Assessing the State of the Art

Alexandre David 1.2.05 adavid@cs.aau.dk

Important Properties

Correctness

- much harder than sequential program
- P-independence: Same output on the same input regardless of the arrangement of processes.
 Try to remove sensitivity to interleavings.
- Global view languages preserve P-independent program behavior.
- Local view languages do not preserve it.
 - Locks, send, receive local view abstraction.
 - forall loops, barrier, reduce, scans global view abstraction.

Important Properties

Performance

- How much is enough?
 - Little inherent parallelism → low speedup, good concurrency → good speedup. Concurrency → efficiency.
 - Good locality good for caches, superlinear possible.

Scalability

- Effect when number of processors increases?
- Compare with size of the problem.

Portability

Performance portability. CTA model.

Evaluating POSIX Threads

- Powerful & flexible too much flexible.
 - Deadlocks, races, uncontrolled memory accesses. Any threads can write anything anywhere at any time.
 - Shared address space paradigm does not encourage locality – not good for performance.
 - Locks & condition variables not easy to use, against modularity & abstraction
 - Locks are not composable.
 - Locking is a global property (correctness + performance).

Evaluating POSIX Threads

- False sharing easy to obtain.
- Locking: not possible to hide it, difficult to specify in an interface.
 - Issues with deadlocks & performance.
- The argument that it is similar to sequential programs encourages programmers to write inefficient code.

Evaluating Java Threads

- Similar to POSIX threads.
- Hide some of the complexity.
 - But with the price of added unspecified behavior for threads & volatile memory.

Evaluating OpenMP

- Global view "language", clean and simple.
- Very easy to use but only simple forms of parallelism.

Evaluating MPI

- Thinner interface than pthreads, more restricted communication.
- But many low-level details must be specified. Very easy to get it wrong.
- P-dependent point-to-point communication but collective communication operations supported.
- Private memory paradigm, encourages locality, but efforts needed.
- Overhead of message passing encourages coarse grained parallelism – good for performance. Suitable for static distributions.
- Not so portable w.r.t. performance.

Evaluating PGAS Languages (Partitioned Global Address Space)

- Improve upon MPI with higher level mechanisms for communication.
 - Global view offered, global data structures.
 - But retain local view of computations.
- ZPL: Good concepts for parallel computations, encourages to think differently but unfamiliar concepts (regions, flooding...) no pointers, limited memory management, not object-oriented...

Lessons for the Future

- Hidden parallelism largely hidden from programmer.
- Locality always important. Some languages encourage it.
- Constrained parallelism too much flexibility or power is bad – force discipline on programmers.
 - Flexibility can allow interactions that are difficult to reason about – correctness issues.
 - Flexibility has performance issues if it obscures the performance model.
 - The goal is to make effective use of the available resources (locality, limit dependencies, sync,...) not to expose maximal parallelism.
 - Pthreads allows almost anything compare with other approaches.

Lessons – cont.

Implicit vs. explicit parallelism.

- What's the right level to expose it?
- Ex. GPU: shading routines are customized serial code, parallel code is written by the vendor.
- Other domain specific languages are very efficient.
- General vs. domain specific is like explicit (+general) vs. implicit (+convenient) parallelism.