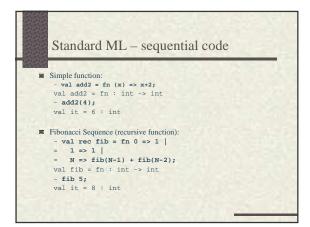


Standard ML - History

- Originally developed in the early 70s in Edinbourgh
- The name, ML, comes from Meta-Language
- Combines features of Algol and Lisp-like languages
- **#** Its first goal was to facilitate the development of a theorem proving package

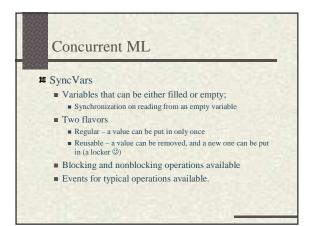


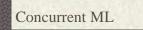
Standard ML - Higher Order Functions Applying a function to all elements of a list - val map = fn (f, nil) => nil | = (f, h::t) => (f h)::(map (f, t)); val map = fn : ('a -> 'b) * 'a list -> 'b list - map (add2, [1,2,3]); val it = (3,4,5) : int list We could also just return a function that would perform the operation... ...but we are to talk about concurrency, not ML @



Concurrent ML

- send/recv functions (blocking)
- sendPoll/recvPoll functions (nonblocking)
- sendEvt/recvEvt create events associated with
- sending/receiving messagesMulticast channels also available
- Events
- Events
 - sync synchronize on an event
 - ... and many other event related functions that could appear in any library supporting concurrency





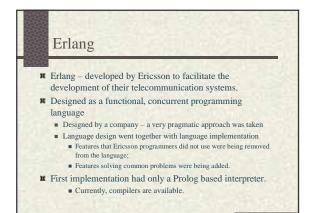
MailBoxes

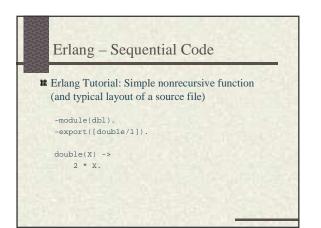
- A marriage of SyncVars and channels
 - Producer may put packages into the mailbox
 - Consumer may pick them up
 - No limit on the amount of packages in the mailbox, so if produces is much faster then we run out of memory
- Operations possible on a MailBox:
- send always nonblocking recv/recvPoll – blocking/nonblocking receive
- Events associated with mailboxes (receiving)

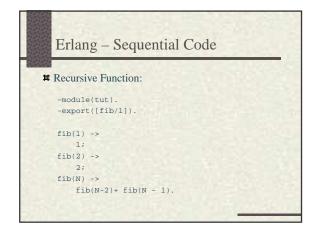
Concurrent ML

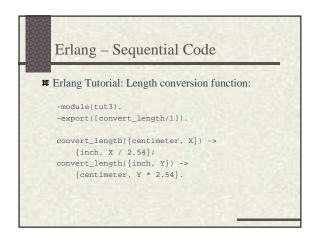
Summary

- Nice functional programming language...
- ... but the syntax has no support for concurrency
- Concurrency was added using a well designed library, but it did not facilitate concurrent programming too much (no more than pthreads facilitated concurrent programming in C/C++).
- Functional properties of ML seem to be leaving a lot of space for advanced support of parallel programming templates, but CML does not have them (at least not in a out-of-the-box fashion)





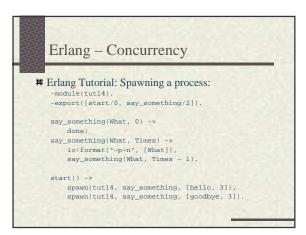


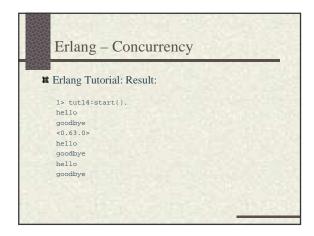


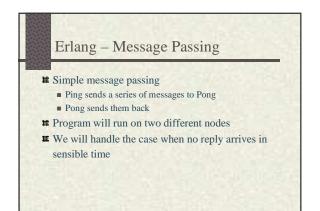
Erlang - Concurrency

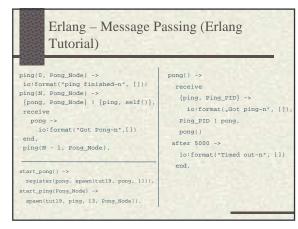
Erlang can spawn many processed

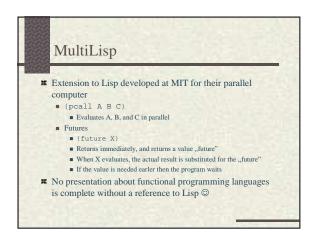
- No support for shared memory
- All communication through messages
- Messages may be addressed to specific PIDs...
- ...or sent to special named processes
- Remember! This language was developed by a telecommunication company to satisfy its own needs ^(C)

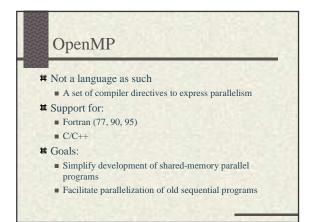


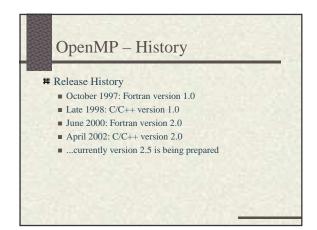


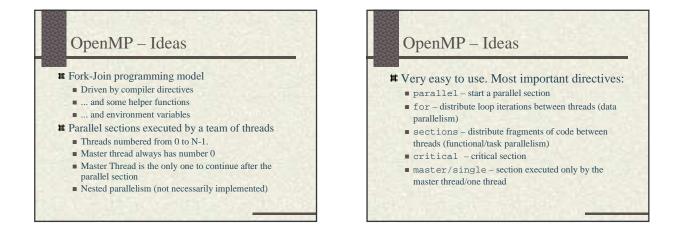


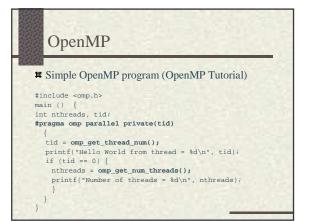


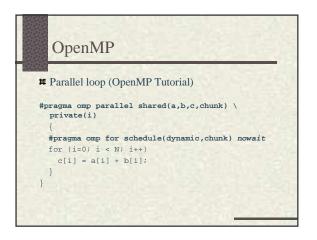










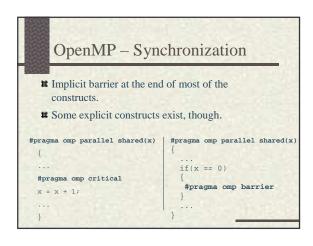


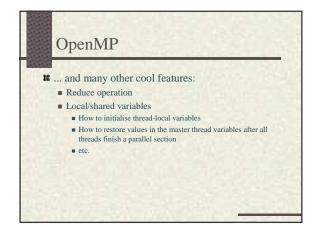
OpenMP

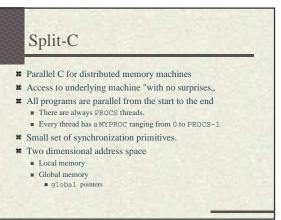
Parallel loop, but shorter (OpenMP Tutorial)

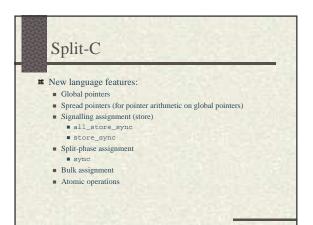
#pragma omp parallel for \
 shared(a,b,c,chunk) private(i) \
 schedule(static,chunk)
 for (i=0; i < n; i++)
 c[i] = a[i] + b[i];</pre>

OpenN	/IP	Sec.	1.2
# Functional	/Task parallelism		
#pragma omp	sections nowait		
{ #pragma	omp section		
	; i < N/2; i++)		
	a[i] + b[i];		
	omp section		
	/2; i < N; i++)		
c[i] =	a[i] + b[i];		
}			

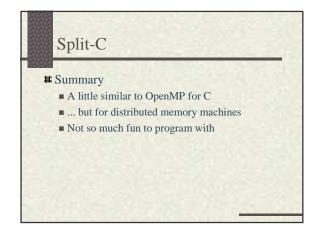


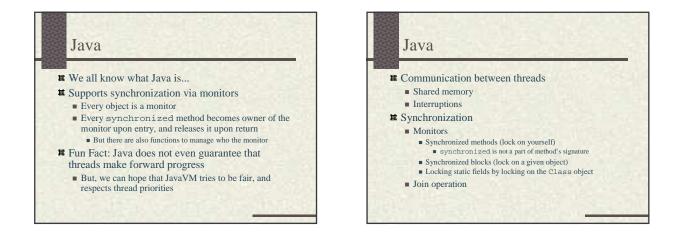


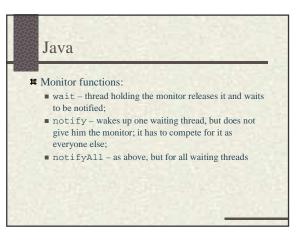


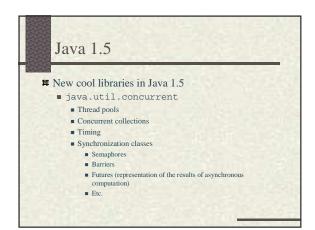












Summary

- # Different languages, different applications
 - Erlang satisfies industry's need of a language for quick development of distributed systems
 - OpenMP high performance computing on shared memory multiprocessors
 - Split-C high performance computing on distributed memory multiprocessors
 - Java general purpose programming language with native support for parallel computation