The role of programming models in the multicore menace

Christoph von Praun

http://www.in.ohm-hochschule.de/professors/praun/

JUG, October 21, 2010

Many talks on that topic go like this:

- We are in the multicore era.
- Exploiting multicore technology requires parallel programming.
- Developing parallel programs is hard.
- Not much progress in parallel programming technology during past 20 years.
- Looming software crisis.

Many talks on that topic go like this:

- We are in the multicore era
- Exploiting multicore tec parallel programming.
- Developing paral ard.
- Not much pr programming technolo years
- Looming sc sis.

Too negative and pessimistic!

A more optimistic view

- Multicore can serve ever growing demand for computational resources
  - many app domains did parallel computing before multicore era; multicore makes their business more economic
  - few people will need 256-core notebook

A more optimistic view

- Multicores are energy efficient
  - for many task-parallel workloads: throughput, not speed matters
  - Case-study [5]:
    big-iron 2xIBM Power6 vs.
    highly-multithreaded 1xSUN Niagra-2
    (1.44x power efficiency on webserver workload)

A more optimistic view

- Most developers of highly-concurrent software don't need to do parallel programming
  - SQL
  - Task-parallel frameworks:
    - Web-application servers (WebSphere, …)
    - MapReduce (Lucene, Hadoop, …)
  - Data-parallel frameworks:
    - Parallel MATLAB
    - BLAS, HTAs

A more optimistic view

- If parallel programming is required: Most common pitfalls can be avoided with more discipline and planning:
  - pitfalls: data races, deadlock, violation of atomicity
  - help: tools, annotations, type-systems, new programming models, education!
  - sequential programming languages have overcome "null-pointer" challenge
- Writing correct parallel code not much more difficult than writing correct sequential code.

A more optimistic view

- Only few people need to handle the hard parts of parallel programming
  - OS code
  - non-blocking data structures
  - implementation of synchronization
  - library functionality (e.g. MapReduce shuffle)
  - can be fun

A more optimistic view

- Programming technology is important but not key
  - not more so than for seq. programming
  - small factor in multicore software engineering productivity (= time and resources to achieve a high-performance solution)
  - most scalable codes to date use Fortran/MPI (IBM BlueGene/L,P)

Challenges remain

- Performance engineering remains difficult
  - hardware (computer architects?) is to blame
  - locality and scalability are permanent problems
  - requires very broad skill set

Challenges remain

- Education and practice matter
  - Technology races ahead like achilles … CS curricula evolve like turtles
  - teach concepts, not tools and technologies
  - use patterns to encode and teach expert advice

Summary

- **Grand challenges**: need human intelligence to be solved (art)
  - Scalability
  - Access locality
- **Engineering problems**: threat does not go away, but can be mitigated with programming technology, discipline, and education:
  - Choice of algorithm and architecture
  - Data races, deadlock, ...
- **Non-issues**: can be automated with (novel) technology and tools:
  - Register allocation, ILP, ...

Recent programming technology

Libraries and
OO frameworks

TBB

Language extensions

New programming
languages

Cilk++ Chapel

RapidMind

Fortress

Domain-specific libraries
and generators

FFTW
ATLAS

New languages vs. libraries

- Libraries are adopted more quickly than new programming languages.
- New languages offer little advantage in practice
  - benefit mostly "academic"
  - programming language research

Cilk++

Chapel

RapidMind

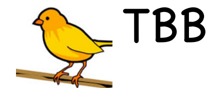
Fortress

FFTW

ATLAS

General purpose vs. domain specific

- Many technologies try to be general purpose
- High-/peak-performance is achieved more easily with technologies targeted to a specific application domain



TBB

OpenCL

Cilk++

Chapel

RapidMind

Fortress

FFTW

ATLAS

“Academic” Benefit of new Languages

specifically:

- For Teaching:
  - combines elements of functional and imperative programming
  - parallel programming concepts can be expressed directly in the language
  - succinct programs
  - explore issues at different “tiers of parallelism”
  - managed runtime system

“Academic” Benefit of new Languages

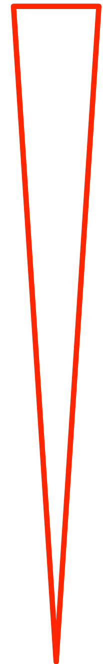
specifically:

- For Research:
  - seamless combination of shared memory and distributed computation
  - advanced type system
  - semantics of parallel programs
    - shared memory model
    - transactional synchronization
    - deadlock freedom
    - determinacy

Tiers of Parallelism

parallelization	techniques
(1) automatic	parallelizing compiler
(2) deterministic	fully independent computations or serialization
(3) explicitly synchronized (data race free)	critical sections, transactions, threads, actors
(4) low-level (with race conditions)	implementation of threads, synchronization mechanisms, non-blocking data structures

high-level



low-level

Source: [1]

- Tier is an abstraction layer
- Every layer has its abstractions and vocabulary
- Upper tiers build on lower-level tiers (not the other way round).

Tiers of Parallelism

parallelization	techniques
(1) automatic	parallelizing compiler
(2) deterministic	fully independent computations or serialization
(3) explicitly synchronized (data race free)	critical sections, transactions, threads, actors
(4) low-level (with race conditions)	implementation of threads, synchronization mechanisms, non-blocking data structures

Focus of X10

Source: [1]

Sequential X10 is like Java plus ...

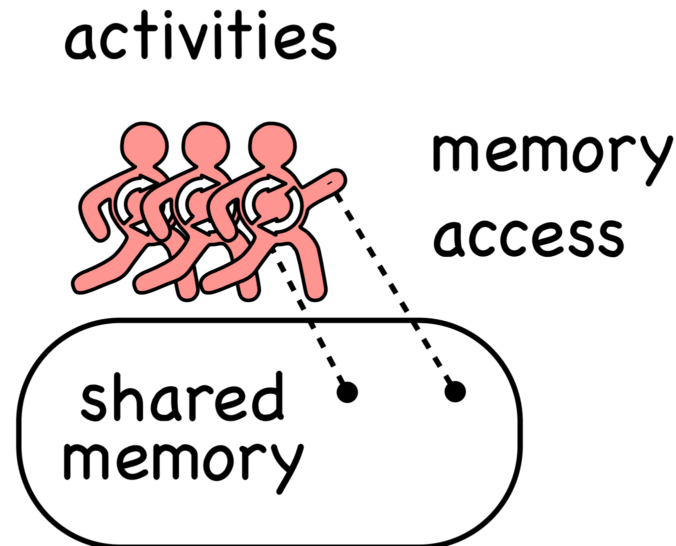
+ constraints
+ anonymous functions / closures
+ structs (constants objects w/o header)
+ minor improvements
    - slicker syntax
    - var/val = Java final / non-final (val is default)
    - type inference
    - better constructors

Source: [6]

Research Aspects in X10

- Seamless combination of shared memory and distributed computation
- Advanced type system
- Semantics of parallel programs
  - shared memory model
  - transactional synchronization
  - deadlock freedom
  - determinacy

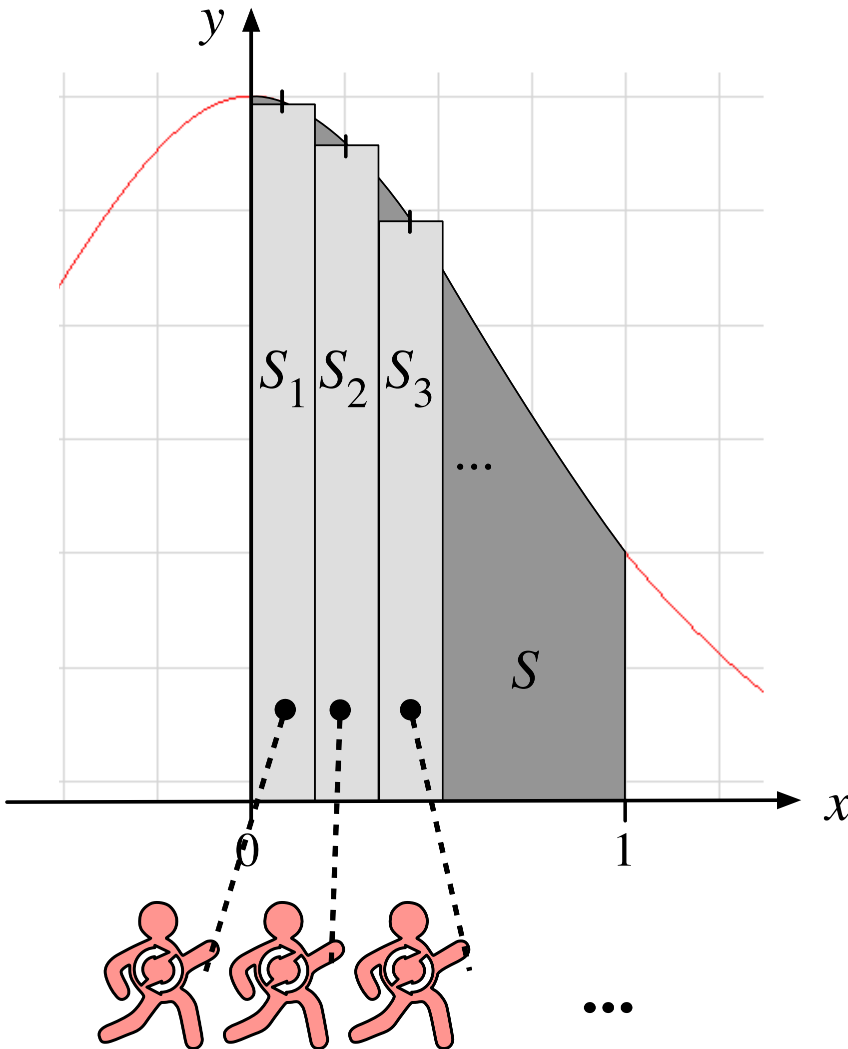
Parallel X10



Activities:

- Lightweight tasks
- Scheduled through thread queue (work stealing being implemented)

Parallel Numeric Integration

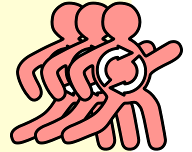


Computations of the S_i are independent.

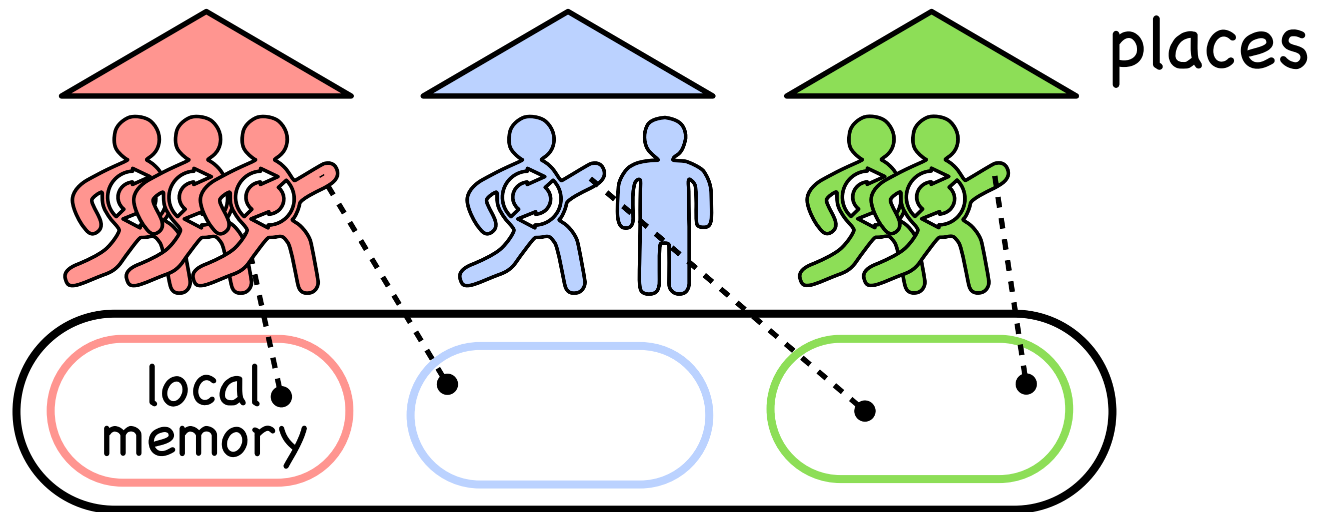
Parallel X10

```
val NSTEPS: Long = 1000000l;
var sum: Double = 0.0;

def integrate() {
    val step: Double = 1.0 / NSTEPS;
    finish foreach ((i) in 0..NSTEPS) {
        var x: Double  = (i + 0.5) * step;
        atomic sum += 4.0 / (1.0 + x * x);
    }
    Console.OUT.println("S = " + (sum * step));
}
```



Distributed X10



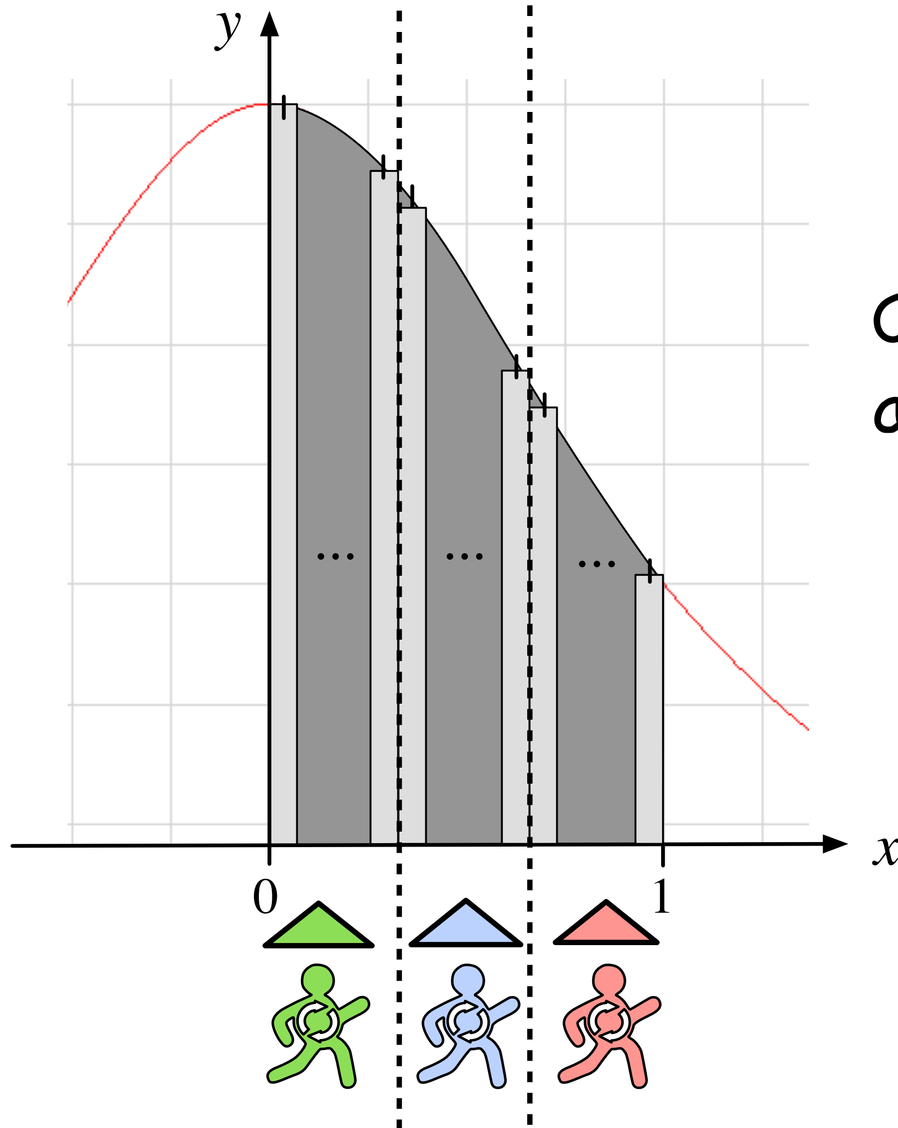
partitioned global address space (PGAS)

Place (≈ Process)

- `here` refers to current place
- Objects are localized
  - `o.home` is place where object is allocated
  - read/write access only from home place
- Local and remote references
  - can refer to any data from any place
  - can pass references freely
- Place shifting: `at(p) S` - execute `S` on place `p`
- Arrays can be distributed (partitions in multiple places)

Source: [6]

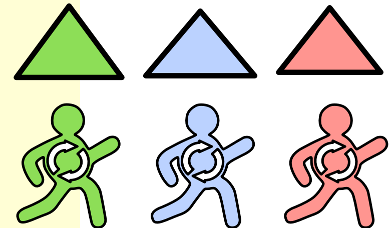
Distributed Numeric Integration



Computation blocked across three places.

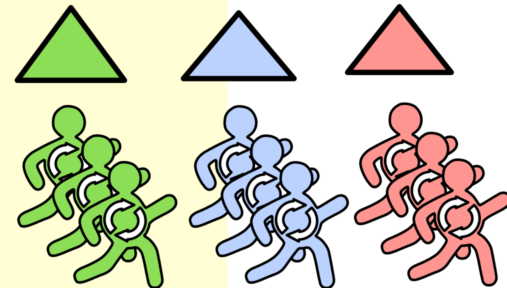
Distributed X10

```
val NSTEPS: Long = 1000000l;
def integrate_dist() {
    val unique = Dist.makeUnique();
    val arr =  DistArray.make[Double](unique);
    val step: Double = 1.0 / NSTEPS;
    val dist = Dist.makeBlock(0..NSTEPS);
    finish foreach (p in unique.places()) {
       at (p) {
          for ((i) in dist|here) {
             val x  = (i + 0.5) * step;
             arr(p.id) += 4.0 / (1.0 + x * x);
          }
       }
    }
    val sum = arr.reduce(Double.+, 0) * step;
   Console.OUT.println("S = " + sum);
}
```



Parallel & Distributed X10

```
val NSTEPS: Long = 1000000l;
def integrate_dist() {
    val unique = Dist.makeUnique();
    val arr =  DistArray.make[Double](unique);
    val step: Double = 1.0 / NSTEPS;
    val dist = Dist.makeBlock(0..NSTEPS);
    finish foreach (p in unique.places()) {
       at (p) {
          foreach ((i) in dist|here) {
             val x  = (i + 0.5) * step;
             atomic arr(p.id) += 4.0 / (1.0 + x * x);
          }
       }
    }
    val sum = arr.reduce(Double.+, 0) * step;
   Console.OUT.println("S = " + sum);
}
```



Research Aspects in X10

- Seamless combination of shared memory and distributed computation
- Advanced type system
- Semantics of parallel programs
  - shared memory model
  - transactional synchronization
  - deadlock freedom
  - determinacy

Constraints

- Constraints are boolean expressions
  - can refer to self, immutable fields of the objects (properties) and compile-time constants
- Constraints are checked at compile-time

Type Constraints

- Types can be annotated with constraints that restrict the domain of the type
  - Unconstrained: `Int`
  - Constrained: `Int{self > 0}`

argument must be a 1-dim array.

```
def sum(arr: Array[Int]{rank==1}): Int {
   var sum: Int = 0;
   for (i in arr)
      sum += arr(i);
   return sum;
}
```

Method Constraints (Guards)

- States precondition for which a method is available.

x, y must have same length property

```
def dotProduct(x: Vec, y: Vec) {x.len == y.len}
{...}
```

method only available if rank of array is 1

```
public final class Array[T] (region: Region) {
    public property rank: Int = region.rank;

    public safe def apply(i0:Int){rank==1}:T
    {...}
    public safe def apply(i0:Int, i1:Int){rank==2}:T
    {...}

    public safe def apply(pt:Point
                {self.rank==this.rank}):T
    {...}
    ...
}
```

rank of point must be rank of array

```
public abstract class Matrix (m:Int, n:Int) {

    static type Matrix (m: Int, n: Int) =
        Matrix {self.m == m && self.n == n};

    abstract operator this *
        (that: Matrix{self.m == this.n}) :
        Matrix (this.m, that.n);

    public static def main(Rail[String]) {
        val a: Matrix(3,5) = ...;
       val b: Matrix(5,7) = ...;
       use (A*B); // OK
       use (B*A); // does not work
    }
}
```

Source: [6]

Place Constraints

- Constraints often refer to place expressions
  - "place type"
  - compiler knows whether reference refers to local or remote object
- `here` refers to current place
- `SomeType!` means `SomeType{self.home==here}`

```
def fun(x: SomeType!) {
    x.access();
}
```

guaranteed
place local access

```
def fun(x: SomeType) {
    x.access();
}
```

maybe local /
maybe remote

- In strict compilation mode: ERROR
- Normal compilation mode: runtime check

```
def fun(x: SomeType) {
    at (x) {
      x.access();
    }
}
```

Place shifting

Access is always local, no runtime check.

Research Aspects in X10

- Seamless combination of shared memory and distributed computation
- Advanced type system
- Semantics of parallel programs
  - shared memory model
  - transactional synchronization
  - deadlock freedom
  - determinacy

Deadlock Freedom

- X10 programs that use `async/foreach/ateach` and `finish` cannot deadlock
  - task graph forms a tree
  - parents can wait on children (`finish`)
  - children cannot wait on parents
  - wait-relation is acyclic

Deadlock Freedom

- Unconditional `atomic` blocks cannot introduce deadlock.
- X10 programs that use `clocks` (generalized form of barrier for "phased" computations) cannot deadlock.
  - proof in [2]

Determinacy

- Many parallel programs are determinate
  - Order of execution does not change result
- X10 does not guarantee determinacy
  - Research on a determinacy checker / determinate subset of X10 is underway

Data Race Control

- X10 does not guarantee data race freedom
- X10 makes it difficult to introduce data races
  - functional programming style:
    - value types are default
    - structs are immutable
  - mutable shared state:
    - shared variable (not yet implemented)
    - fields of heap objects
    - mutable array variables
  - no global mutable state

Teaching Aspects in X10

- Combines elements of functional and imperative programming
- Parallel programming concepts can be expressed directly in the language
- Succinct programs
- Explore different "tiers of parallelism"

Functions

- Function literals
  - `(x: Int) => x`
  - `(x: Int) => 4 / (1 + x*x)`

- Function types
  - `(Int) => Int`
  - Contravariant in the argument type
  - Covariant in the result type

- Function application
  - `f(i)`

```
static def runTimed( fn: () => Void, n:String) {
    var start: long = -System.nanoTime();
    fn();
    start += System.nanoTime();
    start /= 1000000;
    Console.OUT.println(n + " duration=" + start + "ms");
}

static def primePrint() {
    var j: long = 0;
    while (j < MAX) {
        j++;
        if (isPrime(j))
            Console.OUT.println(j + ", ");
    }
}

public static def main(Rail[String]){
    runTimed(()=>{
        primePrint();
    }, "sequential");
}
```

Teaching Aspects in X10

- Combines elements of functional and imperative programming
- Parallel programming concepts can be expressed directly in the language
- Succinct programs
- Explore different "tiers of parallelism"

X10 in my Parallel Programming class

15 week undergraduate course (5 ECTS credits)

25 students

- syllabus structured according to parallel programming patterns
- 6 major programming assignments to exercise different structural patterns:
  - NumericalIntegration, HeatTransfer, PrefixSum, MapReduce, MergeSort, TSP
  - very simple, succinct solutions
- about 1/3 of students used X10

MapReduce

Input & Output: each a set of key/value pairs
Programmer specifies two functions:

```
map (InKey, InValue): List[Pair[OutKey, IntermediateValue]]
```

- Processes input key/value pair
- Produces set of intermediate pairs

```
reduce (OutKey, List[IntermediateValue]): List[OutValue]
```

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usually just one)

Word Counting: map function

map (InKey, InValue): List[Pair[OutKey, IntermediateValue]]

```
/* @param  inkey    document name
 * @param  invalue  document contents
 * @param  result   collect intermediate result
 */
def map (inkey: String, invalue: String,
         result: List[Pair[String, Int]]) {
    val chars = iv.chars();
    var sb: StringBuilder = new StringBuilder();
    for (c in chars) {
        if (c.isLetterOrDigit())
           sb.add(c.toLowerCase());
        else {
           // emit result
           result.add(Pair[String, Int](sb.result(), 1));
           sb = new StringBuilder();
        }
    }
}
```

Word Counting: reduce function

reduce (OutKey, List[IntermediateValue]): List[OutValue]

```
/* @param  outkey                word
 * @param  intermediate_values   occurrence of word in all texts
 * @return                       sum of occurrences
 */
def reduce (outkey: String,
           intermediate_values: List[Int]): List[Int] {
   var acc: Int = 0;
   val ret = new List[Int]();
   for (i in intermediate_values)
       acc += i;
   ret.add(acc);
   return ret;
}
```

Word Counting: main function

```
public static def main(Rail[String]!) {
   // (1) initialize
   val mr = new MapReduceImpl[String, String, Int, String, Int](3,3);
   val m = (ik:String, iv:String, result:List[Pair[String, Int]]) =>
          { mr.map(ik, iv, result); };
   val r = (ok: String, iv: List[Int]): List[Int] =>
          { return mr.reduce(ok, iv); };
   mr.registerReduceFun(r);
   mr.registerMapFun(m);
   mr.setInput(input);

   // (2) run it
   mr.run();

   // (3) print result
   val result <: List[Pair[String,Int]] = mr.getResult();
   for (p in result)
      Console.OUT.println(p.first + ", " + p.second);
}
```

Generic MapReduce

```
public interface MapReduce
    [InKey, InVal, IntermediateVal, OutKey, OutVal] {

    property def num_map_tasks(): Int;
    property def num_reduce_tasks(): Int;

    def setInput(i: ValRail[Pair[InKey, InVal]]);
    def registerMapFun(m: (InKey, InVal,
                List[Pair[OutKey, IntermediateVal]]) => Void);
    def registerReduceFun(r: (OutKey,
                List[IntermediateVal]) => List[OutVal]);
    def run();
    def getResult(): List[Pair[OutKey, List[OutVal]]];
}
```

Take away

- Situation is not as bleak as many other talks say
  - Opportunities not menace!
  - "~~embarrassingly~~ parallel" => "fortunately parallel"
  - Interesting time to do language and programming technology research

Take away

parallelization	techniques
(1) automatic	parallelizing compiler
(2) deterministic	fully independent computations or serialization
(3) explicitly synchronized (data race free)	critical sections, transactions
(4) low-level (with race conditions)	implementation of threads, synchronization mechanisms, non-blocking data structures

- Goal: Most programmers should operate at tier-1 or tier-2
  - high-performance with tier-1 and tier-2 solutions
  - problem-driven research: Berkeley Dwarfs [4]
  - focussed, domain-specific solutions as starting point (scientific apps, games, webservices...)

Take away

- Different domains have different challenges:
  - middleware for web-services: market timing
  - scientific applications: performance
  - games: simplicity of the programming interface, programmability
- Teaching/eduction is an important success factor

Sources

[1] Michael Scott: “Making the simple case simple”, Position paper, Workshop on Curricula for Concurrency, in conjunction with OOPSLA, 2009.

[2] Vijay Saraswat, Radha Jagadeesan: “Concurrent clustered programming”. In CONCUR - Concurrency Theory, 2005.

[3] Philippe Charles, Christopher Donawa, Kemal Ebcioglu, Christian Grothoff, Allan Kielstra, Christoph von Praun Vijay Sarsawat, Vivek Sarkar. X10: “An Object-Oriented Approach to Non-Uniform Cluster Computing”, Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), October 2005.

[4] http://view.eecs.berkeley.edu/wiki/Dwarf_Mine

[5] Rolf Kersten, Sun Microsystems: “Effizienz im Rechenzentrum”, September 2007.

[6] Bard Bloom, Vijay Saraswat: “Sketch of X10”, X10 Workshop at TJ Watson, April 16, 2010.

Thank you!

praun@acm.org

SOME RIGHTS RESERVED

This work is licensed under a Attribution-Noncommercial-Share Alike 3.0 Unported License.

You are free:

- **to Share** — to copy, distribute and transmit the work
- **to Remix** — to adapt the work

Under the following conditions:

- **Attribution**. You must attribute the work to "Christoph von Praun" (but not in any way that suggests that the author endorses you or your use of the work).
- **Noncommercial.** You may not use this work for commercial purposes.
- **Share Alike**. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to http://creativecommons.org/licenses/by-nc-sa/3.0/