Threads for Interoperable Parallel Programming

Presenter: S. Krishnan

Authors: L. V. Kale, J. Yelon, T. Knauff

Parallel Programming Laboratory

Department of Computer Science

University of Illinois, Urbana

Converse Goals

- To facilitate building of runtime systems for parallel languages.
- To achieve portability across many platforms.
- Multilingual Interoperability:
  - the ability to link modules written in different parallel languages.

Converse Threads: Features

1. User control over thread scheduling.

2. User control over preemption and nonpreemption.

3. Low-cost primitive abstractions.

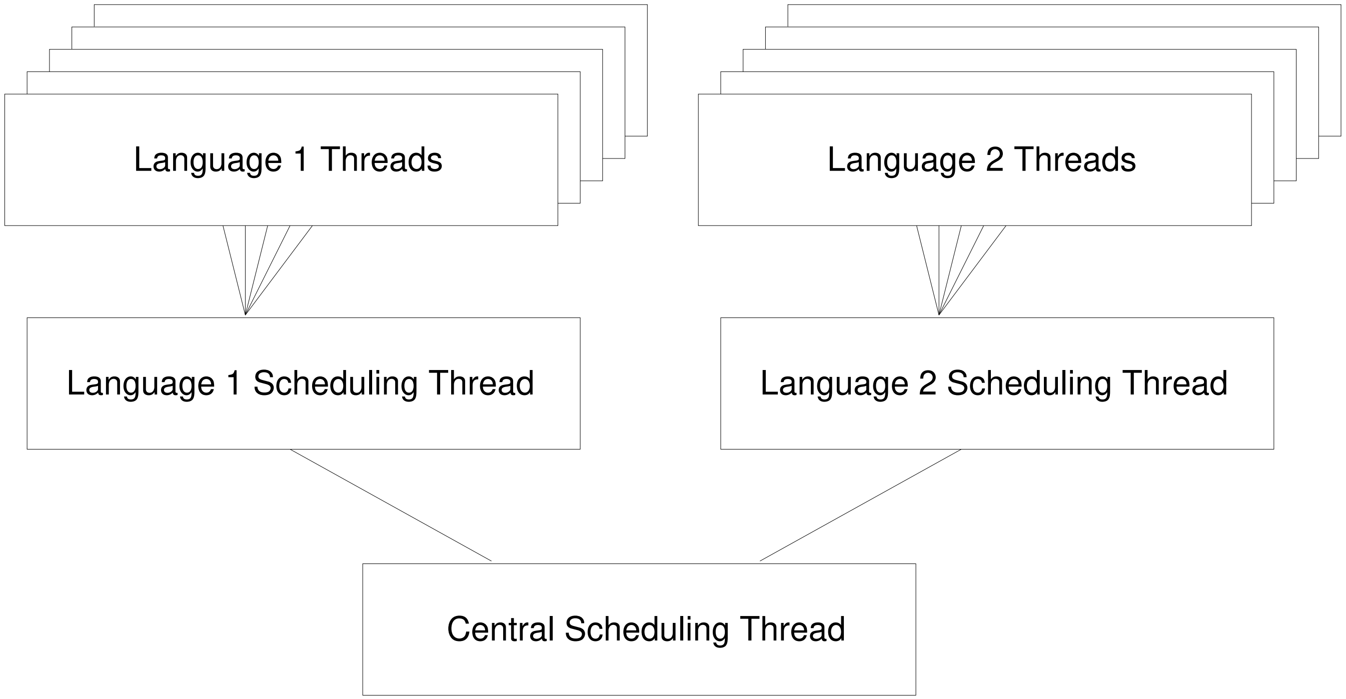
4. Useful model for shared and private data.

Feature 1: User control over thread scheduling

A good thread-scheduler should support all kinds of thread execution orders. Such orders might include:

- Critical-path intensive.
- Depth-First.
- Resource-bound.

User control over thread scheduling



User control over scheduling can be achieved using a hierarchical scheduling model.

Feature 2: User control over Preemption

- Problems with Preemptive Context Switching:
  - Must write all subroutines to be reentrant.
  - High cost due to frequent locking.
- Problems with Manual Context-Switching:
  - Increased possibility of I/O latency.
  - Priorities not kept as current as with preemption.
  - Preemption is part of many Parallel Programming models.
- Converse Solution:
  - allow each thread to choose whether it is preemptible.

Feature 3: Low-Cost Abstraction Layer

- Parallel programs may create thousands of threads.
- Threads have to be inexpensive.
- Synchronization abstractions have to be inexpensive.
- Thread-Private data must be inexpensive.

Low-Cost Abstraction Layer: Primitives

- Explicit context-switching.
- Explicit thread-handles and thread-queues.
- Macro interface to thread-private data.
- Multiple levels of sharing/synchronization.

Feature 4: a Usable Model for Shared Data.

Three possible levels of data-sharing are recognized:

- Thread-Private Data (completely unshared).
- Processor-Private Data (shared by all threads on 1 CPU).
- Semi-Shared Data (shared within one physical address space).

Simulated "global variables" are provided with each level of sharing.

Processor-Private Variable Declaration and Use

```
CpvDeclare(int, x);

incx() { CpvAccess(x) = CpvAccess(x) + 1; }

main() { CpvInitialize(int, x); ... }
```

Converse Threads API: Creation/Destruction

- typedef struct CthThreadStruct *CthThread;

  - the thread datatype, an opaque type.

- CthThread CthCreate(void (*fn)(void *), void *arg, int size)

  - create a thread, return its handle.

- CthThread CthSelf()

  - return the thread which is currently executing.

- void CthFree(CthThread t)

  - free the specified thread

Converse Threads API: Context-Switching

- void CthResume(CthThread t)

  - immediately context switch to thread t

- void CthSuspend();

  - transfer to any thread in the appropriate ready-queue.

- void CthAwaken(CthThread t);

  - add thread t to the appropriate ready-queue.

- void CthYield();

  - put self on ready-queue, then yield to another thread

Converse Threads API: High-Level Control

- void CthSigYieldEnable(int flag);

  - enable/disable preemption of the current thread.

- void CthAutoYield(int usec);

  - enable time-sliced preemtion of current thread.

- **void CthSetStrategy(CthThread t, ...)**

  - choose ready-queue and scheduler to use for thread t.

Converse Threads: Features

1. User control over thread scheduling.
2. User control over preemption and nonpreemption.
3. Low-cost primitive abstractions.
4. Useful model for shared and private data.