Should parallel languages be concurrent?

UW/MSR 2008

John Reppy

Department of Computer Science
University of Chicago

August 4, 2008
Overview

- What is the difference between parallel and concurrent programming?
- What is the role for concurrency in the multicore world?
- Some thoughts on language design for multicore systems.
Overview

- What is the difference between parallel and concurrent programming?
- What is the role for concurrency in the multicore world?
- Some thoughts on language design for multicore systems.
Overview

- What is the difference between parallel and concurrent programming?
- What is the role for concurrency in the multicore world?
- Some thoughts on language design for multicore systems.
What is the difference?

Parallel and concurrent programming address two different problems.

- Parallelism is about speed — exploiting parallel processors to solve problems quicker.
- Concurrency is about nondeterminism — managing the unpredictable external world.
What is the difference?

Parallel and concurrent programming address two different problems.

- Parallelism is about speed — exploiting parallel processors to solve problems quicker.
- Concurrency is about nondeterminism — managing the unpredictable external world.
What is the difference?

Parallel and concurrent programming address two different problems.

- Parallelism is about speed — exploiting parallel processors to solve problems quicker.
- Concurrency is about nondeterminism — managing the unpredictable external world.
What is the difference?

Application domains for parallel and concurrent languages differ.

- Traditional applications of parallelism (HPC, multimedia) do not benefit from non-determinism.
- Some concurrent languages have parallel implementations, but the effort is often not worth the benefit.
What is the difference?

Application domains for parallel and concurrent languages differ.

- Traditional applications of parallelism (HPC, multimedia) do not benefit from non-determinism.
- Some concurrent languages have parallel implementations, but the effort is often not worth the benefit.
What is the difference?

Application domains for parallel and concurrent languages differ.

▶ Traditional applications of parallelism (HPC, multimedia) do not benefit from non-determinism.

▶ Some concurrent languages have parallel implementations, but the effort is often not worth the benefit.
What is the difference?

Environments for parallel and concurrent languages differ.

- Parallel computing is dominated by HPC applications running on large, expensive machines.
- Traditional applications of concurrency (e.g., UIs and OS kernels) mostly run on single-processor desktop systems.
What is the difference?

Environments for parallel and concurrent languages differ.

- Parallel computing is dominated by HPC applications running on large, expensive machines.
- Traditional applications of concurrency (e.g., UIs and OS kernels) mostly run on single-processor desktop systems.
What is the difference?

Environments for parallel and concurrent languages differ.

- Parallel computing is dominated by HPC applications running on large, expensive machines.
- Traditional applications of concurrency (e.g., UIs and OS kernels) mostly run on single-processor desktop systems.
Should concurrent languages be used for parallelism?

Concurrent languages are not necessarily well suited to parallel programming.

- hard to get grain-size right
- scheduling for parallel computation
- nondeterminism
Should concurrent languages be used for parallelism?

Concurrent languages are not necessarily well suited to parallel programming.

- hard to get grain-size right
- scheduling for parallel computation
- nondeterminism
Should concurrent languages be used for parallelism?

Concurrent languages are not necessarily well suited to parallel programming.

- hard to get grain-size right
- scheduling for parallel computation
- nondeterminism
Should concurrent languages be used for parallelism?

Concurrent languages are not necessarily well suited to parallel programming.

► hard to get grain-size right
► scheduling for parallel computation
► nondeterminism
Why concurrency?

- Many applications are **reactive systems** that must cope with non-determinism (e.g., users and the network).
- Concurrency provides a clean abstraction of such interactions by hiding the underlying interleaving of execution.
- Thread abstraction useful for large-grain, heterogeneous parallelism.
Why concurrency?

► Many applications are reactive systems that must cope with non-determinism (e.g., users and the network).
► Concurrency provides a clean abstraction of such interactions by hiding the underlying interleaving of execution.
► Thread abstraction useful for large-grain, heterogeneous parallelism.
Why concurrency?

- Many applications are reactive systems that must cope with non-determinism (e.g., users and the network).
- Concurrency provides a clean abstraction of such interactions by hiding the underlying interleaving of execution.
- Thread abstraction useful for large-grain, heterogeneous parallelism.
Why concurrency?

- Many applications are reactive systems that must cope with non-determinism (e.g., users and the network).
- Concurrency provides a clean abstraction of such interactions by hiding the underlying interleaving of execution.
- Thread abstraction useful for large-grain, heterogeneous parallelism.
Concurrent programming has a reputation of being hard. The problem is that shared-memory is the dominant model in concurrent languages.
Concurrency is hard(?)

Concurrent programming has a reputation of being hard. The problem is that shared-memory is the dominant model in concurrent languages.
Message passing

➤ Well-defined interfaces between independent, sequential, components.
➤ Natural encapsulation of state.
➤ Extends more easily to distributed implementation.
Message passing

- Well-defined interfaces between independent, sequential, components.
- Natural encapsulation of state.
- Extends more easily to distributed implementation.
Message passing

- Well-defined interfaces between independent, sequential, components.
- Natural encapsulation of state.
- Extends more easily to distributed implementation.
Message passing

- Well-defined interfaces between independent, sequential, components.
- Natural encapsulation of state.
- Extends more easily to distributed implementation.
First-class synchronization

Concurrent ML makes synchronization first-class, which enables user-defined communication abstractions.

- RPC in various flavors
- futures and promises
- multicast channels (observer pattern)
First-class synchronization

Concurrent ML makes synchronization first-class, which enables user-defined communication abstractions.

- RPC in various flavors
- futures and promises
- multicast channels (observer pattern)
First-class synchronization

Concurrent ML makes synchronization first-class, which enables user-defined communication abstractions.

- RPC in various flavors
- futures and promises
- multicast channels (observer pattern)
First-class synchronization

Concurrent ML makes synchronization first-class, which enables user-defined communication abstractions.

- RPC in various flavors
- futures and promises
- multicast channels (observer pattern)
Manticore — a heterogeneous parallel language

Collaboration with Matthew Fluet at TTI-C.
Focus on “commodity applications” on “commodity hardware.”

- Strict functional core language — SML w/o refs.
- CML-style concurrency — threads, message-passing, and first-class synchronization.
- Implicitly-threaded nondeterminism
- Implicitly-threaded deterministic parallelism — NESL-style data parallelism, fork-join, data-flow.
- [http://manticore.cs.uchicago.edu](http://manticore.cs.uchicago.edu)
Manticore — a heterogeneous parallel language

Collaboration with Matthew Fluet at TTI-C.
Focus on “commodity applications” on “commodity hardware.”

- Strict functional core language — SML w/o refs.
- CML-style concurrency — threads, message-passing, and first-class synchronization.
- Implicitly-threaded nondeterminism
- Implicitly-threaded deterministic parallelism — NESL-style data parallelism, fork-join, data-flow.

http://manticore.cs.uchicago.edu
Manticore — a heterogeneous parallel language

Collaboration with Matthew Fluet at TTI-C.
Focus on “commodity applications” on “commodity hardware.”

- Strict functional core language — SML w/o refs.
- CML-style concurrency — threads, message-passing, and first-class synchronization.
- Implicitly-threaded nondeterminism
- Implicitly-threaded deterministic parallelism — NESL-style data parallelism, fork-join, data-flow.

http://manticore.cs.uchicago.edu
Manticore — a heterogeneous parallel language

Collaboration with Matthew Fluet at TTI-C. Focus on “commodity applications” on “commodity hardware.”

- Strict functional core language — SML w/o refs.
- CML-style concurrency — threads, message-passing, and first-class synchronization.
- Implicitly-threaded nondeterminism
- Implicitly-threaded deterministic parallelism — NESL-style data parallelism, fork-join, data-flow.

http://manticore.cs.uchicago.edu
Manticore — a heterogeneous parallel language

Collaboration with Matthew Fluet at TTI-C. Focus on “commodity applications” on “commodity hardware.”

- Strict functional core language — SML w/o refs.
- CML-style concurrency — threads, message-passing, and first-class synchronization.
- Implicitly-threaded nondeterminism
- Implicitly-threaded deterministic parallelism — NESL-style data parallelism, fork-join, data-flow.

http://manticore.cs.uchicago.edu
Manticore — a heterogeneous parallel language

Collaboration with Matthew Fluet at TTI-C.
Focus on “commodity applications” on “commodity hardware.”

- Strict functional core language — SML w/o refs.
- CML-style concurrency — threads, message-passing, and first-class synchronization.
- Implicitly-threaded nondeterminism
- Implicitly-threaded deterministic parallelism — NESL-style data parallelism, fork-join, data-flow.

http://manticore.cs.uchicago.edu
Conclusion

Should parallel languages be concurrent?

- Maybe — HPC applications probably do not benefit from explicit concurrency.

Should concurrent languages be parallel?

- Yes — parallel implementation of concurrency constructs
- Yes — support for deterministic parallel computation in threads
Conclusion

Should parallel languages be concurrent?
- Maybe — HPC applications probably do not benefit from explicit concurrency.

Should concurrent languages be parallel?
- Yes — parallel implementation of concurrency constructs
- Yes — support for deterministic parallel computation in threads
Conclusion

Should parallel languages be concurrent?
▶ Maybe — HPC applications probably do not benefit from explicit concurrency.

Should concurrent languages be parallel?
▶ Yes — parallel implementation of concurrency constructs
▶ Yes — support for deterministic parallel computation in threads
Conclusion

Should parallel languages be concurrent?
▶ Maybe — HPC applications probably do not benefit from explicit concurrency.

Should concurrent languages be parallel?
▶ Yes — parallel implementation of concurrency constructs
▶ Yes — support for deterministic parallel computation in threads
Conclusion

Should parallel languages be concurrent?
  ▶ Maybe — HPC applications probably do not benefit from explicit concurrency.

Should concurrent languages be parallel?
  ▶ Yes — parallel implementation of concurrency constructs
  ▶ Yes — support for deterministic parallel computation in threads