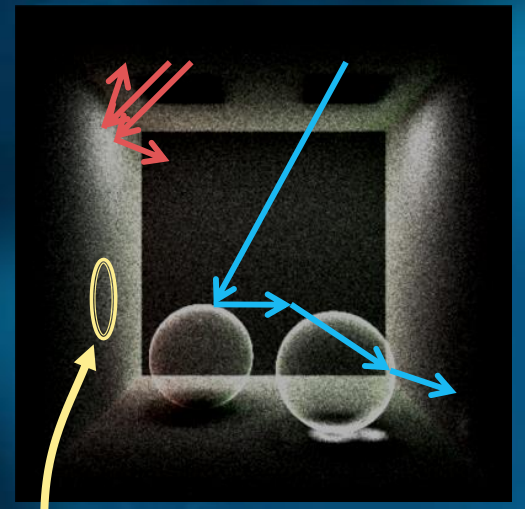
ParKD

Microsoft Research

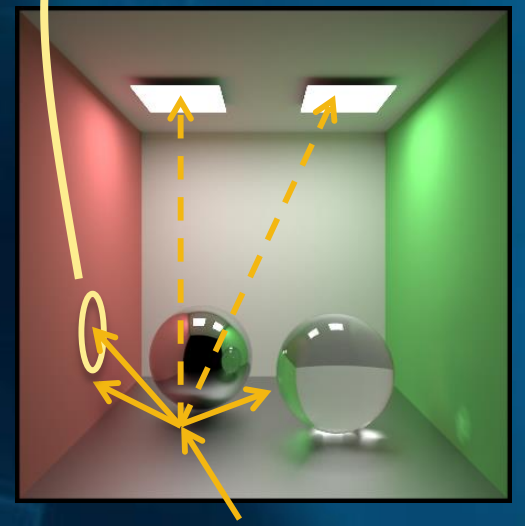
Objective: build parallel library with efficient spatial data structures to accelerate graphics and physical simulation

- Ray triangle intersection
- Photon→ray gathering
- 5-D ray classification/bundling
- Point→surface reconstruction
- Collision detection
- Also vision, machine learning

Photon Mapping scatters light particles into scene



Ray Tracing gathers light particles along lines of sight



Issues

Microsoft Research

- Simple, Direct Ray Tracing algorithms work well on GPUs, but do not provide good quality images
- Global Illumination algorithms are more irregular and do not map well on GPUs
- Different data structures and algorithms perform better on different images
- GPU is harder to program

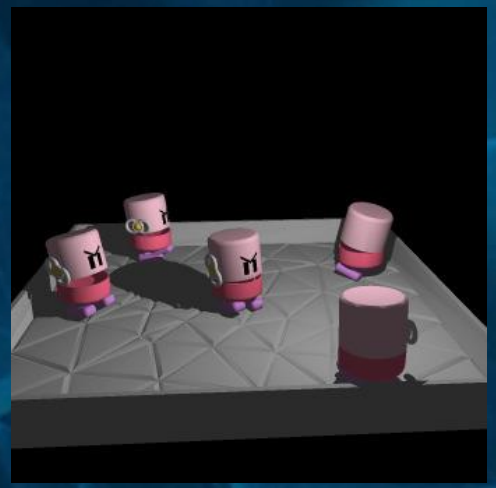
CPU vs. GPU Hierarchies

Microsoft Research

- CPU better than GPU on organizing hierarchical data
  - GPU lacks cache and stack
  - GPU needs synchronous control flow

4-core CPU: 5.84.fps
192-core GPU: 6.40 fps

4-core CPU: 23.5 fps
192-core GPU: 32.0 fps



M. Shevtsov, A. Soupikov & A. Kapustin.
Highly Parallel Fast KD-Tree Construction for Interactive Ray Tracing of Dynamic Scenes.
Eurographics 2007.

K. Zhou, Q. Hou, R. Wang & B. Guo.
Real-Time KD-Tree Construction on Graphics Hardware.
SIGGRAPH Asia 2008.

Programmer Productivity

Microsoft Research

- Tim Sweeney on impact of specialized hardware on programming time:
  - “Anything over 2x is uneconomical for software companies”
- Implemented straightforward parallel kd-tree build
  - Full quality SAH computation
  - Based on TBB
  - Nested parallelism
  - Programmer simplicity

Multithread:	**2x**
PS3 Cell:	**5x**
GPGPU:	**10x +**

“The Future of Games Programming,” Need for Speed, 22 April 2009

Image & Video Processing

Microsoft Research

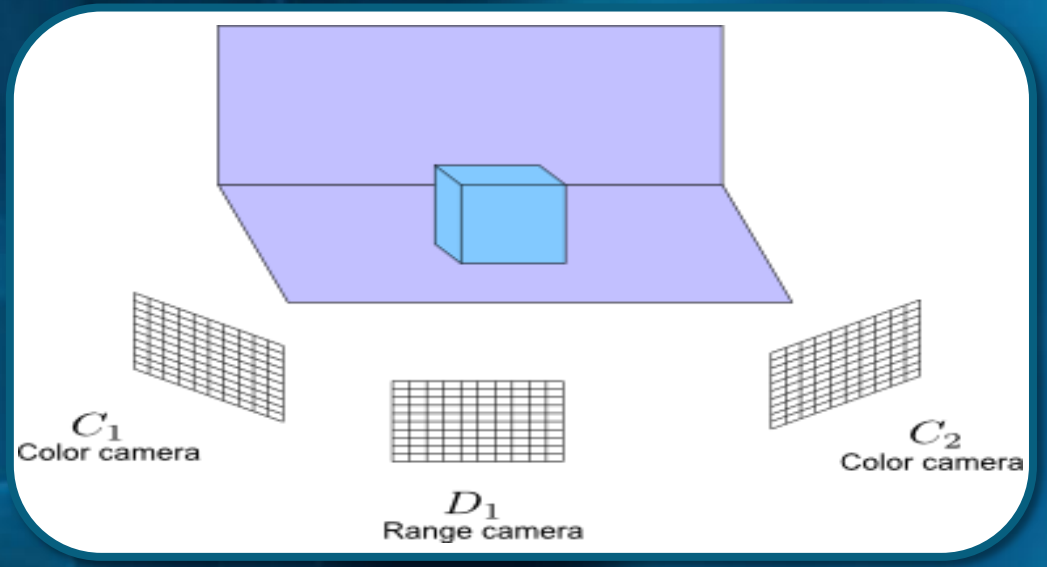
- Objective: Build a parallelized toolkit for image-based rendering and video processing
- 2 On-going Projects:
  - Depth Image-Based Rendering (Q. Nguyen & D. Kubacki)
  - Video Event Detection (D. Lin & M. Dikmen)

Depth Image-Based Rendering

Microsoft Research

Input: video from color and depth cameras at arbitrary locations.
Output: generated video from arbitrary viewpoints.

- Applications:
  - Image & video synthesis/fusion
  - 3D and free-viewpoint video
  - Telepresence
  - Enhanced recognition
- Issues:
  - Need for speed: real-time operation
  - Computation done at client (point of display)
  - Quality, robustness increase computational demand



Mapping to GPGPU architecture

Microsoft Research

- The algorithm is consciously developed with techniques that have high degree of locality (e.g. bilateral filtering, Sobel operator, block-based search)
- Preliminary result:
  - Naïve CUDA-based implementation for a part of the algorithm
  - Hardware platform: Intel Core2 Duo E8400 3.0GHz and NVIDIA GeForce 9800GT
  - Speedup of ~200x

COMPARISON IN THE RUNNING TIME OF THE DEPTH PROPAGATION OCCLUSION REMOVAL, AND DCBF STEPS

Mode	Frame rate (in fps)	Time (in msec)
Sequential	0.0878	11,389
Parallel	24.37	41.03

NLP Motivating Example:
Comprehension

A process that maintains and updates a collection of propositions about the state of affairs.

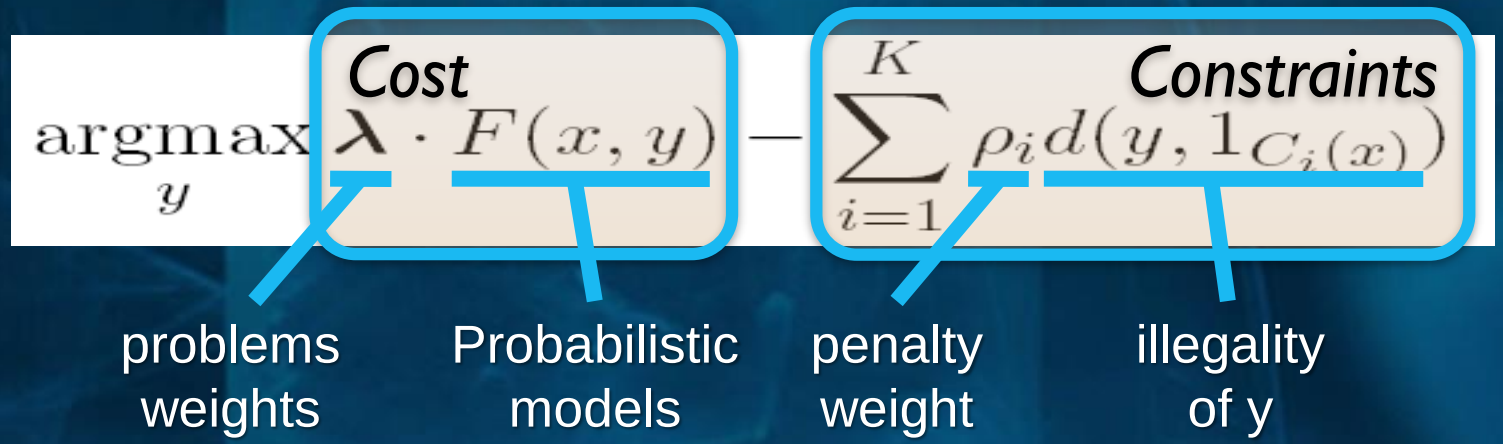
(ENGLAND, June, 1989) - Christopher Robin is alive and well. He lives in England. He is the same person that you read about in the book, Winnie the Pooh. As a boy, Chris lived in a pretty home called Cotchfield Farm. When Chris was three years old, his father wrote a poem about him. The poem was printed in a magazine for others to read. Mr. Robin then wrote a book. He made up a fairy tale land where Chris lived. His friends were animals. There was a bear called Winnie the Pooh. There was also an owl and a young pig, called a piglet. All the animals were stuffed toys that Chris owned. Mr. Robin made them come to life with his words. The places in the story were all near Cotchfield Farm. Winnie the Pooh was written in 1925. Children still love to read about Christopher Robin and his animal friends. Most people don't know he is a real person who is grown now. He has written two books of his own. They tell what it is like to be famous.

1. Christopher Robin was born in England.
2. Winnie the Pooh is a title of a book.
3. Christopher Robin's dad was an author.
4. Christopher Robin must have been at least 64.

Multiple context sensitive disambiguation problems are solved (via machine learning methods) and global decisions are derived as constrained optimization over these.

Vision and Natural Language
Algorithmic Cores

Microsoft Research



- Statistical Machine Learning algorithms in Vision and Natural Language are Optimization Based (e.g., conjugate gradients, integer linear programming; search and sampling
- Feature Extraction and Inference require graph operations and sub-isomorphism
- In addition: significant data parallelism needs to be exploited, e.g., for massive indexing of processed data

Parallelizing NLP Applications

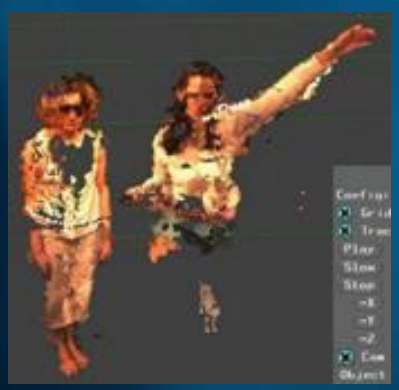
Microsoft Research

- **Applications**: coded in LBJ, a language for learned functions
  - Parts of Speech Tagger (which part-of-speech a word is?)
  - Co-Reference Finder (which words refer to the same entity?)
  - key components for intelligent spell-checking
- **Parallelism opportunities**: independent computations at the sentence (POS Tagger) or document level (CoRef)
- **Results**:
  - parallelized applications using lightweight Java tasks
  - almost linear speedups up to 4 cores
- **Future Work:**
  - parallelize Semantic Role Labeler
  - document common NLP patterns
  - semi-automate the parallelization process

Tele-Immersion

Microsoft Research

- Multiple 4-camera arrays convert scene into scattered 3-D data points from different viewpoints
- Points sent over net and reconstructed remotely
- Applications in collaborative engineering design, telemedicine, gaming, etc.
- Currently requires system of > 10 workstations – would want to integrate in one small box



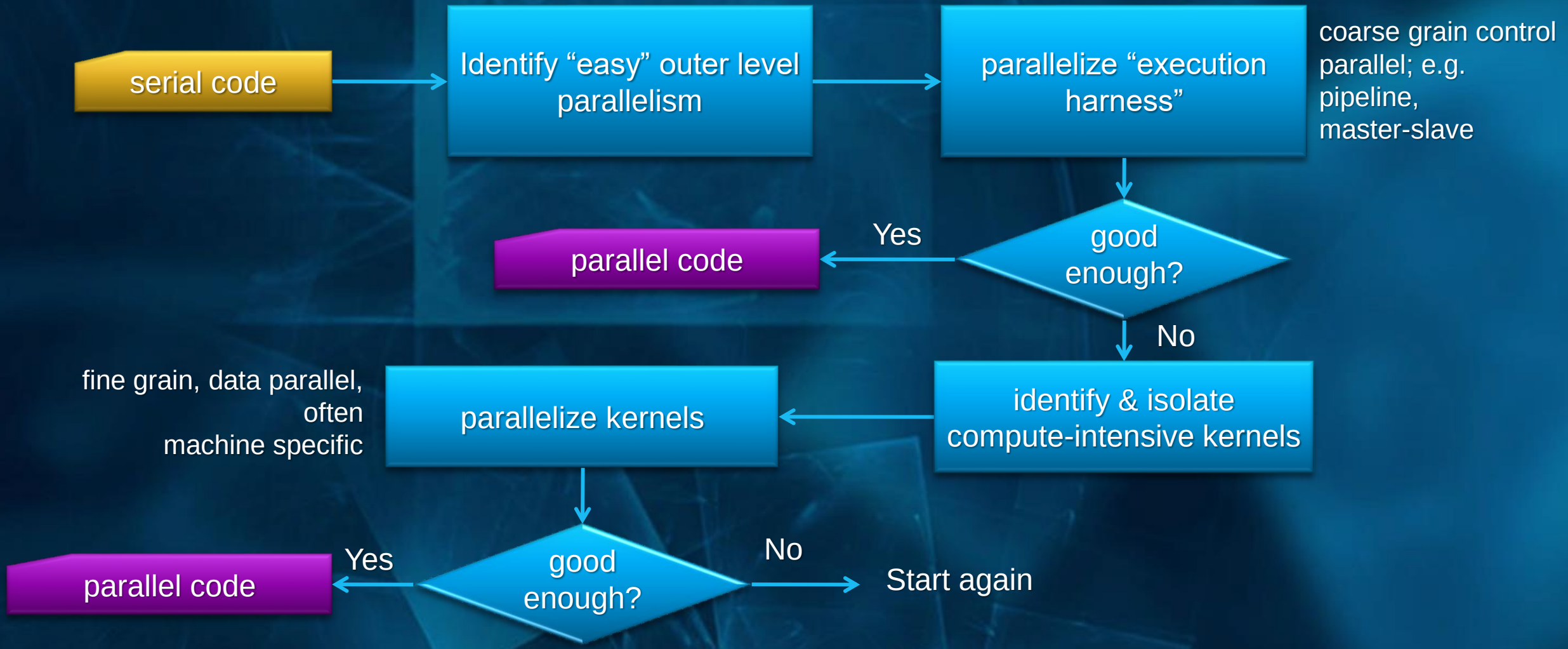
Microsoft Research

UPCRC Illinois

parallel programming

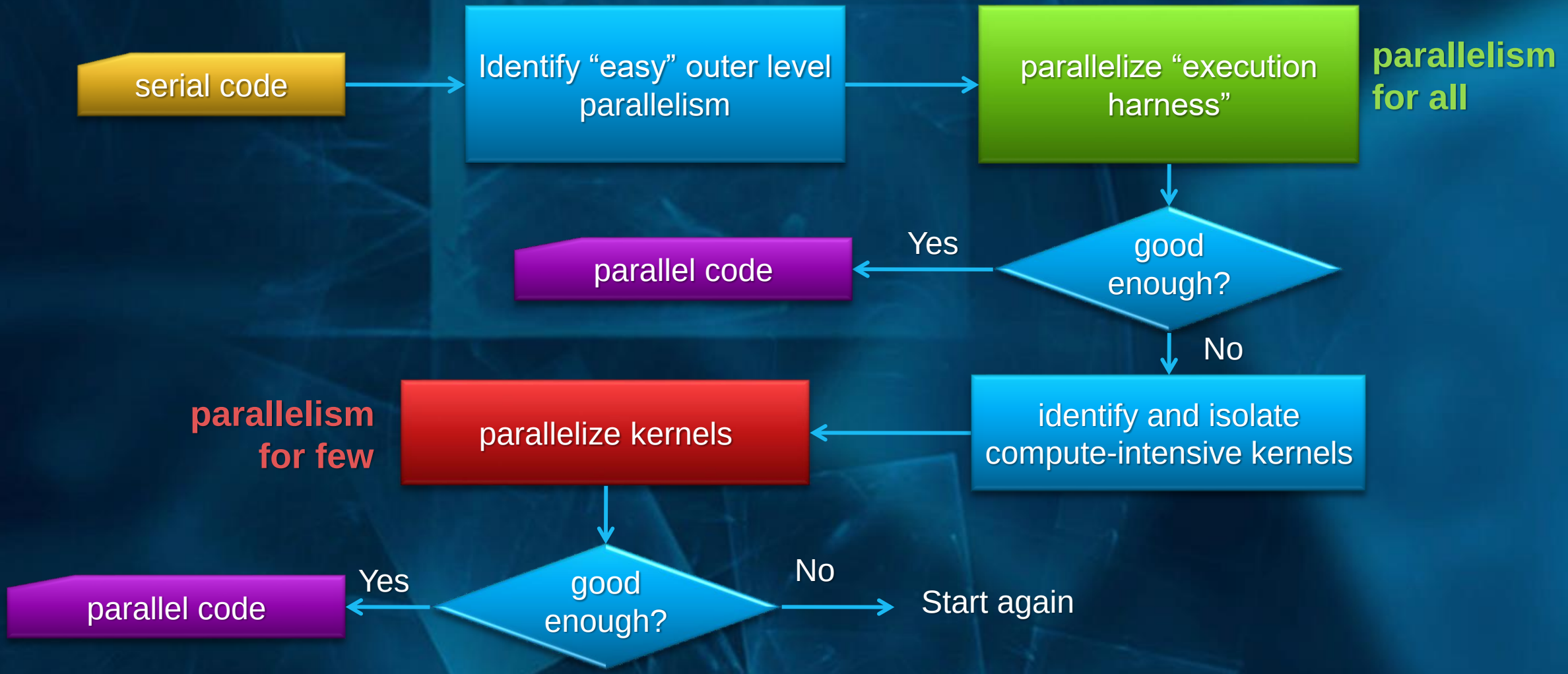
Schematic Workflow

Microsoft Research



Division of Labor

Microsoft Research



Parallel Programming vs. Concurrent Programming

Microsoft Research

- Concurrent programming: concurrency is part of application semantics
  - reactive code, e.g. system code, GUI, online transaction processing: managing concurrent inputs or concurrent outputs
  - focused on concurrency management and synchronization: mutual exclusion, atomic transactions, etc.
- Parallel programming: concurrency is only needed for performance
  - transformational code, e.g. scientific computing, streaming: speeding up computation, while external interactions are serial.
  - focused on deterministic programs and consumer-producer synchronization
- Multi-core creates significant new demand for parallel programming, but no significant new demand for concurrent programming

Alternative Hypotheses

Microsoft Research

1. Parallel programming is inherently hard
   - multi-core will “fail”
2. Parallel programming, *as currently done,* is hard; it should not be much harder than sequential programming, if one uses the right methodologies, tools and environments
   - multi-core can succeed, but will require significant investments in SW

- We have a strong bias toward 2
- We have some objective reasons to believe that 2 is right

Arguments for 2

Microsoft Research

- Some forms of parallelism are routinely used:
  - vector operations (APL/Fortran90), domain-specific dataflow languages (Cantata/Verilog/Simulink), concurrent object languages (Squeak, Seaside, Croquet, Scratch)…
- Work on shared memory programming has been almost exclusively focused on (hard) concurrent programming
- Investments on SW to support parallel programming have been minuscule and focused on expert programmers and “one-size-fits-all” solutions

What Would Make Parallel Programming Easier?

Microsoft Research

- *Isolation*: effect of the execution of a module does not depend on other concurrently executing modules
- *Concurrency safety:* Isolation is enforced by language
- *Determinism:* program execution is, by default, deterministic; nondeterminism, if needed, is introduced via explicit notation
- *Sequential semantics:* sequential operational model, with simple correspondence between lexical execution state and dynamic execution state
- *Parallel performance model:* work, depth

Why is Determinism Good?

Microsoft Research

- Testing is much easier (single execution per input)
- Makes debugging much easier
- Easy to understand: execution equivalent to sequential execution
- Easy to incrementally parallelize code
- Can use current tools and methodologies for program development
- Nondeterminism seldom (if ever) needed for performance parallelism
  - with exceptions such as chaotic relaxation, branch and bound, some parallel graph algorithms
  - when needed, can be highly constrained (limited number of nondeterministic choices)

Example

```
for s in S do f(s);
```

- S is ordered set (or ordered iteration domain)
- iterations executed in order

```
forall s in S do f(s);
```

- Iterations are expected to be independent
- Safety: an exception occurs (compile time? runtime?) otherwise
- Execution deterministic

`forall` has same semantic as **`for`**, but parallel performance model

Goal

Microsoft Research

- Notation that “makes parallel programming easier” (concurrent safe, deterministic by default, sequential semantics, parallel performance model)
- Thesis (UPCRC research): can have such a notation for OO languages that
  - is not too painful on user
  - is not too restrictive in expressiveness
  - is not leaving too much performance on the table

Is it Already Done?

Microsoft Research

- Auto-parallelization:
  - ✓ determinism, sequential semantics, safety
  - ✗ fails too frequently; no parallel performance model
- Functional languages (e.g., NESL):
  - ✓ determinism, isolation, safety, parallel performance model
  - ✗ limited expressiveness and/or inadequate performance: cannot handle concurrent updates to shared object

Is it Already Done? (cont.)

Microsoft Research

- Conventional parallel language (e.g., OpenMP), augmented with run-time checks (speculative execution)
  - safety, determinism, parallel performance model
  - ✗ not clear can be done efficiently
  - ✗ still need to test for concurrency bugs

Key Research Question

Microsoft Research

How to enforce safety in a language with shared references to mutable objects

- compile-time enforcement: programmer provides additional information on access patterns
  - ✓ more performing, provides better feedback to programmer
  - × may require extensive program annotations
- run-time enforcement: run-time & hardware detect conflicting, non-synchronized accesses to shared memory
  - × requires HW support for performance
  - ✓ less coding effort (but possibly more debugging effort)
  - × testing is harder

Key Religious Question

Microsoft Research

Should parallelism be implicit or explicit?

1. Write sequential program
   - add annotations that provide sufficient information for compiler to safely parallelize
   - ✓ easier port strategy
   - × performance model not well-defined
2. Write parallel program
   - add annotations that provide sufficient information for compiler to ensure that program is race-free
   - ✓ well-defined performance model
   - × harder port strategy

- No fundamental differences between two approaches if parallelism is deterministic:
  - (2) parallel constructs (e.g. forall) has same semantic as sequential construct (e.g. for); vs.
  - (1) parallel construct results from parallelization of sequential construct

Microsoft Research

UPCRC Illinois

a sequential endian approach

Gluon – specification Information to Enable Parallelization (W. M-Hwu)

Microsoft Research

(W M Hwu)

- Developers specify pivotal information at function boundaries
  - Heap data object shapes and sizes
  - Object access behavioral guarantees
  - Some can be derived from global analyses but others are practically infeasible to extract from source code.
- Compilers leverage the information to
  - Expose and transform parallelism
  - Perform code and data layout transformations for locality without harming source code portability

Gluon Programmer Specification Example

No side effect on input elements of d, only 1 element accessed

```
struct data {
   float x; floa   ; float z;
};

int cal_bin(st uct data *a,
            struct data *b) {
1. __spec(*a: r, (data)[1]);
2. __spec(*b: r, (data)[1]);
3. __spec(ret_v: range(0,SZ));

 int bin = .  ;  use *a and
*b*/
 return(bin);
}
```

Range of return value: hist safe to privatize

d is a 1-D array of length given by the first argument.

```
int *tpacf(int len, s uct data *d) {
4. __spec(d: r, (int)[len]);

    int *hist = malloc(SZ*sizeof(int));
5. __spec(hist: (int)[SZ]);

    for (i=0; i < len; i++) {
       for (j = 0; j < len; j++) {

6.       int bin = cal_bi  &d[i],&d[j]);

7.       hist[bin] += 1;
       }
    }
}
```

This is parallelizable nested loop

Gluon Research

Microsoft Research

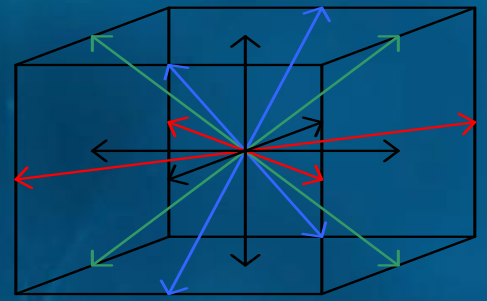
1. Show that such notations are sufficient to generate good parallel code (in many cases).
2. Facilitate the creation of such notations (code refactoring).

Gluon Enables More Than Parallelism
LBM Example

Microsoft Research

Array of Structure: [z][y][x][e]

Address F(z, y, x, e) = z * |Y| * |X| * |E| + y * |X| * |E| + x * |E| + e



Structure of Array: [e][z][y][x]

Address F(z, y, x, e) = e * |Z| * |Y| * |X|+ z * |Y| * |X| +y * |X| + x

4X faster than AoS on GTX280

Will also have significant effect on SSE

The Best Layout Is Neither SoA nor AoS

Microsoft Research

- Tiled Array of Structure, using lower bits in x and y indices, i.e. $x_{3:0}$ and $y_{3:0}$ as lowest dimensions: $[z][y_{31:4}][x_{31:4}][e][y_{3:0}][x_{3:0}]$
  - $F(z, y, x, e) = z * \lceil |Y|/2^4 \rceil * \lceil |X|/2^4 \rceil * |E| * 2^4 * 2^4 + y_{31:4} * \lceil |X|/2^4 \rceil * |E| * 2^4 * 2^4 + x_{31:4} * |E| * 2^4 * 2^4 + e * 2^4 * 2^4 + y_{3:0} * 2^4 + x_{3:0}$
- 6.4X faster than AoS, 1.6X faster than SoA on GTX280:
  - Better utilization of data by calculation of neighboring cells
  - This is a scalable layout: same layout works for very large objects.

We are automating this transformation based on Gluon.

Microsoft Research

UPCRC Illinois

a parallel endian approach

Deterministic Parallel Java
(Vikram Adve)

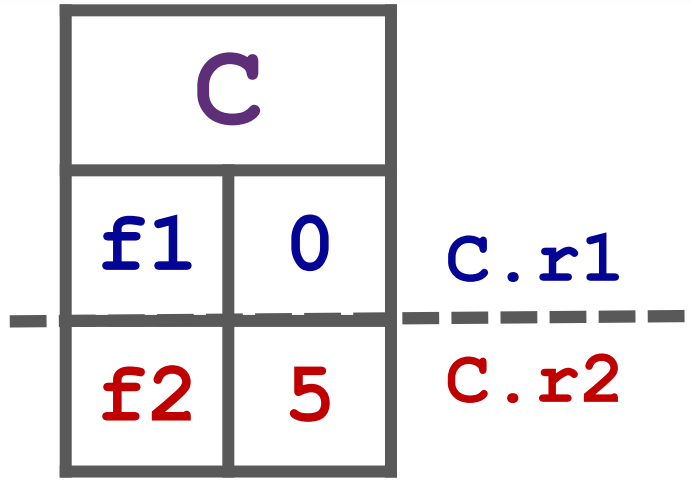
Microsoft Research

- Extension to Java (and soon, C++)
  - Compatible with existing Java (C++) sofware
- Explicit Parallel Control
- Explicit type and effect system
  - Recursive parallelism on linked data structures
  - Array computations
  - Safe use of parallel object-oriented frameworks
- Ongoing work
  - Run-time support for greater expressivity
  - Safe, explicitly non-deterministic algorithms

http://dpj.cs.uiuc.edu/

Example: Regions and Effects

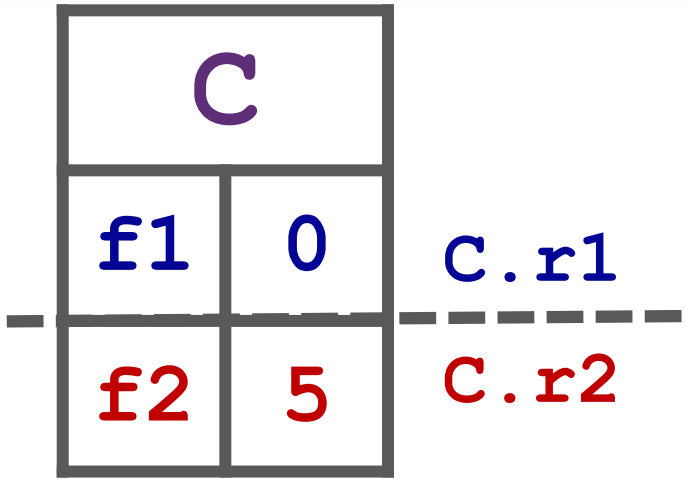
```
class C {
  region r1, r2;
  int f1 in r1;
  int f2 in r2;
  void m1(int x) writes r1 {f1 = x;}
  void m2(int y) writes r2 {f2 = y;}
  void m3(int x, int y){
    cobegin {
      m1(x);
      m2(y);
    }
  }
}
```



Partitioning the heap

Example: Regions and Effects

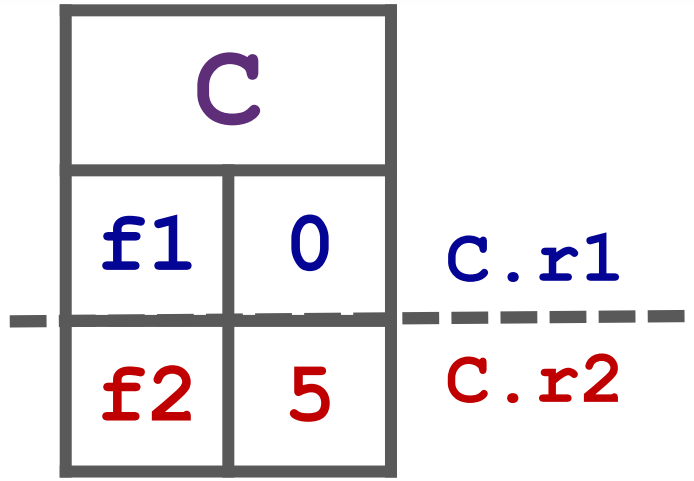
```
class C {
  region r1, r2;
  int f1 in r1;
  int f2 in r2;
  void m1(int x) writes r1 {f1 = x;}
  void m2(int y) writes r2 {f2 = y;}
  void m3(int x, int y){
    cobegin {
      m1(x);
      m2(y);
    }
  }
}
```



Summarizing method effects

Example: Regions and Effects

```
class C {
  region r1, r2;
  int f1 in r1;
  int f2 in r2;
  void m1(int x) writes r1 {f1 = x;}
  void m2(int y) writes r2 {f2 = y;}
  void m3(int x, int y){
    cobegin {
      m1(x); // Effect = writes r1
      m2(y); // Effect = writes r2
    }
  }
}
```



Expressing parallelism

Expressiveness

Microsoft Research

- Can handle recursive partitioning (e.g., concurrent operations on trees)
- Can handle partitioning of linear structures
- Can handle standard parallel algorithms (merge-sort, Barnes-Hut…)

- Can invoke trusted libraries with deterministic overall behavior, even if those are implemented in conventional language
- Refactoring tools can help Java->DPJ port

DPJizer: Porting Java to DPJ

An interactive Eclipse Plug-in (Danny Dig)

Microsoft Research

Input: Java program + region declarations + `foreach / cobegin`

Output: Same program + effect summaries on methods

Features:

- Fully automatic effect inference
- Supports all major features of DPJ
  - Region parameters; regions for recursive data; array regions
- Interactive GUI using Eclipse mechanisms

Next step: Infer regions automatically as well

Microsoft Research

UPCRC Illinois

and something for the old church

Support for Traditionalists

Microsoft Research

- Evolution to new, safer languages will be slow; need to support programmers using traditional multi-threading support (C++, C#, Java, etc.)
  - Refactoring
  - Program testing (detection of concurrency bugs)

Finding concurrency bugs: Attacking the interleaving explosion problem

(Madhusudan Parthasarathy, Darko Marinov)

Microsoft Research

- A key problem for testing concurrent programs:
  - Even on a *single* test, a concurrent program exhibits myriad executions…. Interleaving explosion!
- Question: How do we effectively search this extremely large space of interleavings?
  - Eg. CHESS – checks all interleavings with a few context switches
- Our idea:
  - Don't look randomly at schedules for errors!
  - Examine bug databases for interleaving patterns that lead to bugs
  - Test a program and obtain one execution; use this trace to predict other executions that correspond to the interleaving pattern
  - Using the test harness, check if alternate execution leads to an error

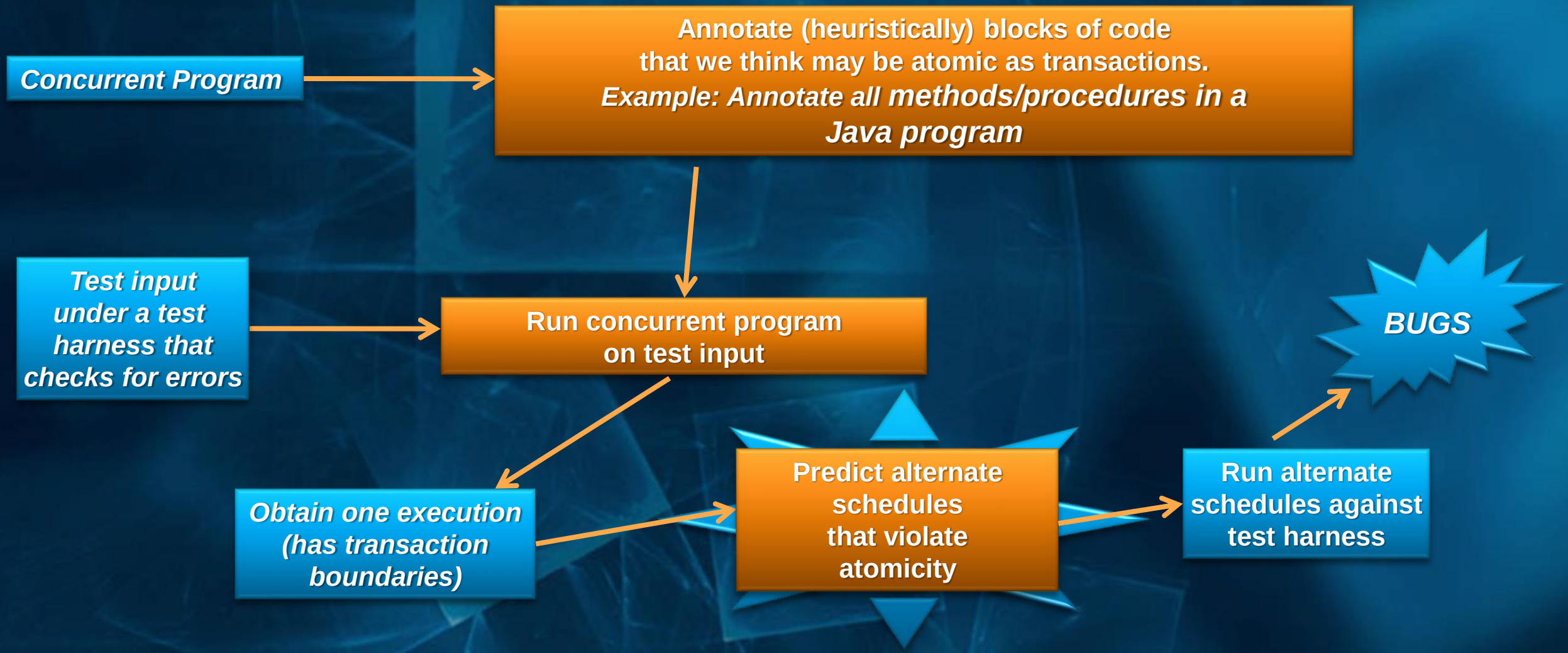
A Common Interleaving Pattern:
Violation of Atomicity

Microsoft Research

- Atomicity
  - A local piece of code often wishes to access shared data without (real) interference from other threads
  - Extremely common intention; violation leads to many errors
  - In bug studies, we and others (Lu-YYZ-et al) have found that majority of errors (~70%) are due to atomicity violations
- Questions
  - Can we predict from one run alternate runs that violate atomicity?
  - If so, can we execute those non-atomic schedules and test them against a test harness?

Finding Bugs Using Atomicity Violations

Microsoft Research



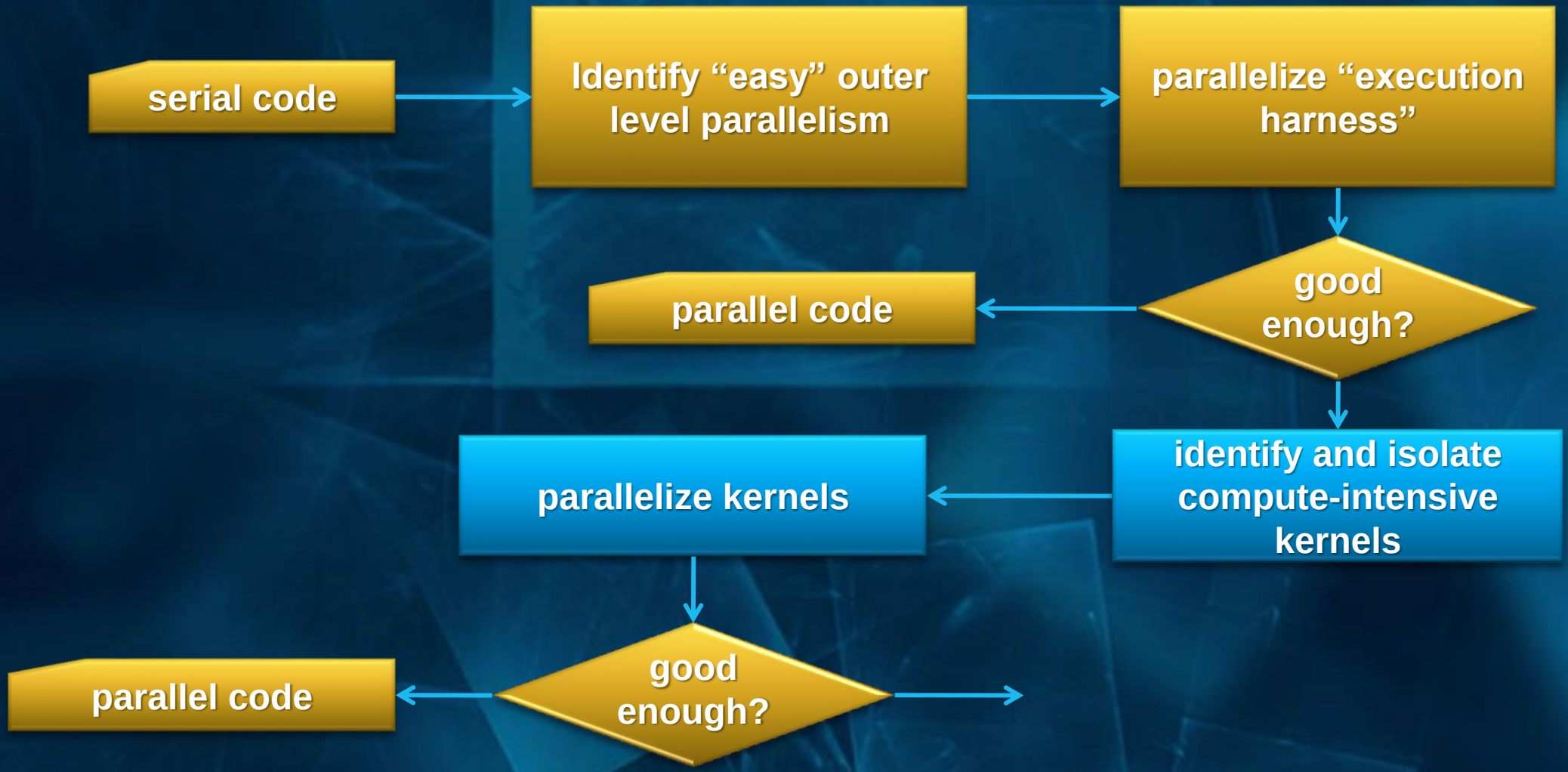
Current and future work

Microsoft Research

- Algorithms for efficiently prediction
  - Tractable solution if no synchronization
  - No practical solution likely for locking programs [TACAS'09]
  - However, for nested locking programs, efficient prediction algorithms exist. Atomicity violations involving 2 threads and 1 variable can be found in linear time! [CAV'09]
- Practical implementation of testing tool
  - Ongoing: Bytecode transformation to test alternate schedules
  - Challenges: Monitoring pgms; scheduling alternate executions; ensuring predicted path is executed
- New projects: Testing environment for concurrent programs; unit testing; regression testing; incremental testing

Libraries

Microsoft Research



Data Parallel Kernels
(David Padua)

Microsoft Research

- Operations on dense and sparse arrays, sets and other objects
  - Encapsulate parallelism
  - **Raise the level of abstraction**
- Due to encapsulation
  - Parallel programs based exclusively on these operations can be analyzed for correctness as if they were sequential
  - Portability across classes of machines achieved by re-implementing kernels
- Due to higher level of abstraction
  - More opportunities for optimization such as selection of data structures (e.g. to represent sparse arrays)

Data Parallel Kernels (cont.)

Microsoft Research

- In the notation we developed, blocks of objects (tiles) can be directly manipulated to control locality, communication, and granularity (hierarchically tiled arrays)
- This representation has proven quite effective to develop readable and efficient codes
- Ongoing work include
  - Identification of useful operations for symbolic parallel computing
  - Development of automatic optimization strategies (compilers and autotuning)

Patterns: Encyclopedia of Parallel Programming Patterns

(Ralph Johnson)

Microsoft Research

- A good start
  - ***Parallel Programming Patterns*** **by Mattson, Sanders and Massingill**
- But more patterns needed
  - **domain-specific: graphics, particle codes, …**
  - **technology-specific: tiling (cache management), streaming**
- It takes a village…
  - **document patterns used by students and others at Illinois**
  - **work with others, including PPP authors**
  - **annual workshop (ParaPlop), organized with Berkeley**
  - **teach and promote patterns**

Microsoft Research

UPCRC Illinois

architecture

Issues

Microsoft Research

- How doe scale and provide better support for current software?
  - Bulk (J. Torrellas)
- How do we take advantage of and better support new parallel languages?
  - DeNovo (S. Adve)
- How do we scale to >1000 cores?
  - Rigel (S. Patel)

The Bulk Multicore (Josep Torrellas)

Microsoft Research

General-purpose hardware architecture for programmability

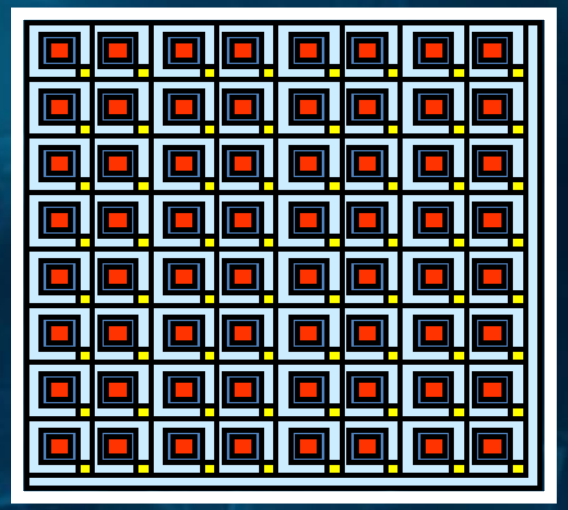
- Novel scalable cache-coherent shared-memory (signatures and chunks)
  - Relieves programmer/runtime from managing shared data
- High-performance sequential memory consistency
  - Provides a more SW-friendly environment
- HW primitives for low-overhead program development and debug (data-race detection, deterministic replay, address disambiguation)
  - Helps reduce the chance of parallel programming errors
  - Overhead low enough to be “on” during production runs

http://iacoma.cs.uiuc.edu/bulkmulticore.pdf

Idea in Bulk Multicore

Microsoft Research

- Idea: Eliminate the commit of individual instructions at a time
- Mechanism:
  - By default, processors commit chunks of instructions at a time (e.g. 2,000 dynamic instr)
  - Chunks execute atomically and in isolation (using buffering and undo)
  - Memory effects of chunks summarized in HW address signatures
- Advantages over current:
  - Higher programmability
  - Higher performance
  - Simpler processor hardware



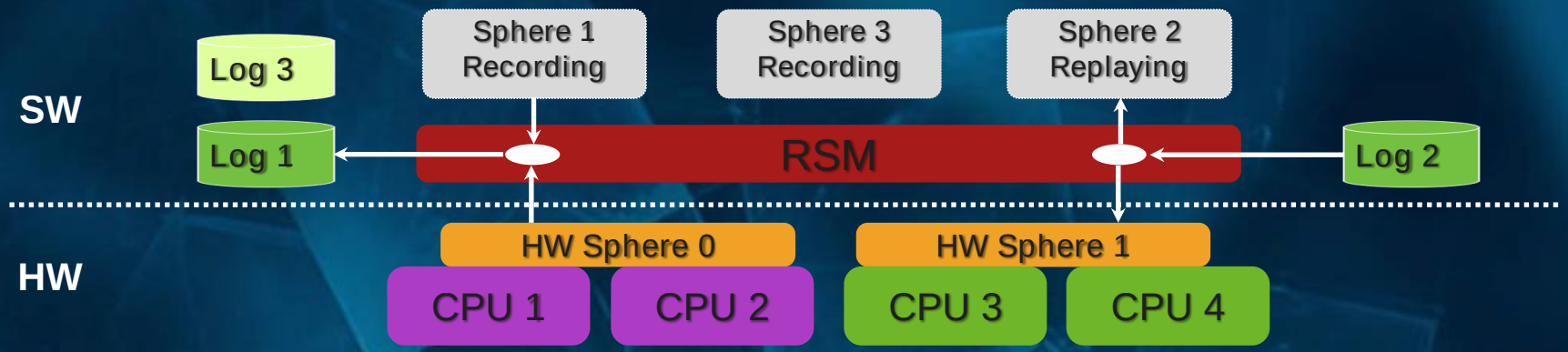
The Bulk Multicore

[CACM 2009]

Result: HW+SW for Deterministic Replay (Sam King)

Microsoft Research

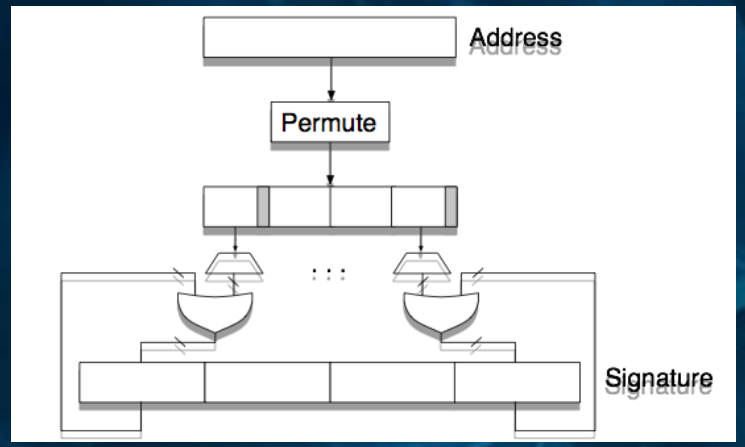
- Goal: Support deterministic replay of parallel programs with minimal recording overhead and tiny logging requirements
- Results:
  - By using the Bulk hardware, only need to record the interleaving of the chunks. Reduced the log size requirements by over 2 orders of magnitude [DeLorean in ISCA 2008]
  - Extended Linux to have multiple Replay Spheres, enabling virtualization of the recording and replay hardware [Capo in ASPLOS 2009]

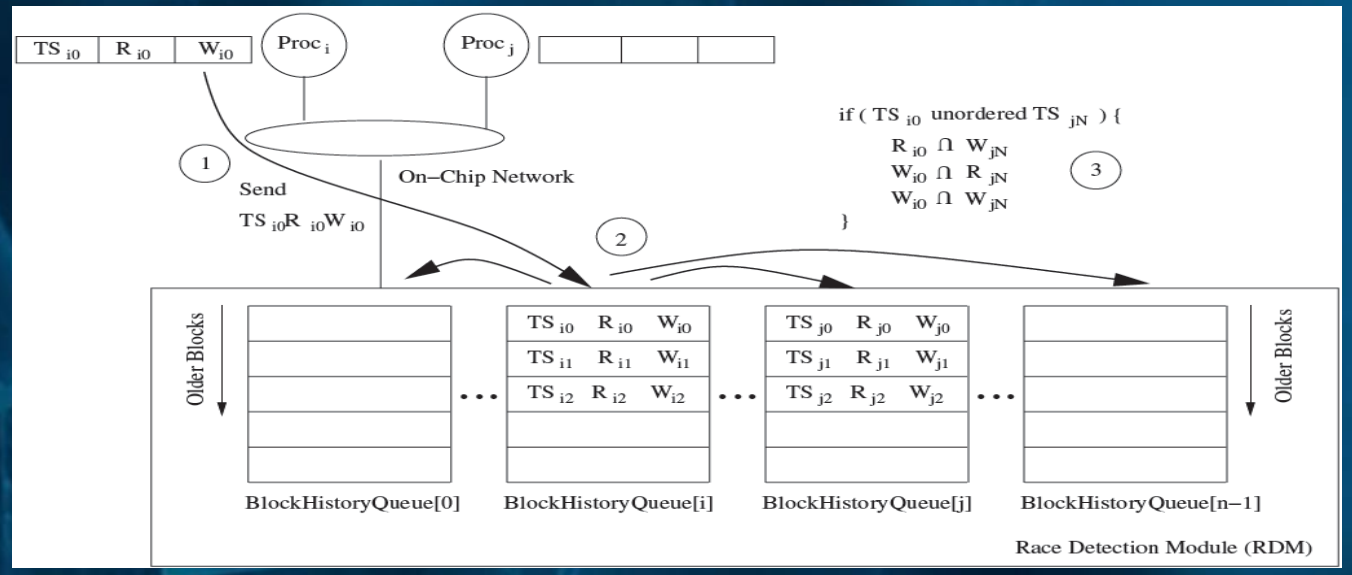


Result: HW Support for Data Race Detection

Microsoft Research

- Goal: Use hardware to detect data races dynamically in production run codes with very low overhead
- Results:
  - Processors automatically collect the addresses accessed in hardware signatures. An on-chip hardware module intersects the signatures in the background and identifies races
  - Effective race detection with only 20% execution overhead [SigRace in ISCA 2009]





DeNovo Architecture (Sarita Adve)

Microsoft Research

Rethinking hardware with disciplined parallelism

- Hypothesis 1: Future hardware will require disciplined parallel models for
  - Scalability
  - Energy efficiency
  - Correctness
    - Verifiability, testability, …
- Hypothesis 2: Hardware/runtime support can make disciplined models more viable
  - How do disciplined models affect hardware (and runtime)?
  - Rethink hardware from the ground up
    - Concurrency model, coherence, tasks, …
  - Co-design hardware and language concurrency models

Goal: Hardware that is

Scalable
Performance
Energy-efficient
Easy to design

Opportunities for Hardware

Microsoft Research

- Disciplined software allows optimizing
  - Communication fabric
  - Memory model and semantics
  - Task scheduling and resource management
- Goal
  - Unprecedented scalability and energy efficiency

Some Key Ideas

Microsoft Research

- Exploit from software
  - Structured control; region/effects; non-interference
- Communicate only the right data to the right core at the right time
  - Eliminates unscalable directory sharing lists, complex protocol races, performance thwarting indirections
  - Enables latency-, bandwidth-, and energy-efficient data transfers
  - No false sharing, efficient prefetching and producer-initiated communication

Ongoing and Future Work

Microsoft Research

- Simulation prototype of DeNovo architecture
- Broadening supported software
  - Unstructured synchronization and speculation
  - Legacy codes
- Runtime support for disciplined languages
  - Speculation, sandboxing, contract verification, …
- Virtual typed hardware/software interface
  - Language-, platform-independent virtual ISA

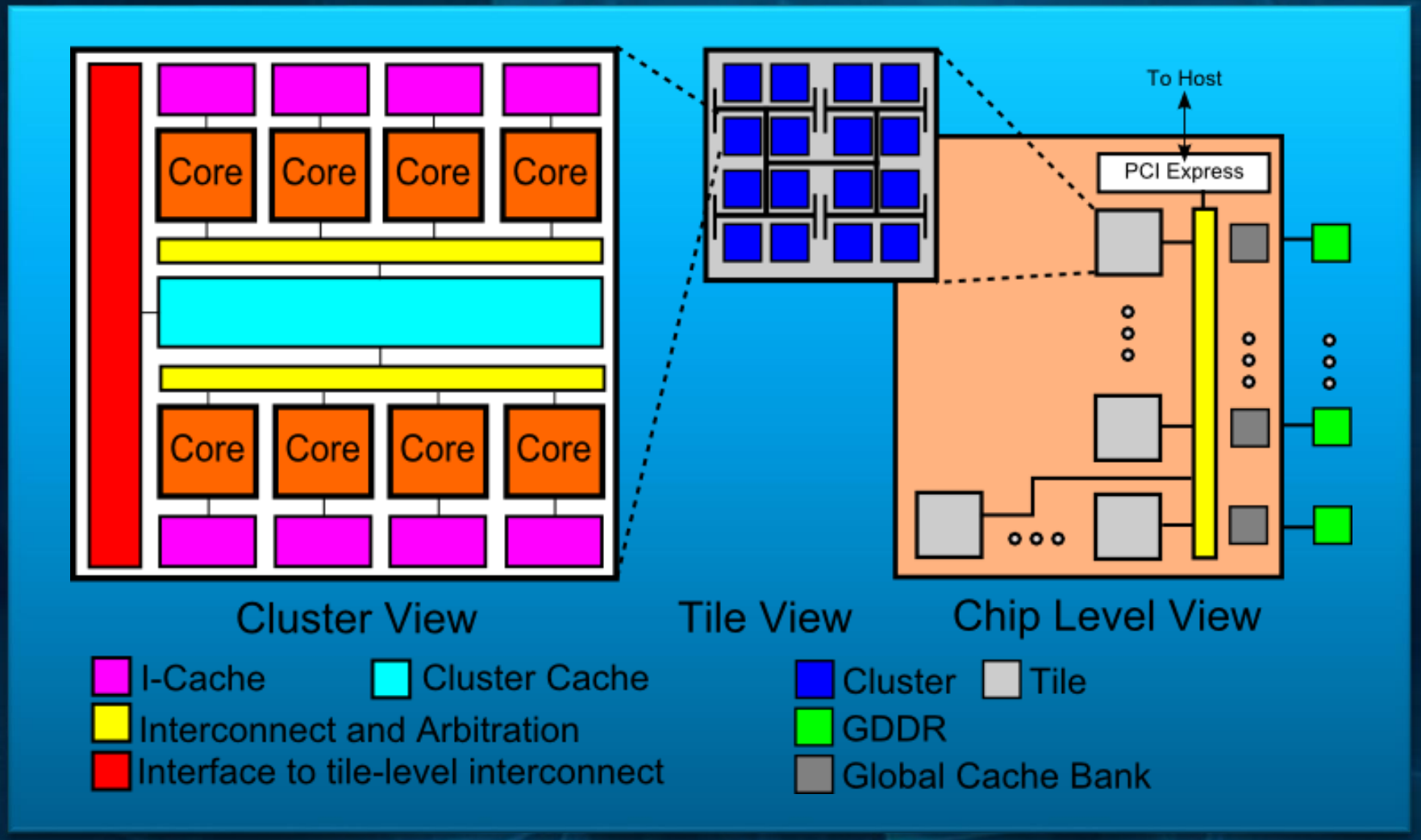
Microsoft Research

UPCRC Illinois

1000+ Core Architectures for Throughput-Oriented Computing

Architectural Framework: The Rigel Architecture (Sanjay Patel)

Microsoft Research



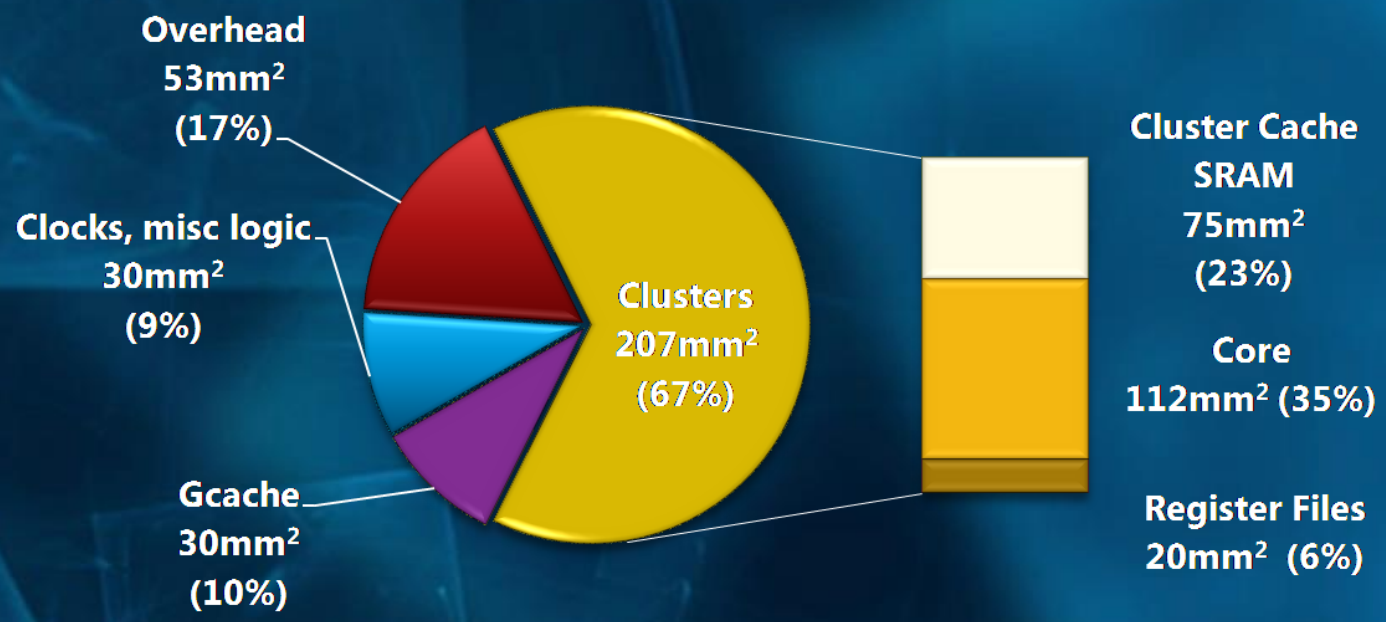
- Non-HW coherent caches
- Area-efficient core design
- Primitive support for scalable synchronization and reductions
- Cache management support for locality enhancement
- Compiler, simulator, RTL all available now

Architectural Framework: The Rigel Architecture

Microsoft Research

- 1024 cores, 8MB Cluster Cache, 4MB Global Cache (~3 TOps/sec)
- Synthesized Verilog @45nm for cores, cluster cache logic
- SRAM Compiler for SRAM banks
- Other Logic: interconnect, mem controllers, global cache
- Typical power ~70-99W

Rigel Area Breakdown



Mapping 1000 cores to 45nm

Microsoft Research

- 1024 cores, 8MB Cluster Cache, 4MB Global Cache (~3 TOps/sec)
- Synthesized Verilog @45nm for cores, cluster cache logic
- SRAM Compiler for SRAM banks
- Other Logic: interconnect, mem controllers, global cache
- Typical power ~70-99W

Questions to Be Addressed

Microsoft Research

- Programming models that scale from 1000 chips in a cluster to 1000 cores in a chip
- Run-time systems for scalable work distribution
- Locality management, architectural optimizations for memory bandwidth
- SIMD efficiency versus MIMD flexibility
- Power/energy optimizations for throughput oriented architectures

Summer School

Microsoft Research

- One intensive week of parallel programming
  - Principles, OpenMP, TBB, Java &C# classes, CUDA, tools
  - Lectures and Lab
- Intel and Microsoft participation in teaching
- 56 on site, 109 online participants
  - Graduate students, faculty, professionals
  - From U.S. to Korea
- High satisfaction

Microsoft Research

Microsoft®

Your potential. Our passion.™