The ConCert Project
Trustless Grid Computing

Robert Harper
Carnegie Mellon University
May, 2002

Credits

- Co-PI's
  - Karl Crary
  - Peter Lee
  - Frank Pfenning
- Students
  - Tom Murphy, Evan Chang, Margaret Delap, Jason Liszka
  - Many, many others!

Grid Computing

- Zillions of computers on the internet.
- Lots of wasted cycles.
- Can we harness them?

Grid Computing

- Some well-known examples:
  - SETI@HOME
  - GIMPS
  - FOLDING@HOME
- Each is a project unto itself.
  - Hosts explicitly choose to participate.

Grid Computing

- Many more solutions than applications.
  - Common run-time systems.
  - Frameworks for building grid app's.
- Many more problems than solutions.
  - How to program the grid?
  - How to exploit the grid efficiently?
- Lots of interest, though!
  - Regularly in the NYTimes.

Some Issues

- Trust: hosts must run foreign code.
  - Currently on a case-by-case basis.
  - Explicit intervention / attention required.
  - Is it a virus?
    - Safety: won't crash my machine.
    - Resource usage: won't soak up cycles, memory.

Some Issues

- Reliability: application developers must ensure that hosts play nice.
  - Hosts could maliciously provide bad results.
  - Current methods based on redundancy and randomization to avoid collusion.

Some Issues

- Programming: how to write grid app's?
  - Model of parallelism?
    - Massively parallel.
    - No shared resources.
    - Failures.
  - Language? Run-time environment?
    - Portability across platforms.
    - How to write grid code?

Some Issues

- Implementation: What is a grid framework?
  - Establishing and maintaining a grid.
  - Distribution of work, load balancing, scheduling.
  - Fault recovery.
  - Many different applications with different characteristics.

Some Issues

- Applications: Can we get work done?
  - How effectively can we exploit the resources of the grid?
    - Amortizing overhead.
  - Are problems of interest amenable to grid solutions?
    - Depth > 1 feasible?

The ConCert Project

- Trustless Grid Computing
  - General framework for grid computing.
  - Trust model based on code certification.
  - Advanced languages for grid computing.
  - Applications of trustless grid computing.
- Interplay between fundamental theory and programming practice.
  - Model: The Fox Project.

Trustless Grid Computing

- Minimize trust relationships among applications and hosts.
  - Increase flexibility of the grid.
  - "The network as computer", with many keyboards.
  - Avoid explicit intervention by host owners for running a grid application.

Trustless Grid Computing

- Adopt a policy-based framework.
  - Hosts state policy for running grid applications in a declarative formalism.
  - Application developers must prove compliance with host policies.
  - Proof of compliance is mechanically checkable.

Trustless Grid Computing

- Example policies:
  - Type- and memory safety: no memory overruns, no violations of abstraction boundaries.
  - Resource bounds: limitations on memory and cpu usage.
  - Authentication: only from .edu, only from Robert Harper, only if pre-negotiated.

Trustless Grid Computing

- Compliance is a matter of proof!
  - Policies are a property of the code.
  - Host wishes to know that the code complies with its policies.
- Certified binary = object code plus proof of compliance with host policy.
  - Burden of proof is on the developer.
  - Hosts simply state requirements.

Code Certification

- Example: type safety.
  - Source language enjoys safety properties.
    - Eg, Java, Standard ML, Safe C.
  - Compiler transfers safety properties to object code.
    - Depends on compiler correctness.
  - But the compiler “knows why” the object code is type safe!

Certifying Compilers

- Idea: propagate types from the source to the object code.
  - Can be checked by a code recipient.
  - Avoids reliance on compiler correctness.
- Needs a new approach to compilation.
  - Typed intermediate languages.
  - Type-directed translation.

Typed Intermediate Languages

- Generalize syntax-directed translation to type-directed translation.
  - Intermediate languages come equipped with a type system.
  - Compiler transformations translate both a program and its type.
  - Translation preserves typing: if e:t then e*:t* after translation.

Typed Intermediate Languages

- Classical syntax-directed translation:

  Source = $L_1 \rightarrow L_2 \rightarrow \ldots \rightarrow L_n$ = target.

  :

  T_1.

- Type system applies to the source language only.
  - Type check, then throw away types.

Typed Intermediate Languages

- Type-directed translation:

  Source = $L_1 \rightarrow L_2 \rightarrow \ldots \rightarrow L_n$ = target.

  $\quad\quad\quad

Typed Object Code

- Typed assembly language (TAL)
  - Type information ensures safety
  - Generated by compiler
  - Very close to standard x86 assembly
- Type information captures
  - Types of registers and stack
  - Type assumptions at branch targets (including join points)

Typed Assembly Language

```
fact:
   ALL rho.{r1:int, sp:{r1:int, sp:rho}::rho}
  jgz r1, positive
  mov r1,1
  ret
positive:
  push r1  ; sp : int::{t1:int,sp:rho}::rho
  sub r1,r1,1
  call fact[int::{r1:int,sp:rho}::rho]
  imul r1,r1,r2
  pop r2   ; sp : {r1:int,sp:rho}:: ret
```

Certifying Compilers

- SpecialJ: Java byte code.
  - Generates x86 machine code.
  - Formal proof of safety in a formalized logic represented as an LF term.
- PopCorn: Safe C dialect.
  - Also generates x86 code.
  - Certificate consists of type annotations on assembly code.

Certifying Compilers

- What can we certify?
  - Type and memory safety.
    - Including system call or device access.
  - Authenticity.
    - Code signing.
- What might we be able to certify?
  - Resource usage: memory bounds, time bounds.
  - But there are hard problems here!

The ConCert Framework

- Each host runs a steward.
  - Locator: building the grid.
  - Conductor: serving work.
  - Player: performing work.
- Inspired by Cilk/NOW (Leiserson, et al.)
  - Work-stealing model.
  - Dataflow-like scheduling.

The ConCert Framework

- The steward is parameterized by the host policy.
  - But currently it is fixed to be TAL safety.
- Host can either trust our steward or write her own!
  - Declarative formalism for policies and proofs.
  - Essentially just a proof checker.

The Locator

- Peer-to-peer discovery protocol.
  - Based on GnuTella ping-pong protocol.
  - Hosts send ping's, receive pong's.
  - Start with well-known neighbors.
- Generalize file sharing to cycle sharing.
  - State willingness to contribute cycles, rather than music files.

The Conductor

- Serves work to grid hosts.
  - Implements dataflow scheduling.
  - Unit of work: chord.
  - Entirely passive.
- Components:
  - Listener on well-known port.
  - Scheduler to manage dependencies.

Player

- Executes chords on behalf of a host.
  - Stolen from a host via its conductor.
  - Sends result back to host.
  - Ensures compliance with host policy.
- Components:
  - Communication with neighboring conductors.
  - Proof check for certified binaries.

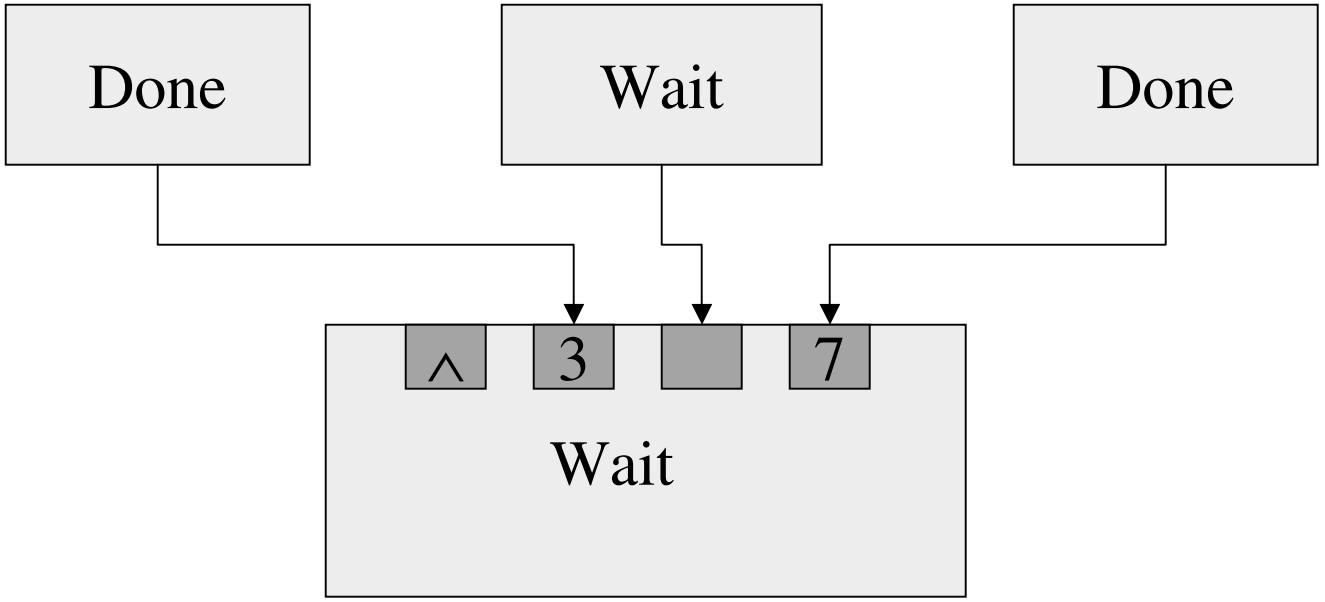
Chords

- A task is broken into chords.
  - A chord is the unit of work distribution.
  - Chords form nodes in an and/or dependency graph (dataflow network).
- Conductor schedules cords for stealing.
  - Ensures dependencies are met.
  - Collects results, updates dependencies.

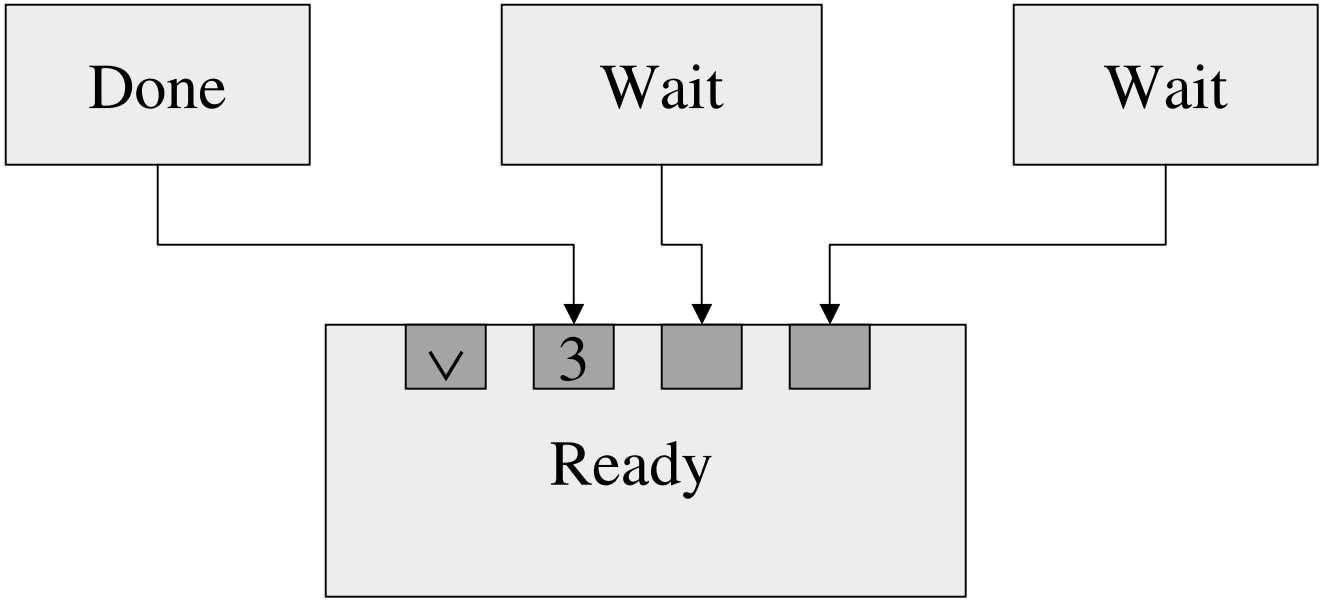
Chords

- A chord is essentially a closure:
  - Code for the chord.
  - Bindings for free variables.
  - Arguments to the chord.
  - Type information / proof of compliance.
- Representation splits code from data.
  - Facilitates code sharing.
  - Reduces network traffic.
  - MD5 hash as a code pointer.

Chord Scheduling



Chord Scheduling



Failures

- Simple fail-stop model.
  - Processors fail explicitly, rather than maliciously.
  - Timeout's for slow or dead hosts.
- Assume chords are repeatable.
  - No hidden state in or among chords.
  - Easily met in a purely functional setting.

Application: Ray Tracing

- GML language from ICFP01 programming contest.
  - Simple graphics rendering language.
  - Implemented in PopCorn.
  - Generates TAL binaries.
- Depth-1 and-dependencies only!
  - Divide work into regions.
  - One chord per region.

Application: Parallel Theorem Proving

- Fragment of linear logic.
  - Sufficient to model Petri net reachability.
  - Stresses and/or dependencies, depth > 1.
  - Focusing strategy to control parallelism.
- Currently uses grid simulator.
  - No certifying ML compiler.
  - Requires linguistic support for programming model.

A Programming Model

- An ML interface for grid programming.
  - Task abstraction.
  - Synchronization.
- Theorem prover uses this interface.
  - Maps down to chords at the grid level.
  - Currently only simulated, for lack of suitable compiler support.

A Programming Model

```
signature Task = sig
  type 'r task
  val inject : ('e -> 'r) * 'e -> 'r task
  val enable : 'r task -> unit
  val forget : 'r task -> unit
  val status : 'r task -> status
  val sync : 'r task -> 'r
  val relax :
    'r task list -> 'r * 'r task list
end
```

Example: Mergesort

```
fun mergesort (l) =
  let
    val (lt, md, rt) =
      partition ((length l) div 3, l)
    val t1 = inject (mergesort, lt)
    val t2 = inject (mergesort, md)
    val t3 = inject (mergesort, rt)
    val (a, rest) = relax [t1,t2,t3]
    val (b, [last]) = relax [rest]
  in
    merge (merge (a, b), sync last)
  end
```

Tasks and Chords

- A task is the application-level unit of parallelism.
  - Cf mergesort example
- A chord is the grid-level unit of work.
- Tasks spawn chords at synch points.
  - Each synch creates a chord.
  - Dependencies determined by the form of the synch.

Malice Aforethought

- What about malicious hosts?
  - Deliberately spoof answers.
  - Example: TP always answers “yes”.
- What about malicious failures?
  - Arbitrary bad behavior by hosts.

Result Certification

- Solution: prove authenticity of answers!
  - Application computes answer plus a certificate of authenticity.
  - Example: GCD(m,n) returns (d,k,l) such that d = km+ln and d|m and d|n.
  - Example: TP computes a formal proof of the theorem!
- Cf. Blum's self-checking programs.
  - Probabilistic methods for many problems.

Summary

- ConCert: a trustless approach to grid computing.
  - Hosts don't trust applications.
  - Applications don't trust hosts.
- Lots of good research opportunities!
  - Compilers, languages.
  - Systems, applications.
  - Algorithms, semantics.

Project URL

`http://www.cs.cmu.edu/~concert`