ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Speculative Synchronization: Applying Thread-Level Speculation to Explicitly Parallel Applications

José F. Martínez and Josep Torrellas
Dept. of Computer Science, University of Illinois, 2002

Presented by: David Itten

Agenda

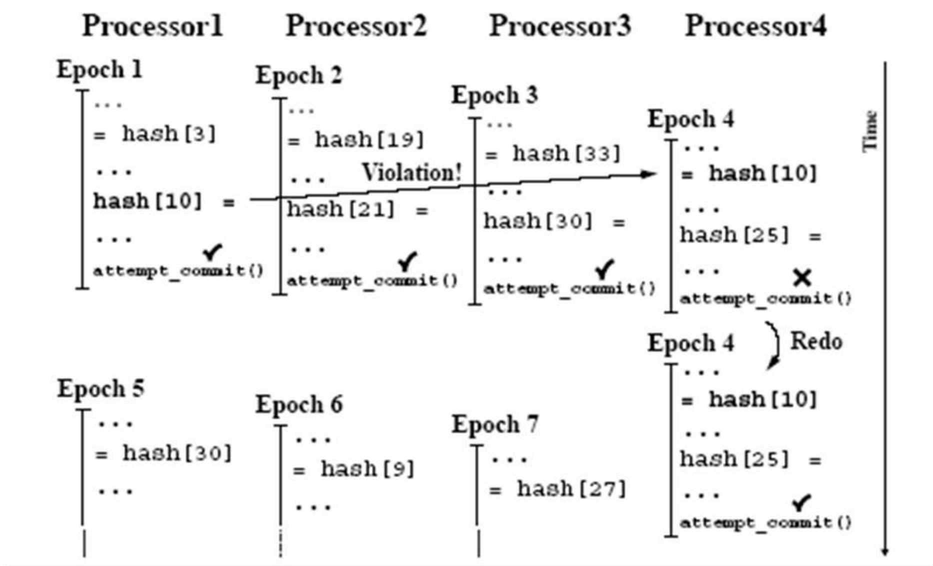
- Thread-Level Speculation
- Introduction to Speculative Synchronisation
- Hardware needed
- Using Speculative Synchronisation
- Evaluation

Thread-Level Speculation TLS

- One safe thread
- Extracts speculative threads from serial code
- Threads go into potentially unsafe program sections
- Epoch numbers, lowest epoch number is safe thread
- Unsafe memory state in buffer

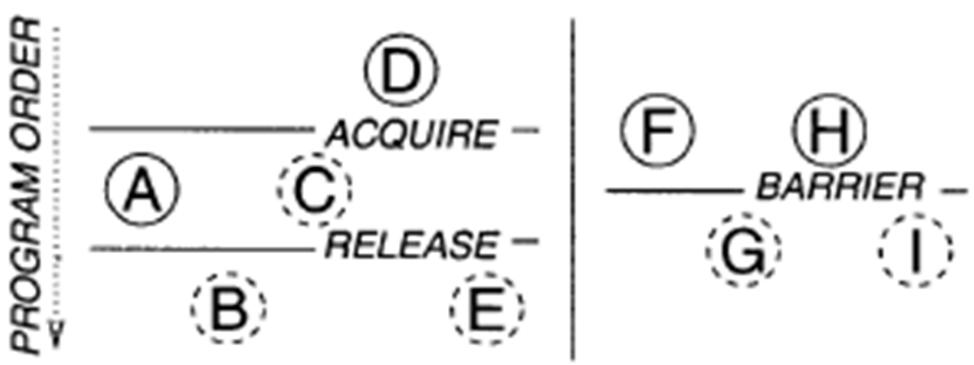
TLS - Example

```
while(continue_condition) {
    ...
    x = hash[index1];
    ...
    hash[index2] = y;
    ...
}
```

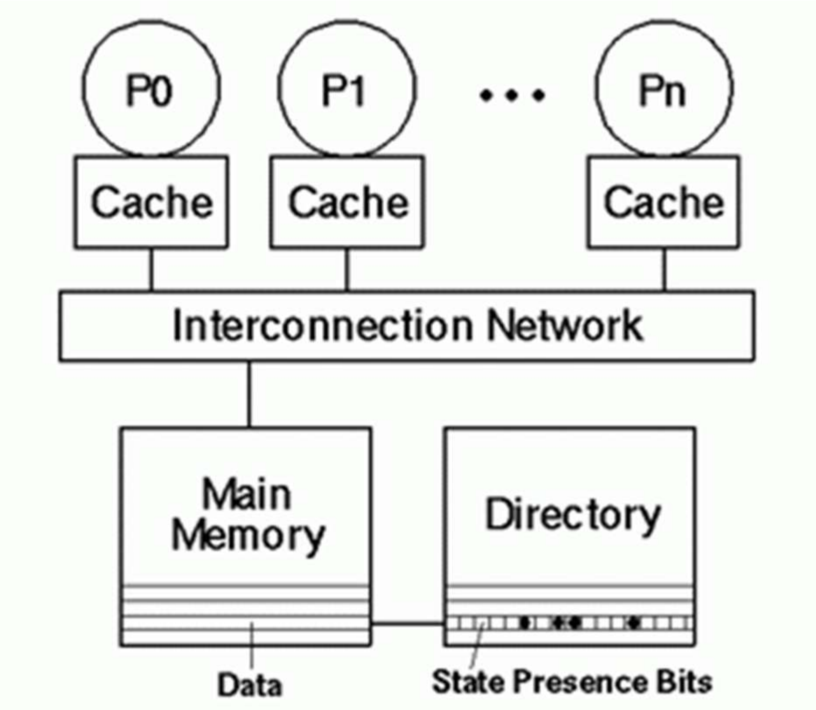


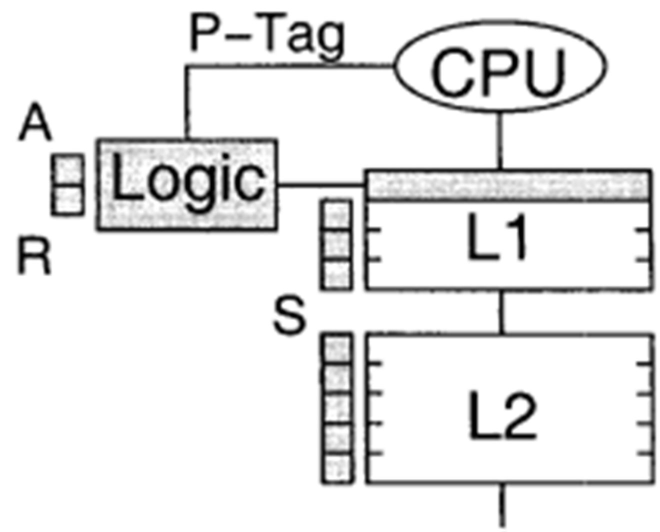
Speculative Synchronization

- Execute code past active barriers, busy locks and unset flags
- Extra concurrency in presence of conservatively placed synchronisation
- Apply TLS concept to explicitly parallel applications
  - No ordering of speculative threads

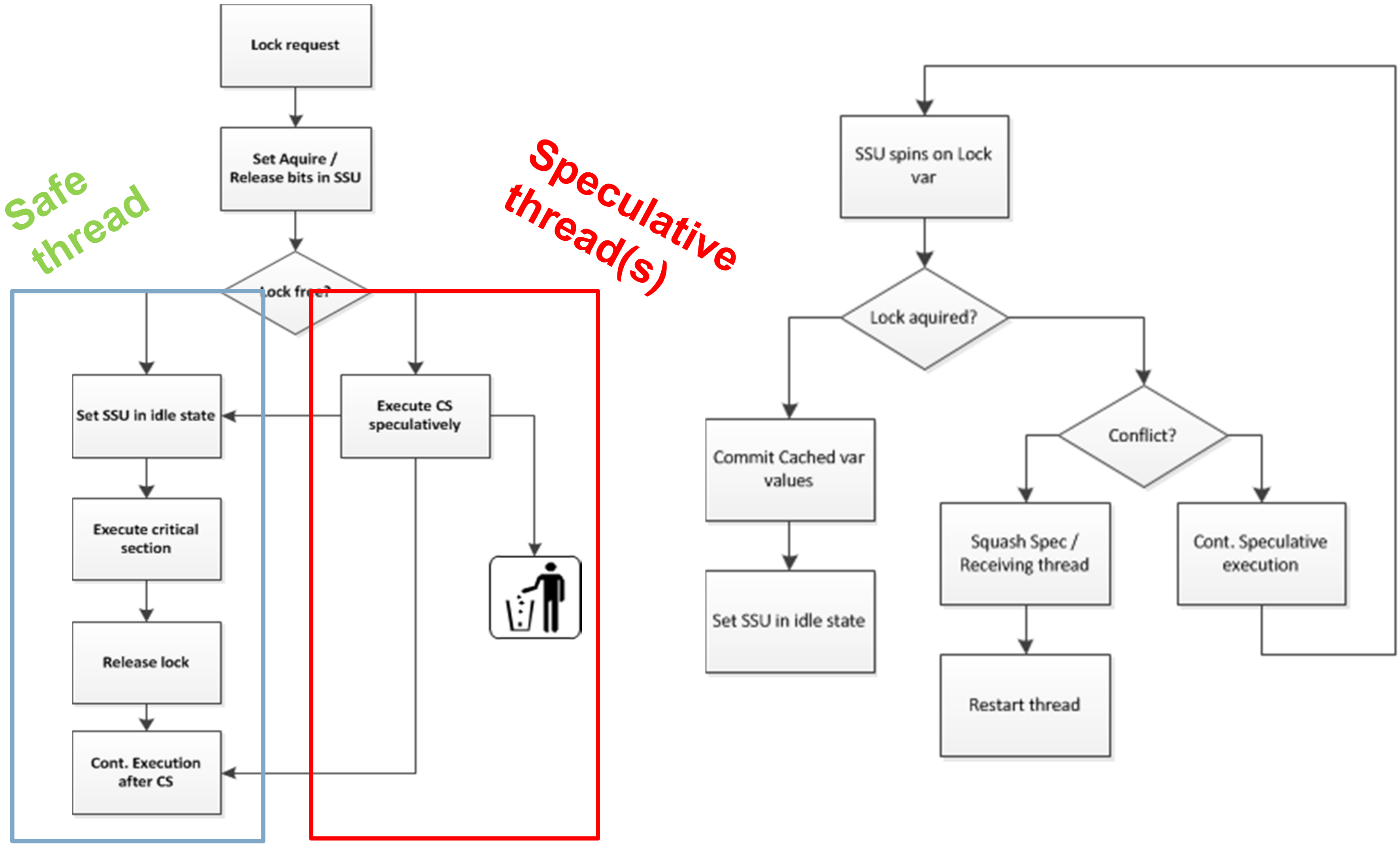


Speculative Synchronization Unit (SSU)





Speculative Synchronization - Process



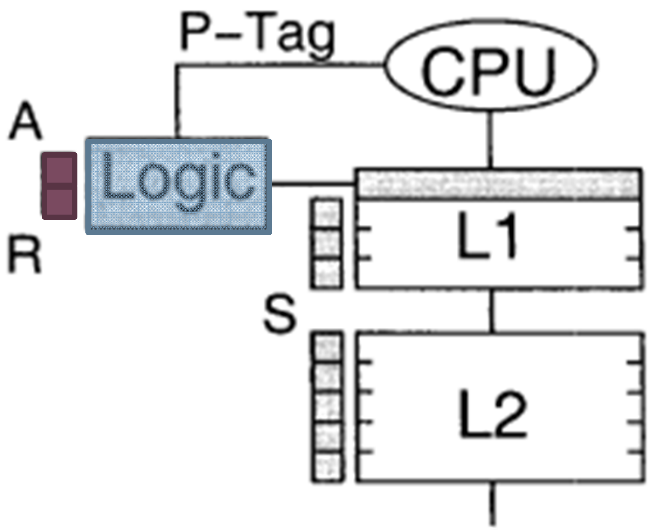
Speculative execution

- Checkpoint the execution
  - Backup register states
- Processor hints for all memory accesses
- Set speculative bit in cache
  - Write back cache content to memory if dirty in all caches

Access conflicts

- Speculative thread receives a message for cache lines marked speculative
  - Squash receiver of message
  - Safe thread never squashed!
- Squash procedure:
  - Invalidate all dirty cache lines with speculative bit
  - Clear all speculative bits
  - Restore check pointed register state
  - Restart thread

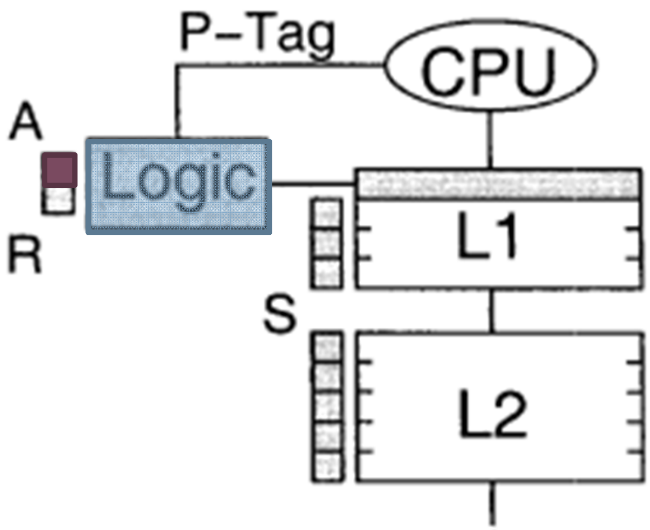
SSU states



- Aquire / Release flags set
- SSU Active

➔ Thread is executing speculatively CS

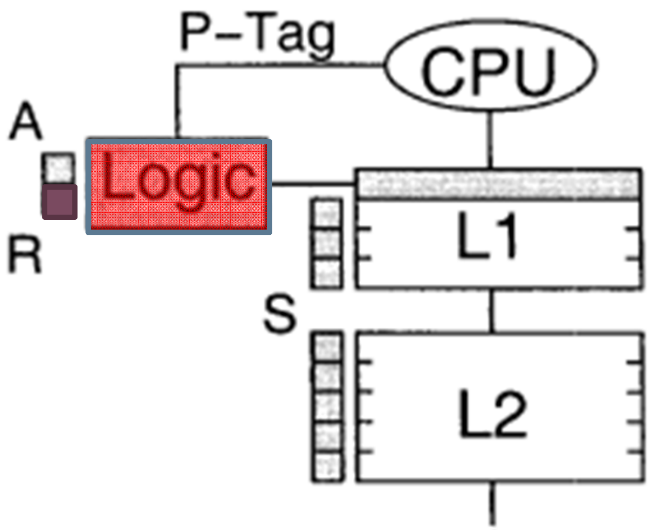
SSU states



- Acquire flag set
- SSU Active

➔ Thread already left the CS, executes code after the CS and wants to commit its values

SSU states



- Release flags set
- SSU Inactive

➔ This is the safe thread, executing the CS

Speculative Flags

- Release bit kept clear (Release while speculative)
- Speculatively execute code after the flag
- Barriers can be built using flags and locks

Potential problems

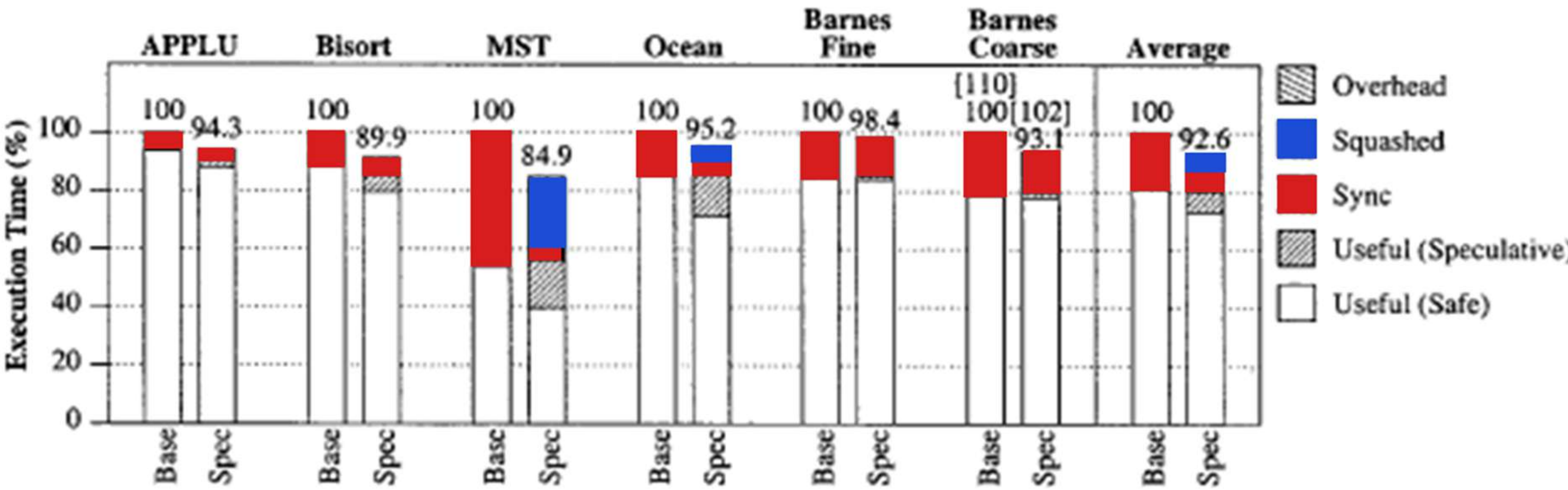
- No free cache lines available: Stall execution
- Second lock:
  - Handle second lock like a normal variable
  - Wait for lock until thread becomes safe or lock is released
- Exceptions: Speculative threads are always squashed
- Irreversible actions (e.g. I/O access)

Evaluation

- NUMA processor architecture
- 16 or 64 nodes
- L1 and L2 write back caches
- 5 concurrent applications used for test
  - Hand-parallelized
  - Parallelizing compiler
  - Annotated applications which are transformed to parallel code

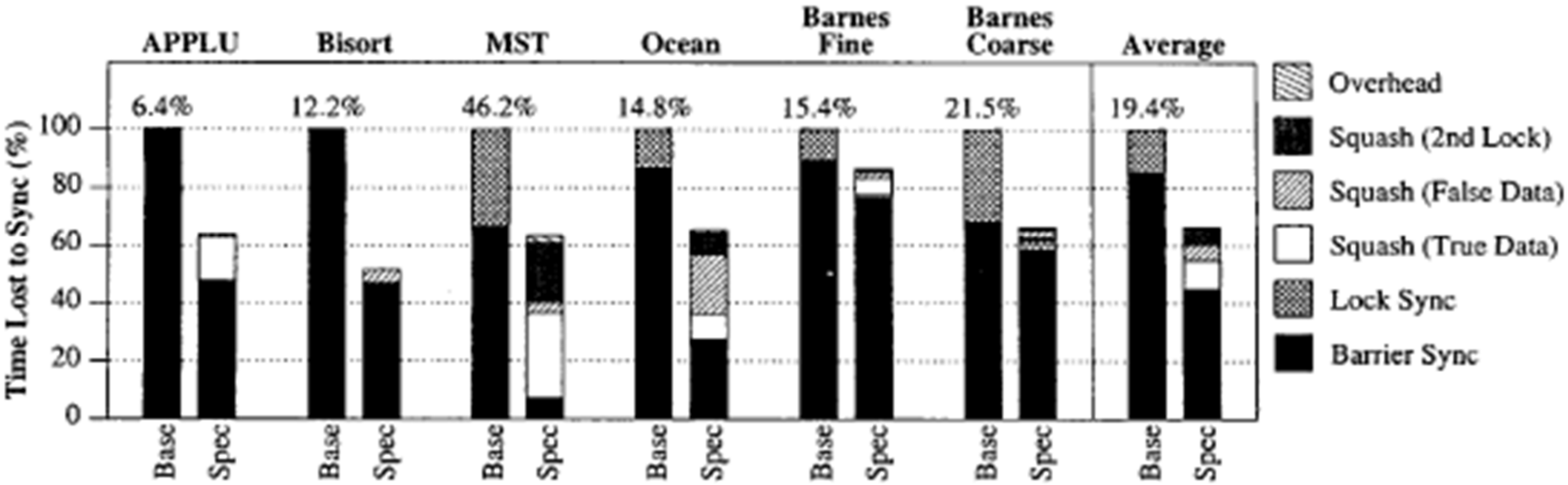
Evaluation

Application	*Description*	*Parallelization*	*Data Size*	*Processors*	*Barriers/Locks*
APPLU	LU factorization	Compiler	Reference	16	Yes/No
Bisort	Bitonic sort	Annotations	16K nodes	16	Yes/No
MST	Minimum spanning tree	Annotations	512 nodes	16	Yes/Yes
Ocean	Ocean simulation	Hand	258x258	64	Yes/Yes
Barnes–Fine	N–body problem	Hand	16K particles	64	Yes/Yes(2048)
Barnes–Coarse					Yes/Yes(512)



Evaluation

Application	*Description*	*Parallelization*	*Data Size*	*Processors*	*Barriers/Locks*
APPLU	LU factorization	Compiler	Reference	16	Yes/No
Bisort	Bitonic sort	Annotations	16K nodes	16	Yes/No
MST	Minimum spanning tree	Annotations	512 nodes	16	Yes/Yes
Ocean	Ocean simulation	Hand	258x258	64	Yes/Yes
Barnes–Fine	N–body problem	Hand	16K particles	64	Yes/Yes(2048)
Barnes–Coarse					Yes/Yes(512)



Conclusion

- Faster parallel execution for free
- Requires a hardware modification
- Barriers are still a problem

Questions