Hierarchical Pointer Analysis for Distributed Programs

Amir Kamil

U.C. Berkeley

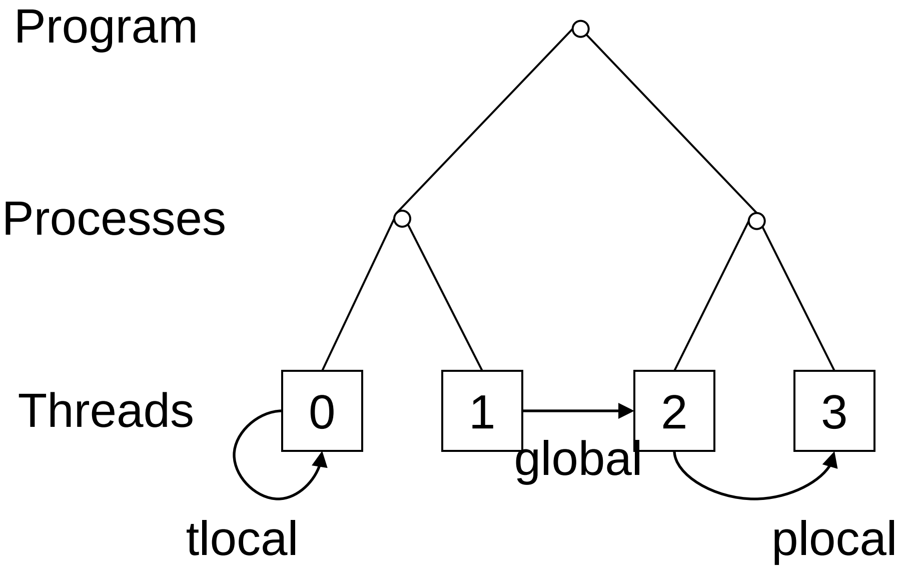
December 7, 2005

Background

- **Titanium is a single program, multiple data (SPMD) dialect of Java**
  - All threads execute the same program text
- **Designed for distributed machines**
- **Global address space – all threads can access all memory**
- **At runtime, threads are grouped into processes**
  - A thread shares a physical address space with some other, but not all threads

Memory Hierarchy

- **Global memory is composed of a hierarchy**



- **Locations can be thread-local (tlocal), process-local (plocal), or in another process (global)**

Goal

- **Our goal is to produce a (flow-insensitive) pointer analysis that takes the memory hierarchy into account**
- **We define a small SPMD language based on Titanium**
- **We produce a type system that accounts for the memory hierarchy**
- **We give an overview of the abstract pointer analysis**

Language Syntax

- **Types**

  τ ::= int | ref$_q$ τ

- **Qualifiers**

  q ::= tlocal | plocal | global

  (tlocal @plocal @global)

- **Expressions**

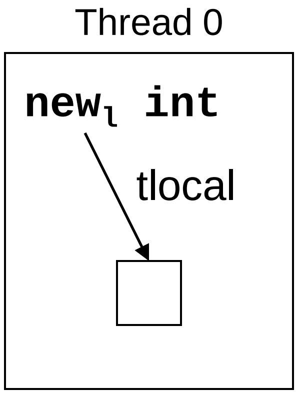
  e ::= new$_l$ τ (allocation)

  | transmit e_1 from e_2 (communication)

  | e_1 Ã e_2 (dereferencing assignment)

Type Rules – Allocation

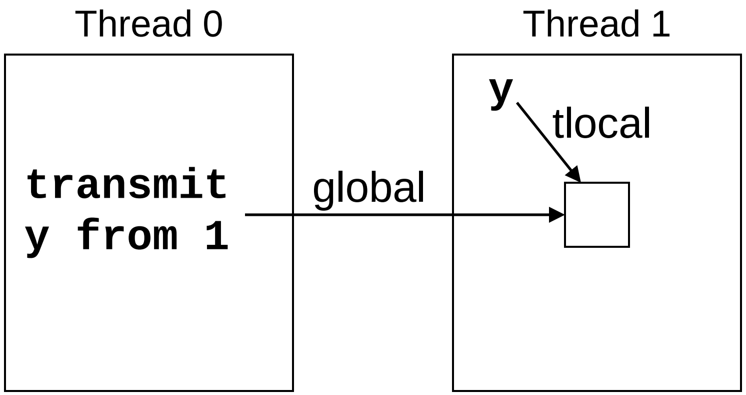
- **The expression new$_l$ τ allocates space of type τ in local memory and returns a reference to the location**
  - The label *l* is unique for each allocation site and will be used by the pointer analysis
  - The resulting reference is qualified with tlocal, since it references thread-local memory



$$\frac{}{\Gamma \text{ ` } \textbf{new}_l\ \tau : \textbf{ref}_{\textbf{tlocal}}\ \tau}$$

Type Rules – Communication

- **The expression transmit e_1 from e_2 evaluates e_1 on the thread given by e_2 and retrieves the result**
- **If e_1 has reference type, the result type must be widened to global**
  - Statically do not know source thread, so must assume it can be any thread



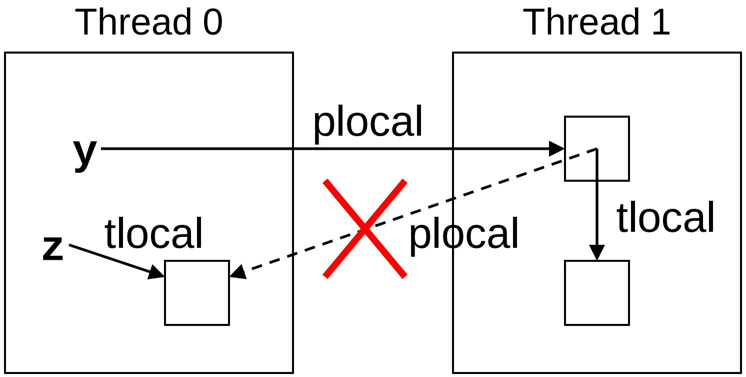
$$\frac{\Gamma\ \grave{}\ \ \mathbf{e_1 : \tau} \qquad \Gamma\ \grave{}\ \ \mathbf{e_2 : int}}{\Gamma\ \grave{}\ \ \textbf{transmit } \mathbf{e_1} \textbf{ from } \mathbf{e_2} : \textit{\textbf{expand}}(\tau, \textit{\textbf{global}})}$$

$$\boxed{\begin{array}{c} \textit{\textbf{expand}}(\textit{\textbf{ref}}_q\ \tau, q') \ \acute{}\ \ \textbf{ref}_{t\ (q,\ q')}\ \tau \\ \textit{\textbf{expand}}(\tau, q') \ \acute{}\ \ \tau \textbf{ otherwise} \end{array}}$$

Type Rules – Dereferencing Assignment

- The expression e_1 Ã e_2 puts the value of e_2 into the location referenced by e_1 (like $*e_1 = e_2$ in C)
- If e_1 has type $\text{ref}_{\text{plocal}}\ \text{ref}_{\text{tlocal}}\ \tau$, and e_2 has type $\text{ref}_{\text{tlocal}}\ \tau$, the assignment could be unsound

$$\frac{\Gamma \text{ ` } e_1 : \text{ref}_q\ \tau \qquad \Gamma \text{ ` } e_2 : \tau \qquad \textit{robust}(\tau, q)}{\Gamma \text{ ` } e_1 \text{ Ã } e_2 : \text{ref}_q\ \tau}$$

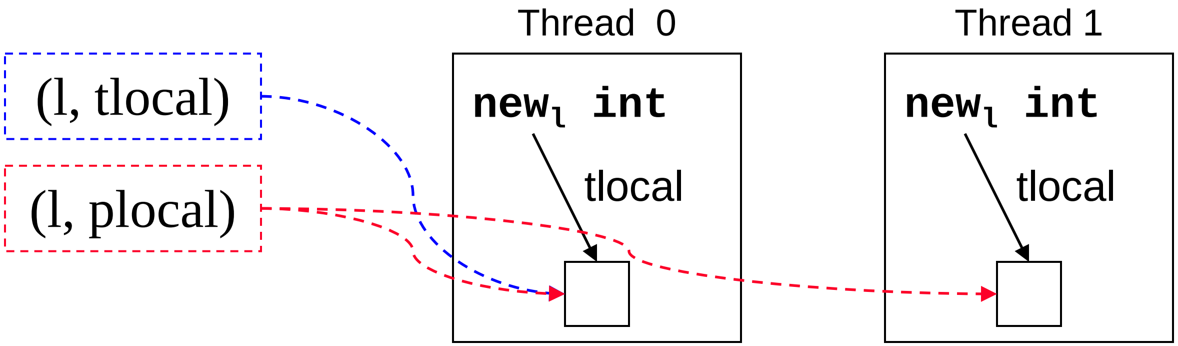


robust$(\textit{ref}_q\ \tau, q')$ ´ ***false*** **if** q @ q'

robust(τ, q') ´ ***true*** **otherwise**

Pointer Analysis

- **Since language is SPMD, analysis is only done for a single thread (assume thread 0)**
- **Each expression has a points-to set of *abstract locations* that it can reference**
- **Abstract locations also have points-to sets**
- **Abstract locations consist of label and qualifier**
  - A-loc (l, q) can refer to any concrete location allocated at label l and with type qualifier v q from thread 0



Pointer Analysis – Allocation and Communication

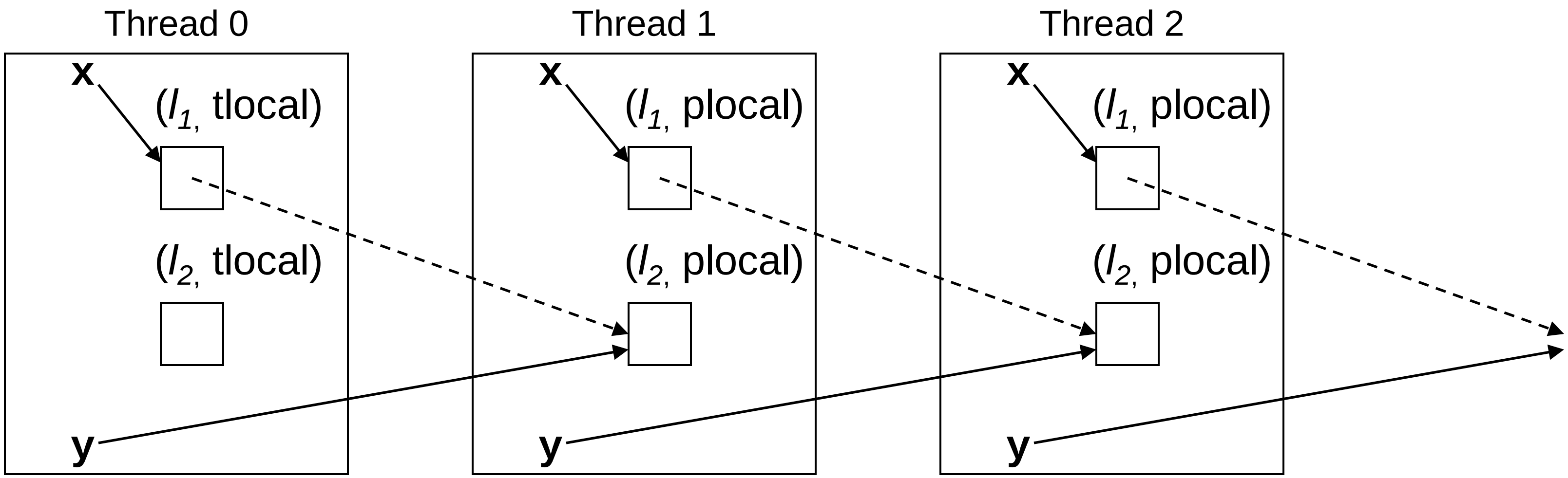
- The abstract semantics for allocation and communication are similar to the type rules
- An allocation $\text{new}_l\ \tau$ produces a new abstract location (l, tlocal)
- The result of the expression transmit e_1 from e_2 is the global versions of the a-locs resulting from e_1

$$e_1\ !\quad \{(l_1, \text{tlocal}), (l_2, \text{plocal}), (l_3, \text{global})\}$$

$$\text{transmit } e_1 \text{ from } e_2\ !\quad \{(l_1, \text{global}), (l_2, \text{global}), (l_3, \text{global})\}$$

Pointer Analysis – Dereferencing Assignment

- **For assignment, must take into account actions of other threads**



x ! $\{(l_1, \text{tlocal})\}$,

y ! $\{(l_2, \text{plocal})\}$

➡

x Ã y : (l_1, tlocal) ! (l_2, plocal),

(l_1, plocal) ! (l_2, plocal),

(l_1, global) ! (l_2, global)

Race Detection Results

- Static race detection in Titanium using pointer analysis + concurrency analysis
- Most are false positives, so lower is better

Benchmark	No Pointer Analysis	One-Level Analysis	Two-Level Analysis
gas	2482	779	223
gsrb	512	187	18
lu-fact	490	177	1
pps	7026	1827	26
spmv	443	177	0