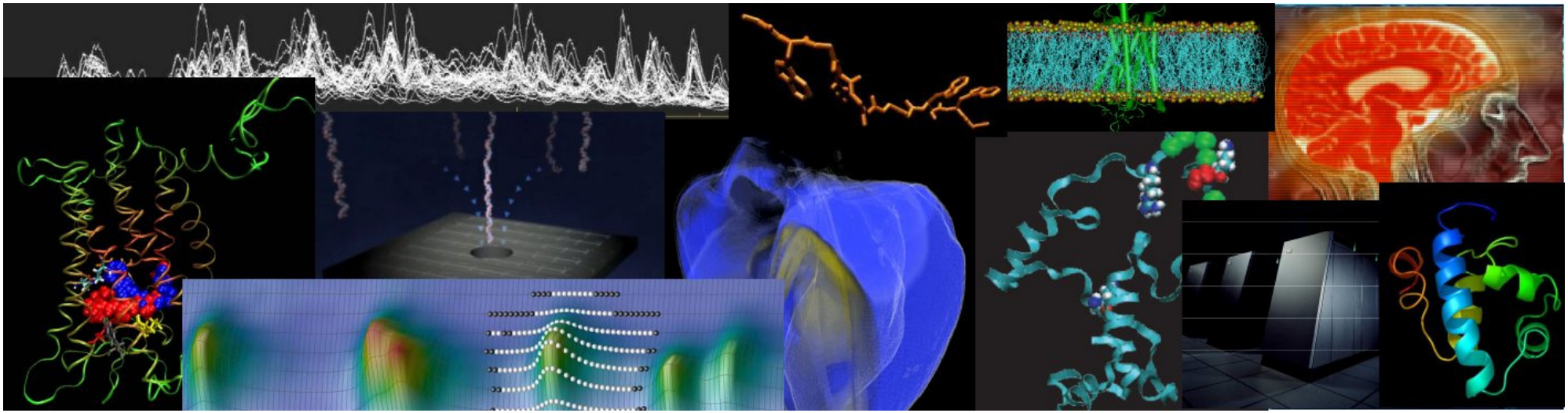
Irina Fedulova – HPC Software Engineer, ESSL & Simics Manager, RSTL
June 28, 2011

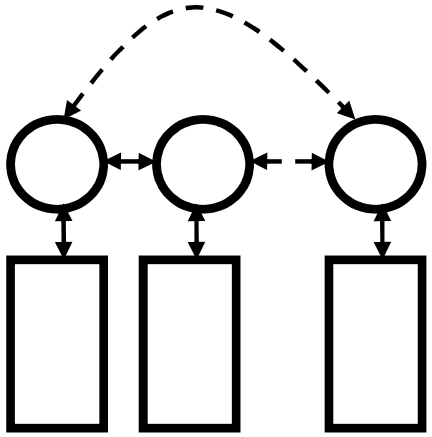
Язык параллельного программирования X10



План лекции

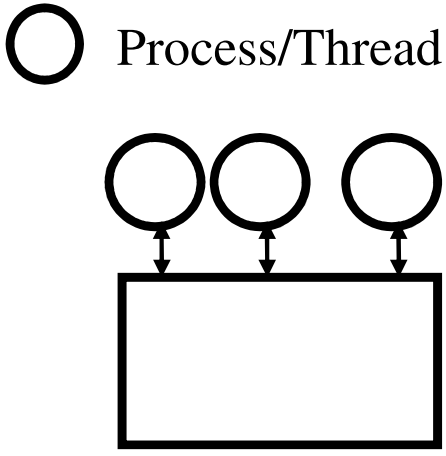
- Концепция PGAS, APGAS
- Модель выполнения X10
- Основные понятия X10
  - Activity
  - Place
  - At
  - Распределенные объекты
  - Атомарные операции
- Аналогия между X10 и MPI
- Пример
- Текущий статус и основные направления развития
- Заключение

Модели параллельных вычислений



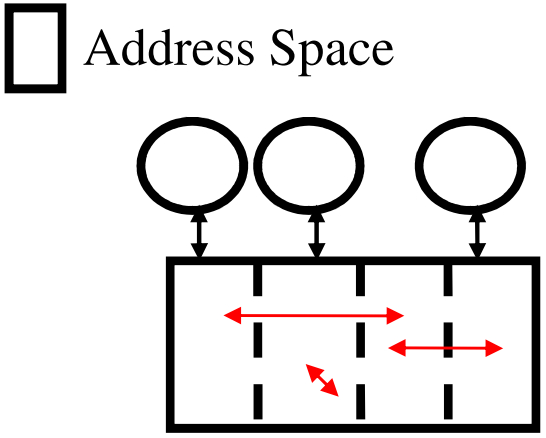
Message passing

MPI



Shared Memory

OpenMP



PGAS

UPC, CAF, Chapel, X10

Partitioned Global Address Space (PGAS)

- Традиционная схема
  - Распределенная память => message passing модели (MPI, Charm++, …)
    - Хорошо масштабируется
    - Сложно программировать
  - Общая память => OpenMP, POSIX threads, …
    - Не очень хорошо масштабируется
    - Не так сложно программировать
  - Гибридное MPI + OpenMP программирование
    - Максимальная производительность
    - Максимальные усилия

«Ассемблер»

- Global Address Space
  - С точки зрения программиста память общая => сравнительная легкость программирования
  - Параллелизм по типу SPMD, как в MPI
  - Языки, поддерживающие концепцию PGAS
    - C => UPC
    - Fortran => Co-Array Fortran
    - Java => Titanium
    - Chapel [Cray]
    - X10 [IBM]

«Языки высокого уровня»

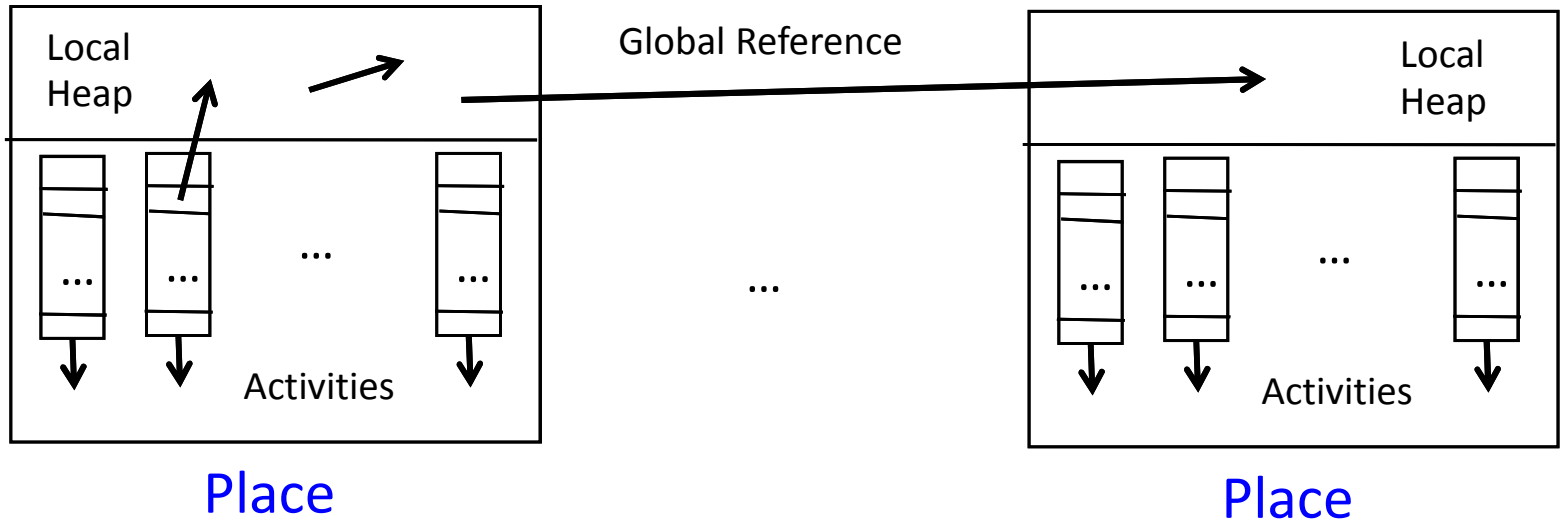
Partitioned Global Address Space (PGAS)

- Вычисления производятся на нескольких «процессах» (threads, MPI processes, …)
- SPMD
- Процессы могут получить доступ к данным других процессов (глобальное адресное пространство)
- Данные физически не перемещаются между процессами
  - Компилятор обрабатывает запросы к чужой памяти автоматически
  - Односторонние коммуникации
- Структуры данных (массивы и т.п.) могут быть распределены между процессами

X10 = Asynchronous PGAS

- APGAS = Asynchronous PGAS
  - X10 [IBM]
  - Chapel [Cray]
- Расширение модели выполнения PGAS
  - Задачи распределяются между процессами динамически
    - Work-stealing scheduler
  - Процессы могут порождать другие процессы динамически
- Разрабатывается в рамках проекта HPCS (High Performance Computing System), финансируется DARPA
- Позволяет использовать native код на C/C++, Java, Scala, …
  - Директива @Native
  - Например, для вызова библиотек (FFTW, BLAS и т.п.)
- [На стадии разработки] Процессы смогут работать на процессорах с различными архитектурами (x86, Power, GPU, …)

Модель выполнения X10



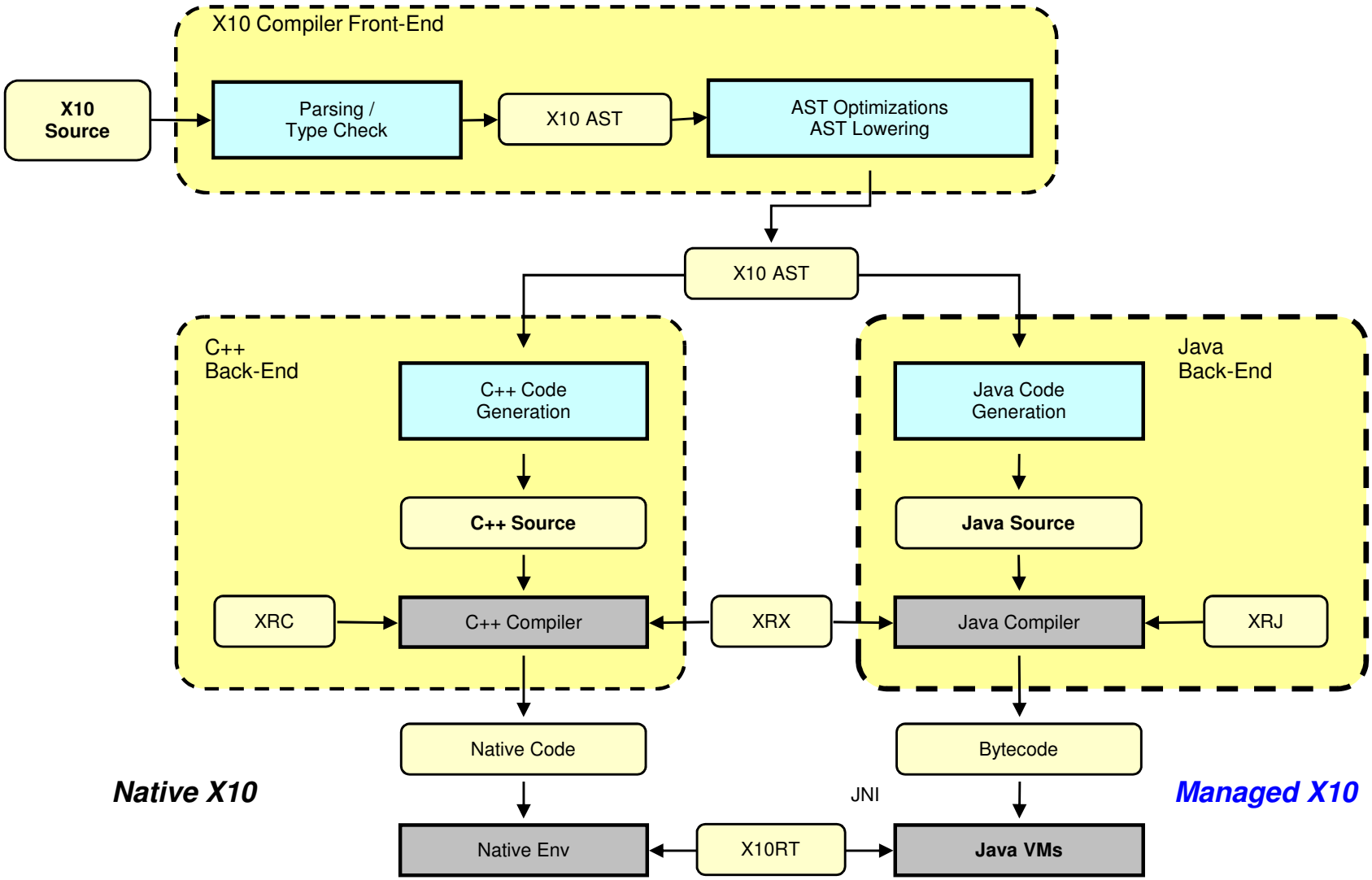
- 2 основных понятия: place и activity
- Мелкозернистый параллелизм: async S
- Удаленные вызовы: at (p) S
- Синхронизация: finish S

Модель выполнения X10 (продолжение)

- activity
  - «Активность», «поток» (последовательная функция, процедура, …), выполняющаяся в рамках одного «процесса» (place)
- place
  - «Процесс», «вычислительный узел», «адресное пространство»
  - Коллекция локальных активностей (activity) и объектов
- Локальная/глобальная синхронизация
  - Локальная синхронность
    - Гарантирована локальная когерентность
  - Глобальная асинхронность
    - Не гарантирован порядок выполнения активностей
    - Явные инструкции для синхронизации
- Правило локальности
  - Доступ к локальным объектам процесса разрешен только для локальных активностей

  => для доступа к удаленным объектам необходимо создавать удаленные активности

Процесс компиляции



Среда выполнения (runtime)

- Standalone
  - Multiple places on a single host, shared memory
- Sockets
  - Использует сокеты и ssh для поддержки multiple places на одном или нескольких хостах
- MPI
  - На основе MPI2, подходит для distributed memory систем
- PGAS
  - Использует Common PGAS runtime
  - Не является open source, распространяется в бинарном виде
  - pgas_sockets, pgas_lapi, pgas_bgp

Hello world

Hello.x10

```
/**
 * The canonical "Hello, World" demo class expressed in X10
 */
public class Hello {

    /**
     * The main method for the Hello class
     */
    public static def main(Array[String]) {

        finish for (p in Place.places()) {
            async at (p) {
                x10.io.Console.OUT.println(
                        "Hello from place " + p.id + " of " + p.MAX_PLACES);
            }
        }
    }

}
```

```
irina@carbon:~/research/x10/workspace/HelloWorld/src
irina@carbon:~/research/x10/workspace/HelloWorld/src$ x10c++ -x10rt mpi Hello.x10 -o Hello
irina@carbon:~/research/x10/workspace/HelloWorld/src$ ls
Hello  Hello.cc  Hello.h  Hello.x10  xxx_main_xxx.cc
irina@carbon:~/research/x10/workspace/HelloWorld/src$ mpirun -np 4 Hello
Hello from place 0 of 4
Hello from place 1 of 4
Hello from place 2 of 4
Hello from place 3 of 4
```

Возможности языка X10

- Многие возможности Java
  - Classes (w/ single inheritance)
  - Interfaces, (w/ multiple inheritance)
  - Instance and static fields
  - Constructors, (static) initializers
  - Overloaded, overrideable methods
  - Garbage collection
  - Familiar control structures, etc.
- Structs
- Closures
- Substantial extensions to the type system
  - Dependent types
  - Generic types
  - Function types
  - Type definitions, inference
- Multi-dimensional & distributed arrays
- Concurrency
- Fine-grained concurrency:
  - async S;
- Atomicity and synchronization
  - atomic S;
  - when (c) S;
- Ordering
  - finish S;
- Distribution
  - GlobalRef[T]
  - at (p) S;

Place

- Place = «вычислительный узел», «процесс»
  - ~= MPI процесс
- Число процессов Place.MAX_PLACES задается при запуске программы и не может быть динамически изменено
- Число процессов, как правило, соответствует числу физических процессоров, но это не обязательное условие
- Адресные пространства процессов не имеют общих областей
- Процесс может ссылаться на объекты в адресном пространстве другого процесса

```
public static def main(Array[String]) {

    finish for (p in Place.places()) {
        async at (p) {
            x10.io.Console.OUT.println(
                    "Hello from place " + p.id + " of " + p.MAX_PLACES);
        }
    }
}
```

activity

- Activity = «активность», «поток»
  - ~= OpenMP thread
- Порождаются динамически
- Каждая активность является последовательной
  - …но может порождать другие активности
- Внутри одного процесса (place) могут выполняться одновременно несколько активностей
- Активность может находиться в одном из следующих состояний
  - Running
  - blocked on some condition
  - terminated

```
public static def main(Array[String]) {

    finish for (p in Place.places()) {
        async at (p) {
            x10.io.Console.OUT.println(
                    "Hello from place " + p.id + " of " + p.MAX_PLACES);
        }
    }
}
```

finish

finish S

- Гарантирует завершение всех активностей, порожденных в S
- Перехватывает все исключения, сгенерированные активностями, порожденными в S
- Генерирует исключение, если хотя бы одна из порожденных активностей завершилась с ошибкой

```
public static def main(Array[String]) {

    finish for (p in Place.places()) {
        async at (p) {
            x10.io.Console.OUT.println(
                    "Hello from place " + p.id + " of " + p.MAX_PLACES);
        }
    }
}
```

async

async S

- Порождает активность, которая выполняет выражение S
- Управление немедленно возвращается
  - Родительский процесс не блокируется
  - Активность может быть еще даже не порождена, а только поставлена в очередь на создание
- S имеет доступ к объектам, видимым в данном месте текущего процесса
- Тело активности должно быть идентично void методу анонимного внутреннего класса, определенного в данной точке кода
  - Нельзя делать return, break, continue, …

```
public static def main(Array[String]) {

    finish for (p in Place.places()) {
        async at (p) {
            x10.io.Console.OUT.println(
                    "Hello from place " + p.id + " of " + p.MAX_PLACES);
        }
    }
}
```

var vs.val

- var
  - «переменная»
- val
  - «значение»
    - переменная, значение которой можно задать 1 раз
    - значение должно быть определено перед первым использованием
- Внутри блока async нельзя ссылаться на var переменные, определенные за пределами async
  - Либо вокруг async должен стоять finish на том же уровне видимости, что var переменная

```
public static def var_and_val() {
    val N = 1000;
    val x = new Array[Int](1..N);

    finish for (var k: Int = 1; k <= N; k++) {
        val kk = k;
        async x(kk) = calculate_x_kth(kk);
    }
}
```

```
public static def var_and_val2() {
    val N = 10;
    val x = new Array[Int](1..N);

    finish for (var k: Int = 1; k <= N; k++) {
        finish {
            async x(k) = calculate_x_kth(k);
        }
    }
}
```

at

at(p) S

- удаленный вызов S на процессе p
  - На процессе p вызывается S
  - Текущая активность блокируется, пока не завершится S
- Значения переменных, на которые ссылается S, копируются в p, и связываются с переменными с теми же именами, но в новом процессе
  - Необходимо следить за тем, что переносящиеся переменные
    - являются либо структурами, либо stateless objects (val)
    - не являются большими объектами (структурами), например перенос this приведет к копированию всего объекта

```
public static def main(Array[String]) {

    finish for (p in Place.places()) {
        async at (p) {
            x10.io.Console.OUT.println(
                    "Hello from place " + p.id + " of " + p.MAX_PLACES);
        }
    }
}
```

Многомерные массивы

- x10.array
  - Point
    - Точка в n-мерном пространстве
  - Region
    - Множество точек (set of Points) одной размерности
    - Может быть прямоугольным
    - …или произвольным
  - Array
    - Определяется над областью (region)

```
val MAX_HEIGHT=20;
val Null = Region.makeUnit(); //Empty 0-dimensional region
val R1 = 1..100; // IntRange
val R2 = R1 as Region(1);
val R3 = (0..99) * (-1..MAX_HEIGHT);
val R4 = Region.makeUpperTriangular(10);
val R5 = R4 && R3; // intersection of two regions

val N = 10;
val x = new Array[Int](1..N);
```

```
public static def sum(a: Array[Int]): Int {
    var s : Int = 0;
    for (p in a) s += a(p);
    return s;
}
```

```
static def addInto(src: Array[Int], dest:Array[Int]) {src.region == dest.region} = {
    for (p in src.region) dest(p) += src(p);
}
```

Распределенные массивы

- x10.array.Dist
  - Распределение области между процессами (mapping of Region over Places)
  - Определено несколько стандартных
    - block
    - constant
    - unique
  - Any user defined (подкласс x10.array.Dist)
- При попытке обратиться напрямую к элементу массива, который расположен на другом процессе генерируется ArrayIndexOutofBoundsException

```
def dist_arr() {
    val R : Region = 1..1000;
    val D : Dist = Dist.makeBlock(R);
    val da : DistArray[Float] = DistArray.make[Float](D, (Point(1))=>0.0f);
}
```

Распределенные массивы (продолжение)

- ateach
  - Параллельное выполнение активности на всех элементах распределенного массива
  - На процессах, поддерживающих SMP, ateach выполняется автоматически в многопоточном режиме
    - «бесплатное» гибридное программирование MPI+OpenMP

```
public class HeatTransfer_v2 {
    static val n = 3, epsilon = 1.0e-5;
    static val BigD = Dist.makeBlock((0..n+1)*(0..n+1) as Region);
    static val D = BigD | ((1..n)*(1..n) as Region);
    static val LastRow = (0..0)*(1..n) as Region;
    static val A = DistArray.make[Double](BigD,(p:Point)=> {
        LastRow.contains(p) ? 1.0 : 0.0
                });
    static val Temp = DistArray.make[Double](BigD);
    def run() {
        val D_Base = Dist.makeUnique(D.places());
        var delta:Double = 1.0;
        do {
            finish ateach (z in D_Base)
            for (p:Point(2) in D | here)
                Temp(p) = stencil_1(p);

            ....
```

GlobalRef

- Глобальная ссылка на объект, которая может быть передана в другие процессы
- Полезна при построении структур данных, которые не очень удобно реализовать с помощью распределенных массивов
  - Деревья, списки
- В MPI для аналогичных задач пришлось бы использовать MPI_Send, MPI_Recv и синхронизацию

atomic

atomic S

- Выполнить действие S атомарно (безусловная атомарность)
- Условия на S
  - Неблокирующее
  - Последовательное (не должно создавать новых активностей)
  - Локальное (не должно обращаться к удаленным данным)

```
var total:Int = 0;
finish for (var k: Int = 1; k <= N; k++) {
    val kk = k;
    async {
        atomic total += calculate_partial_sum(kk);
    }
}
```

when

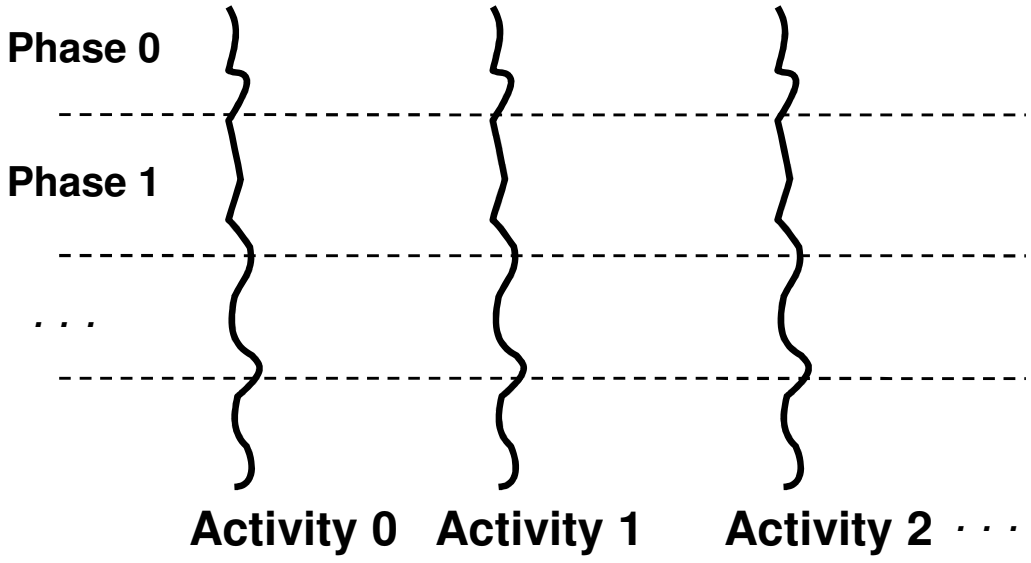
when (E) { S }

- Условная атомарность
- Текущая активность блокируется до тех пор, пока булевское выражение E не примет значение true
- Выражение S выполняется атомарно (atomic)
- Булевское выражение E
  - Неблокирующее
  - Последовательное (не должно создавать активностей)
  - Локальное (не должно обращаться к удаленным данным)
  - Не имеет побочных эффектов (side effects)

```
class sendReceive {

    var datum:Object = null;
    var isFilled:Boolean = false;

    def send(v:Object) {
        when (!isFilled) {
            datum = v;
            isFilled = true;
        }
    }

    def recv():Object {
        when (isFilled) {
            val v = datum;
            datum = null;
            isFilled = false;
            return v;
        }
    }

};
```

clocks

- Координация активностей с помощью finish базируется на окончании работы
- В некоторых случаях необходимо синхронизировать активности «при жизни»
  - Например, итерации алгоритма
- clock ~= MPI_Barrier
- async clocked(cl) S
- Clock.advanceAll() блокирует текущую активность до тех пор, пока все активности, зарегистрированные на данном таймере, не выполнят advanceAll()




```
clocked finish {
    clocked async {
        phase("A", 1);
        Clock.advanceAll();
        phase("A", 2);
    }

    clocked async {
        phase("B", 1);
        Clock.advanceAll();
        phase("B", 2);
    }
}
```

Аналогии между понятиями X10 и MPI/OpenMP

- X10 place ~= MPI процесс
- Активности ~= потоки в различных процессах
- Одиночный таймер (clock) ~= MPI_Barrier.
- Коллективные операции X10 ~= коллективные операции MPI
- X10 является более универсальным, чем MPI в некоторых случаях
  - Не требуется явная синхронизация для чтения/записи переменных в других процессах (~= односторонние операции)
  - Существуют простые средства для реализации атомарности
  - Множественные барьеры (могут быть использованы при многомасштабном моделировании физических процессов)
  - Не только SPMD парадигма

Текущий статус X10

- X10 2.2.0
  - Open Source: Eclipse Public License v1
  - Спецификация языка стабилизирована, последующие релизы будут обратно совместимы
  - Производительность пока не дотягивает до оптимальной
  - Основное внимание пока уделяется функциональности и устранению ошибок
- Две среды выполнения
  - C++
    - Multi-process (one place per process, multi-node)
    - Linux, AIX, Mac OS, Cygwin, Blue Gene
    - x86, x86_64, PowerPC
  - JVM
    - Multi-process (one place per JVM process, multi-node)
    - Windows single process only for now
    - Runs on any Java 5, Java 6 JVM
- X10DT IDE
  - Основана на Eclipse 3.6
  - Windows, Linux, Mac OS

Основные направления развития

- Развитие Cilk-style work stealing планировщика
- Глобальная балансировка нагрузки
- Коллективные операции
- X10 -> CUDA
- Поддержка различных архитектур для places в одной программе
- Библиотеки, фреймворки, …
  - Работа с матрицами
  - MapReduce

Некоторые известные проблемы

- Количество активностей (async)
  - Порождать столько активностей (async), сколько доступно физических потоков (threads)
  - Порождение большого числа активностей (async) может привести к out of memory crash в runtime среде
- В текущем релизе производительность операций с DistArray (finish, ateach) не оптимальна, т.к. реализована через point-to-point операции
  - На большом числе Places масштабирование затруднено
  - Будет исправлено в будущих релизах

Заключение

- X10 позаимствовал многое из OO языков
  - Классы, наследование, обобщенное программирование (Array[Int]), …
- X10 позаимствовал многое из функциональных языков
  - Анонимные функции, замыкания, …
- X10 похож на Java
  - Garbage collection
  - Математические функции
- X10 позволяет использовать нативный код на C/C++, Java
- X10 еще очень молодой, активно развивающийся язык
  - Многое не задокументировано
  - Многое не работает
  - Производительность пока оставляет желать лучшего
- Концепция APGAS сильно отличается от традиционного MPI/OpenMP подхода
  - Но, возможно, именно она скоро будет доминировать

Источники информации

- www.x10-lang.org
- “X10 as a parallel language for scientific computation: practice and experience” Milthorpe, J., Rendell, A., Grove, D.
  - Статья в открытом доступе