Vijay Saraswat - X10

How do you write code to run on:

50,000 cores

↑

very large number cores

Resilient X10

↑

~2000 done work

Scala use of concurrency

High performance computing at scale

Question — what if one node goes down

Get 1 construct

↳ automatically handle node failures

X10 - understand failure of nodes

- programmer aware & write compensating behavior

write own data structures to deal w/ this

APGAS programming model

asyne ~~part~~ partitioned global address space

→ realized natively in X10

Extension of APGAS to resilience

Resilient APGAS

alternatives:
(NPI open NP)

operational semantics

resilience is a "slippery" concept
⇒ formal semantics
abstracted version of language

happens before invariance principle

HBI

X10 - imperative language
problems - mutating state,
arrays

arises issues, memory model, etc.

Managed X10 (on JVM)
better than Spark

Spark - RDD data structure
resilience built on this

ALT research massive asynchrony (global load balancing)
irregular computation on large clusters

richer in X10 than in Spark

X10 language

Scala-like syntax
own language (Scala too young)
if redone, would be Scala-based

- richer type system than Java
- constrained type system
- variables before type syntax

(no supports for Scala implicits)

focus on scale program
many nodes
each w/ many threads
parallelism & distribution
focus: productivity

ex - 1000 nodes
- imperative

What kind of coding model give to prog

any prog in any thread can access global data structure

- local
- remote - transparent to other node

4-5 diff orders of magnitude latency

(DSM - no guarantee about performance)

PGAS - similar

but we added "twist"

↳ Heap is not flat

introduce:

Place

↳ "OS process" (intuition)

memory (virtual

threading

run programs

APGAS

surface places in program model.

- worry about places (data)
- where that state lives

asynchrony is critical (change from PGAS)

~~Single Pla~~

SPMD Single Program Multiple Data

ex. 2D array

ex. weather prediction
ex. heat transfer across wall of room

large array broken into cells

cell - value point in time

run w/ many points in time

end w/ gradient

write code once

distribute to places

send signal to all places

run till hit barrier

shuffling of data

ex. wall broken into 100 × 100 grid

physical boundary adjacent
may be far away node

periodic cyclic boundary condition
variable

- local compute
- boundary
- shuffle
- reduce

want SPMD runs in (A)PGAS model

want 100,000 nodes
problem to impl efficiently

APGAS

1) places
2) asynchrony

} programmer deals with

designed
available
} for real work

native or managed?
(JVM)

raw machine GC, threading

(HPC) (Enterprise)

IBM
(BOTH!)

(most features work on both)

open-source from beginning

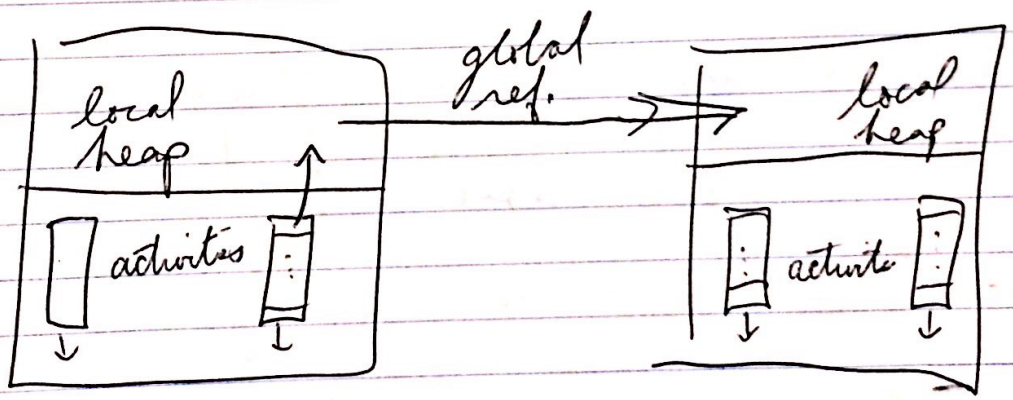
Eclipse - constrains usage
e.g. incremental type-checking

NPI - share nothing abstraction
(no asynchrony)

Open NP - exec large tasks in threads

X - CUDA-like accelerator

Our goal - 1 prog. model



Task parallism
~2004 · async S
· finish S

updates happen in an order

finish (dual to async)

S - tree of async comp.

run till all are terminated

async - not return "thread" obj
design choice

~~data abs~~

at runtime could be many
or few threads

Place-shifting

at(p) S

↳ - stop curr thread
- move state to 'p'
- then exec 'S' 'p'

no restriction on nesting

synchronized - waits for S to complete (?)

at(p) e ⇝ expression-version returns value

nested structures
code

- refer to outer scope in same way
- leave to runtime to get a handle on all references in scope (read-only)

only can mutate in its place

GlobalRef[T]

new Obj, get local ref

pass to global ref.

Generics not impl by erasure (!)
⇒ ∃ runtime cost

can ask
is pointing to local obj?
if yes, can get it back

PlaceLocalHandle[T]

notion: Global handle freely shared
mutable state, 1 place
PlaceLocal – local obj created in each place
collection

collection of objects
can only get the local obj for a place

Concurrency (least successful design)

atomic S

exec as if in a single step
no R/W mingle with this S
S could not have:
async
finish
at

when(c) S

conditional version
(Hoare - cond critic region)

only enter / exec S when c
but both atomic / one step

had to fall back on latches, locks
for performance
(ownership types)

remote eval

```
v = at (p) evalThere (arg1, arg2);
```

SPMD

```
finish for (p in Place.places()) {
    at (p) async runEverywhere();
}
```

! not resilient

(no barrier here)

introduce **clocks** for this lexically scoped barrier

Atomic remote update

```
at (ref) async atomic ref() += v;
```

Data Exg.

```
val _l = l();
finish at (r) async {
    val _r = r();
    r() = _l;
    at (l) async l() = _r;
}
```

not atomic w/resp to other updates

worked for 50,000 places

Sequential language

Java w/ Scala - syntax
model

not reusing Java compiler
(need C++ toolchain)
own compiler

global GC
(can have cycles across places)
(global Ref counting GC)

Java tricky obj creation
allows "proto" obj to be leaked
↳ hand-hat init

lang rules for in constructor
much stricter

doesn't leak proto-objects

value types - structs
(no mutable state
just copy freely)

(distinguish copy freely vs not)
struct obj

Java array Art Off < overhead problem

structs → "built-in" types created by program

ex. complex numbers

kept exceptions → resilience

powerful

resilient X10 - place crashes

only got to place non-zero

by an [a+]

[at] throws exception

thrown asyncly

already built for resilience

Rail vs Array

↓ ↓

multi-dim array Java-style

distrib arrays