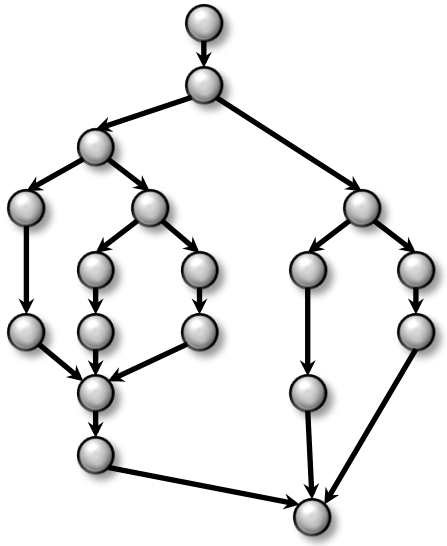
Parallelism without Concurrency

Charles E. Leiserson
MIT

Parallelism versus Concurrency

Parallel computing
A form of computing in which computations are broken into many pieces that are executed simultaneously.



Concurrent computing
A form of computing in which computations are designed as collections of interacting processes.

Eschew Concurrency

- Concurrent computing is hard because interacting processes are inherently nondeterministic and risk concurrency anomalies such as deadlock and races.
- Nondeterministic programs are difficult to understand and even more difficult to get right [Lee 2006, Bocchino et al. 2009] .
- To exploit multicore computers, average application programmers desperately need the field of parallel programming to move away from concurrency* and toward determinism (simplicity).

---

*Concurrent computing is still essential for programming parallel and distributed workloads, as well as for implementing concurrency platforms for parallel programming. This work is best done by experts in concurrency, however, not by average programmers.

Theory Success: Cilk Technology

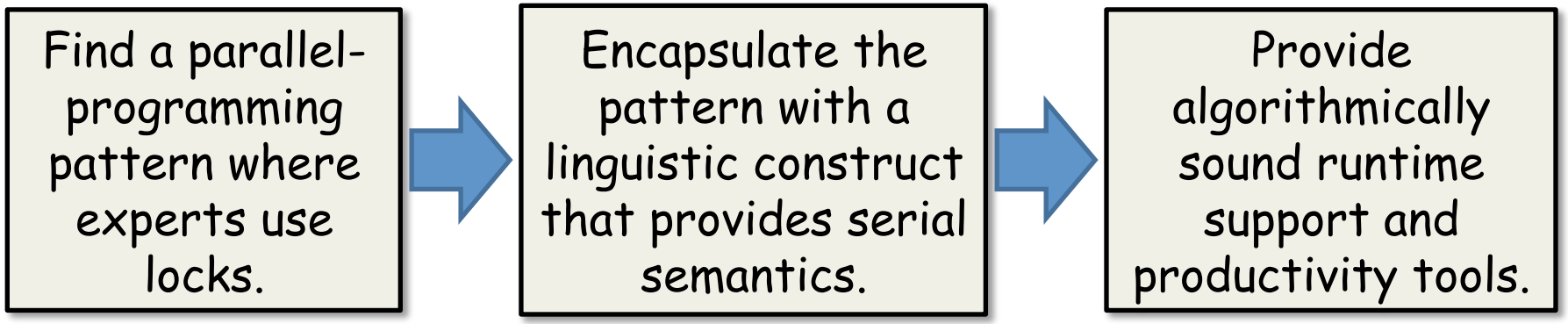
- Cilk encapsulates the nondeterminism of scheduling, allowing average programmers to write deterministic parallel codes using only 3 keywords to indicate logical parallelism.
- Cilk has a provably good scheduler that allows programmers to analyze scalability theoretically and in practice using work/span analysis.
- Cilk's serial semantics factors parallelism from the other issues of performance engineering, e.g., Cilk is cache friendly.
- The Cilkscreen race detector offers provable guarantees of determinism by certifying the absence of determinacy races.
- Cilk's reducer hyperobjects encapsulate the nondeterminism of updates to nonlocal variables, providing deterministic behavior for parallel updates.
- Cilk is embedded in Intel's C++ compiler, and Intel has released an open-source GCC implementation:

  `http://software.intel.com/en-us/articles/intel-cilk-plus/`

Proposed "Dangerous" Research Agenda

Applications still exist that cannot be easily programmed with Cilk's fork-join programming model except by resorting to concurrency, e.g., software pipelining, task-graph execution, etc.

Iterate until Done



Potential Recipe for Disaster?

- Can a parallel-programming model provide a small set of linguistic mechanisms that completely eliminates the need for concurrency in parallel programming?
- Can the programming model be designed to support effective theoretical performance models, especially considering the vagaries of modern hardware models.
- Will the various linguistic mechanisms synergize or conflict with each other?
- What kind of memory model should the programming model export (e.g., support for priority writes and priority reserve)?
- Can theoretically sound productivity tools be created to aid the development of parallel programs?
- How large a codebase can the parallel-programming model support?

Scalability

- Algorithmic theory teaches us to study how software scales with problem size.
- Parallel computing demands that we study how software scales with number of processors.
- The real challenge will be how parallel-computing mechanisms scale with legacy codebase size.
- Concurrency is a major obstacle.

Source lines of code	Software artifact
< 1,000	Classroom examples
1,000–10,000	Benchmarks
10,000–100,000	Libraries
100,000–1,000,000	Applications
> 1,000,000	Software platforms