Abstracting the Hardware / Software Boundary through a Standard System Support Layer and Architecture

Erik Anderson

4 May 2007

Agenda

- Publications and proposal review.
- Problem background.
- Abstracting HW/SW boundary.
- Analysis, comparing HW and SW.
- Results
- Conclusions

Special Thanks to

- David Andrews
- Perry Alexander
- Douglass Niehaus
- Ron Sass
- Yang Zhang
- Jason Agron
- Fabrice Baijot
- Ed Komp
- Andy Schmidt
- Jim Stevens
- Wesley Peck
- Seth Warn

Academic Background

- Bachelor of Science in Computer Science, University of Kentucky, 1993 - 1997, Magna Cum Laude
- Doctoral Candidate in Electrical Engineering, Kansas University, 2003 - 2007

Publications

- “Enabling a Uniform Programming Model Across the Software/Hardware Boundary,” FCCM 2006
- “Supporting High Level Language Semantics within Hardware Resident Threads,” submitted to FPL 2007
- “Memory Hierarchy for MCSoPC Multithreaded Systems,” ERSA 2007
- “Achieving Programming Model Abstractions for Reconfigurable Computing,” Transactions on VLSI, to appear in 2007
- “Hthreads: A Computational Model for Reconfigurable Devices,” FPL 2006
- “The Case for High Level Programming Models for Reconfigurable Computing,” ERSA 2006
- “Run-time Services for Hybrid CPU/FPGA Systems on Chip,” RTSS 2006

Proposal Review

Proposed

- Augment the HWTI
- Extend support for key subset of Hthread API.
- Semantic and implementation differences.
- Hthread test suite.
- Application suite.

Completed

- Augmented HWTI with:
  - User interface and protocol.
  - Globally distributed local memory.
  - Function call stack.
- Extended support for key subset of Hthread API.
  - Remote procedural calls.
- Chapter 5 in dissertation
  - Context similarities.
- Hthread test suite.
  - Abstractions held.
- Application suite.
  - Framework for HLL to HDL.

History of Reconfigurable Computing

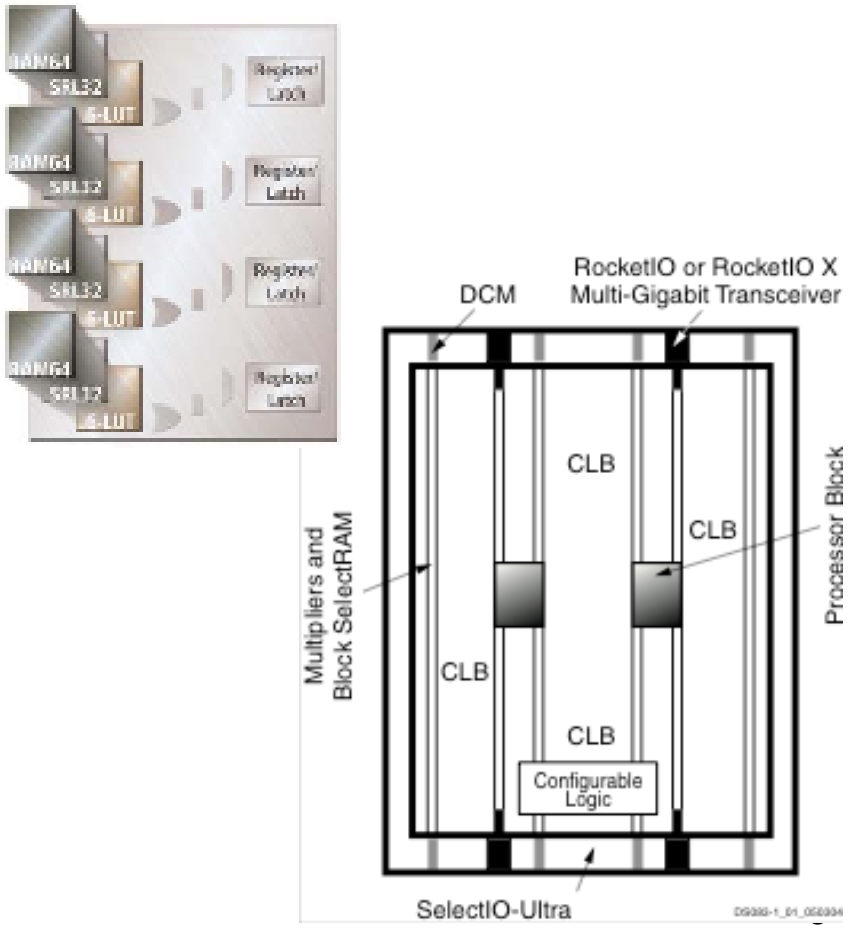
- 1959: Gerald Estrin's Fixed plus Variable Architecture.
- 1984: Xilinx is founded.
- 1993: Athana proposed PRISM-I
- 2006: First 65nm FPGA released.

Conference	Papers
FCCM 2006	25
FPL 2006	30
FPGA 2006	22

IEEE Annals of History of Computing, Oct - Dec 2002, page 5

Reconfigurable Computing Technology

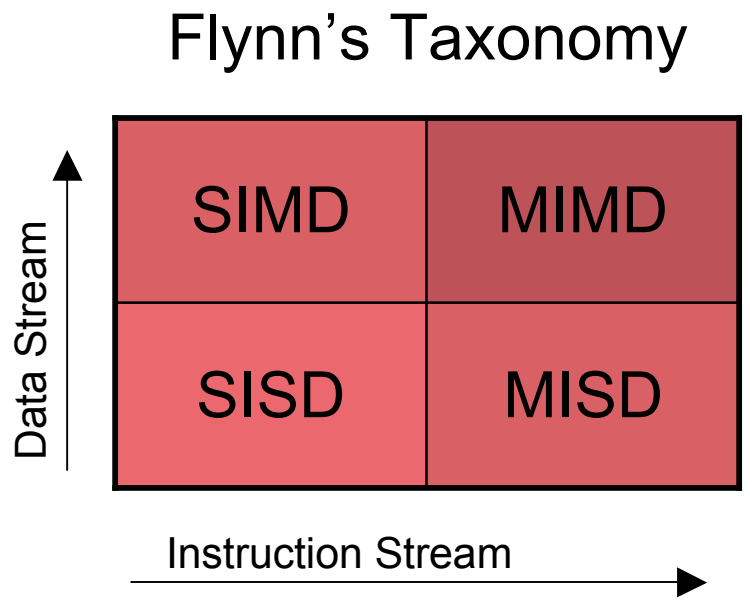
- Post-fabrication circuit design.
- Embedded cores, memory, and multipliers.



From xilinx.com

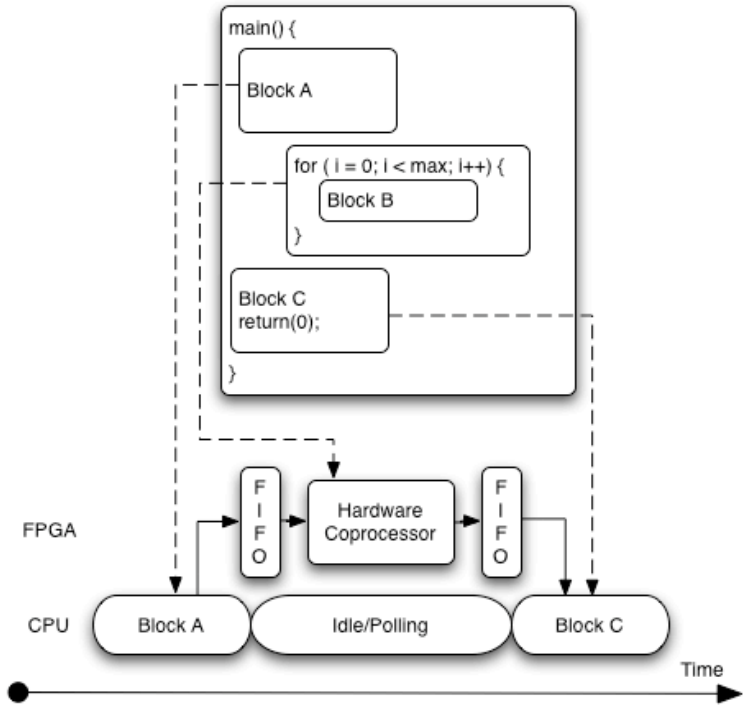
“Blessing and a Curse”

- FPGA’s can take on any computational model post-fabrication.
- But which one to use?



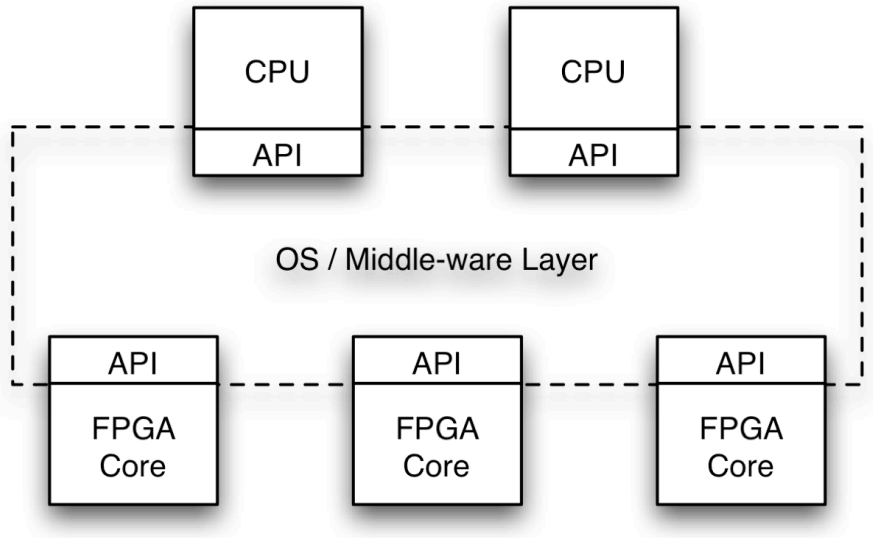
Hardware Acceleration Model

- SISD or SIMD.
- Advantages:
  - Can be successful.
  - C to HDL tools.
- Disadvantages:
  - Custom interfaces between HW and SW.
    - Write once, run once.
  - Costly design-space exploration.
  - Does not use today's MIMD programming models.



Abstract Interfaces

- Parallel programming models to abstract CPU / FPGA interface.
- CPU and FPGA both target an equivalent abstract interface.
- OS / Middleware layer provides communication and synchronization mechanism.

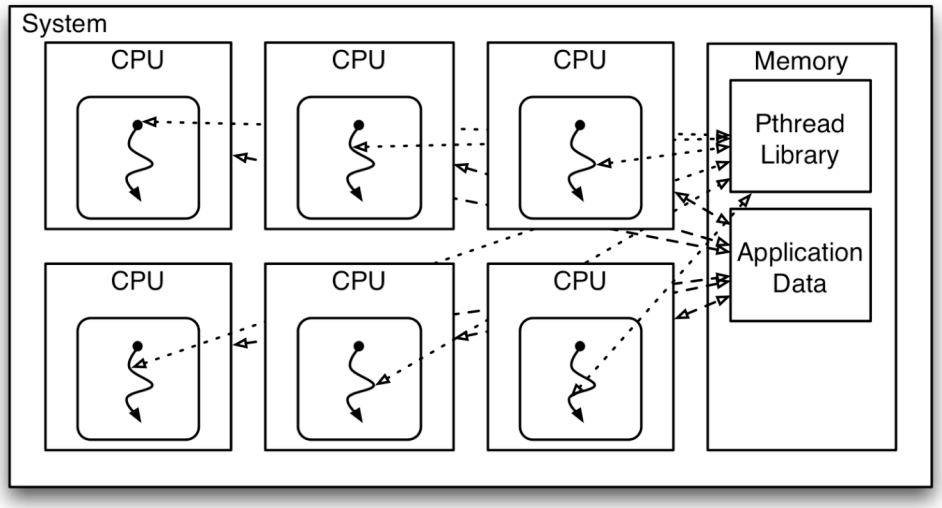


Thesis Statement

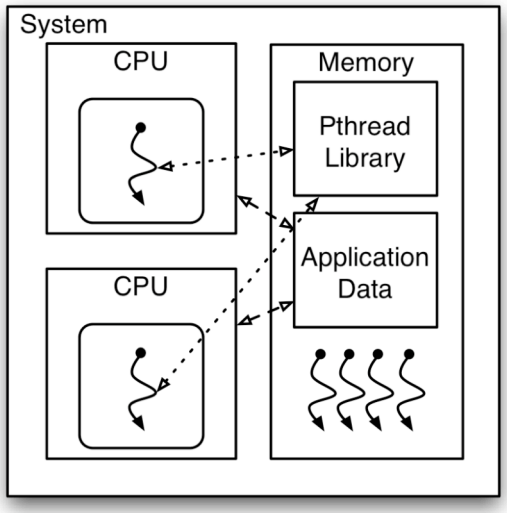
Programming model and high level language constructs can be used to abstract the existing hardware/software boundary that currently exists between CPU and FPGA components.

Extending the Shared Memory Multi-Threaded Model to Hardware

- Pthreads programming



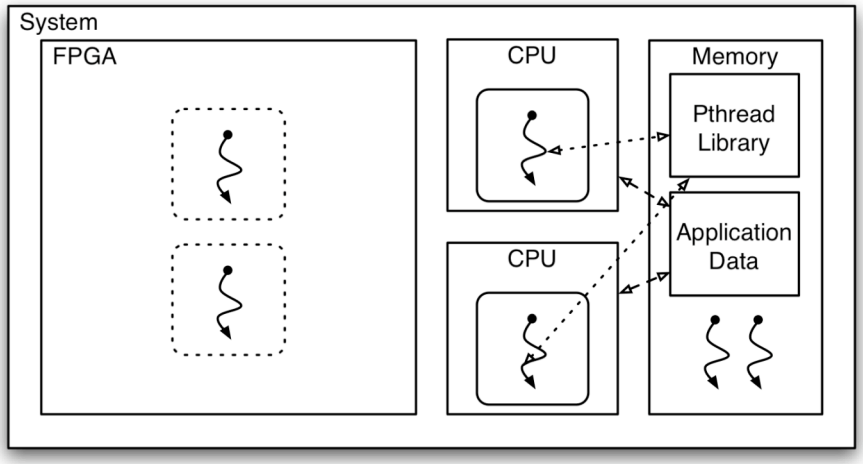
Conceptual



Reality

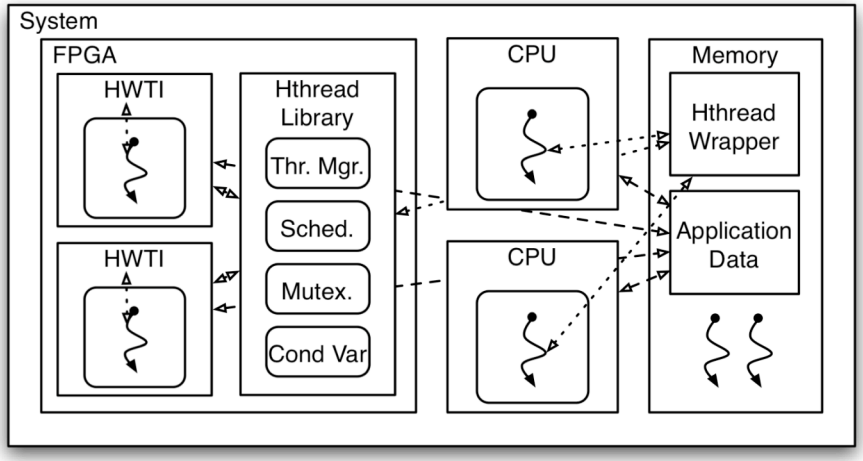
Extending the Shared Memory Multi-Threaded Model to Hardware

- Key Challenges
  - HW access to API library.
  - HW access to application data.
  - Eliminate custom interface to HW.



Extending the Shared Memory Multi-Threaded Model to Hardware

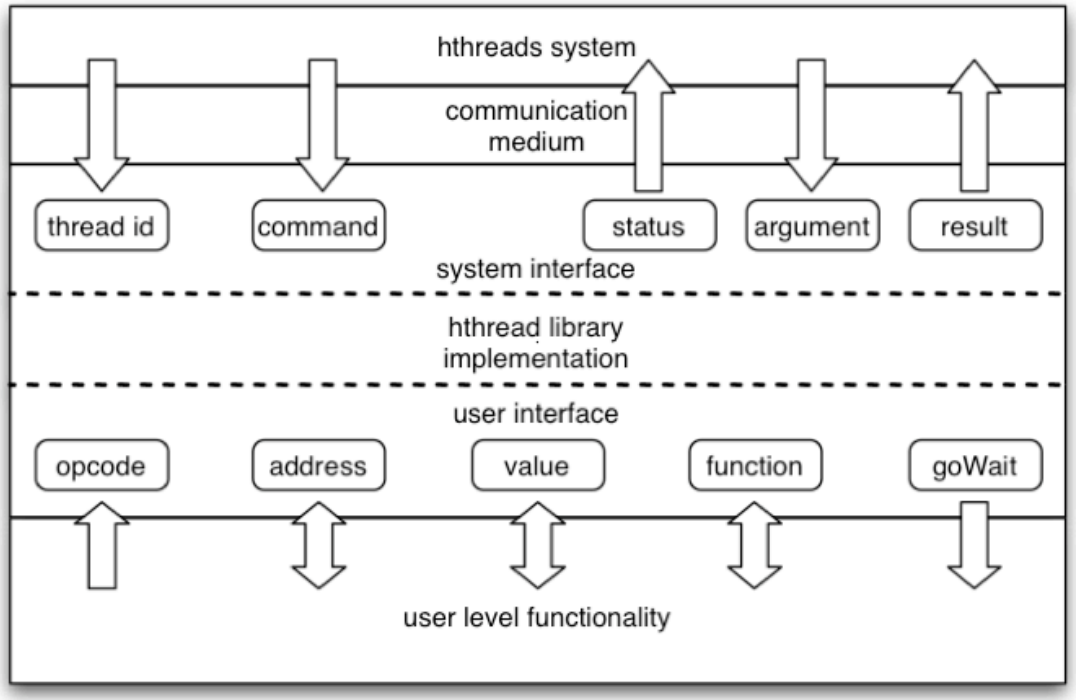
- Hthread's Solutions
  - Access to the same communication medium.
  - Equal or equivalent synchronization services migrated to HW.
  - Standard system support layer.



Hybridthreads System

Hardware Thread Interface

- HWTI provides a standard register set for communication and synchronization services.

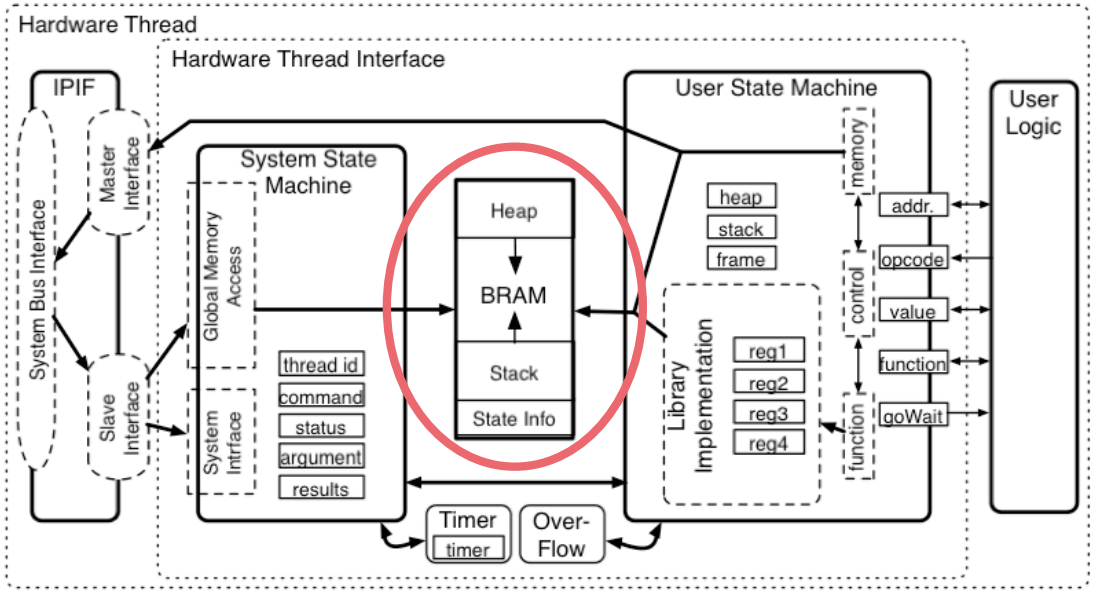


Creating a *Meaningful* Abstraction

- Communication and synchronization are solved.
- Problems persist:
  - "scratchpad" memory:
    - How to instantiate?
    - How to maintain the shared memory model?
  - System versus user function calls?
  - Creating threads from hardware?

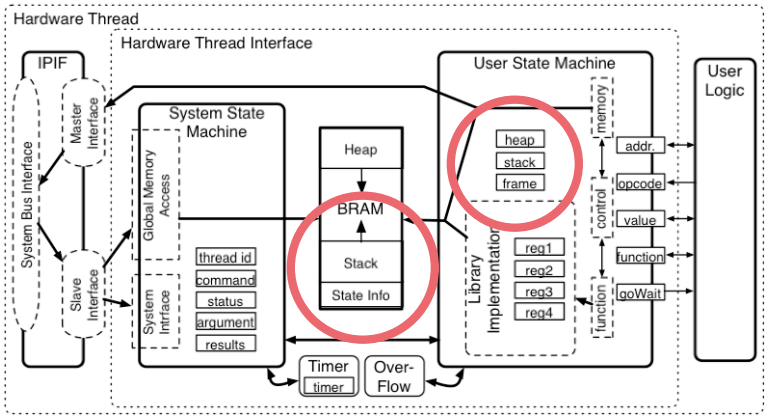
Globally Distributed Local Memory

- Dual ported BRAM.
- “Globally Distributed” = All threads have access.
- “Local” = User logic access is through LOAD and STORE protocols.



Function Call Stack

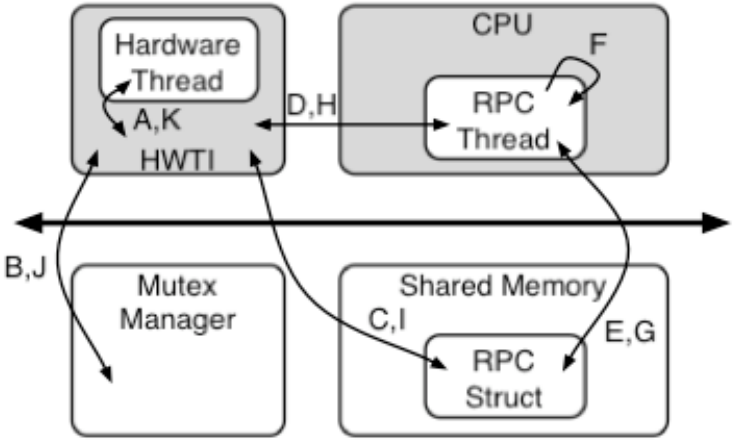
- Abstract access to local memory.
- Consistent function call model.
  - Recursion.
- Works analogously to software based stack.
  - Only difference, user logic pushes “return state” value instead of “return instruction.”



Operation	Clock Cycles
PUSH	1
POP	5
DECLARE	1
READ	3
WRITE	1
ADDRESSOF	1
CALL	3
RETURN	7

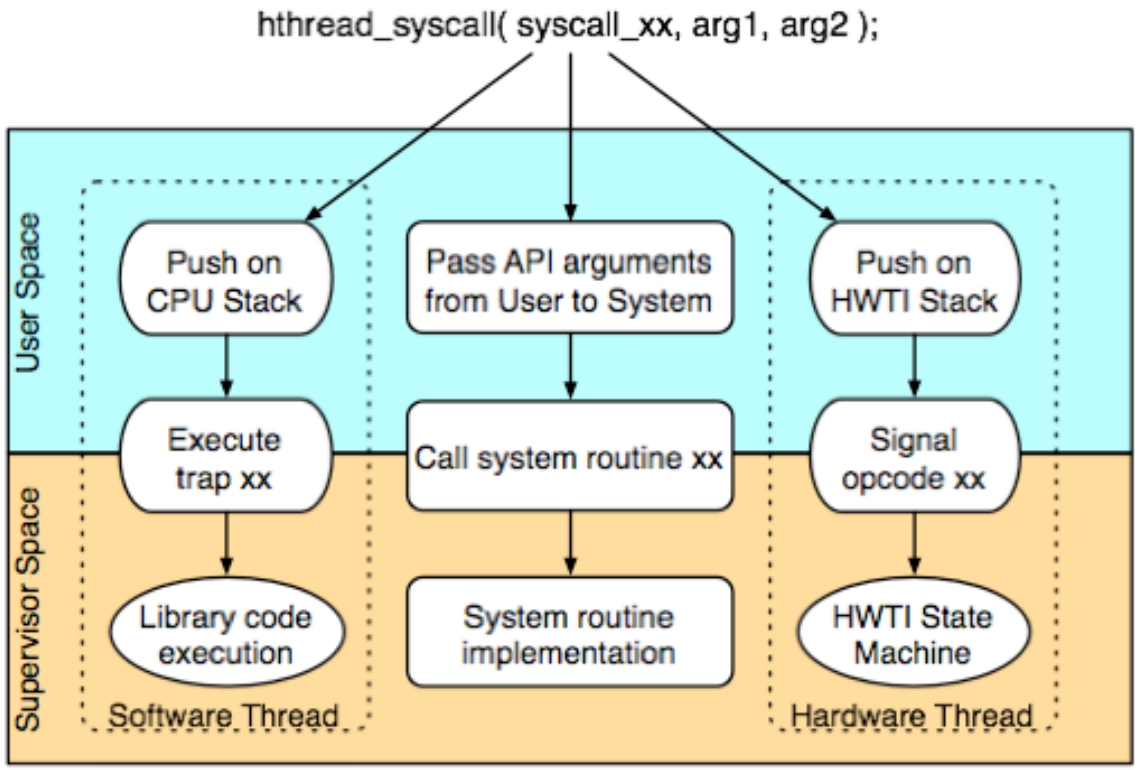
Remote Procedural Calls

- Some functions too expensive to implement in HWTI.
- Utilize existing synchronization primitives to callout to a special software system thread to perform function.



A: UL calls a function not supported by HWTI.
B: HWTI obtains lock on RPC.
C: HWTI writes opcode and arguments to RPC struct.
D: HWTI signals RPC Thread and waits.
E: RPC thread reads the opcode and arguments.
F: RPC performs function on behalf of HWT.
G: RPC writes result to RPC struct.
H: RPC signals HWT operation is complete.
I: HWTI reads results from RPC struct.
J: HWTI releases lock on RPC
K: HWTI returns results to UL.

Hardware / Software Duality



Hthread System Call Implementation Differences

- HW has dedicated resources allocated at synthesis time.
  - HW explicitly blocks.
- SW has shared resources allocated at runtime.
  - SW context switches.

Hthread Size and Performance Comparison

- Size
  - Definition
  - hthread_create / hthread_join
  - hthread_yield
- Performance
  - Definition
  - HW outperforms SW
    - Create/join notable exception
  - Hardware's bus transactions

Demonstrating an Abstract Interface

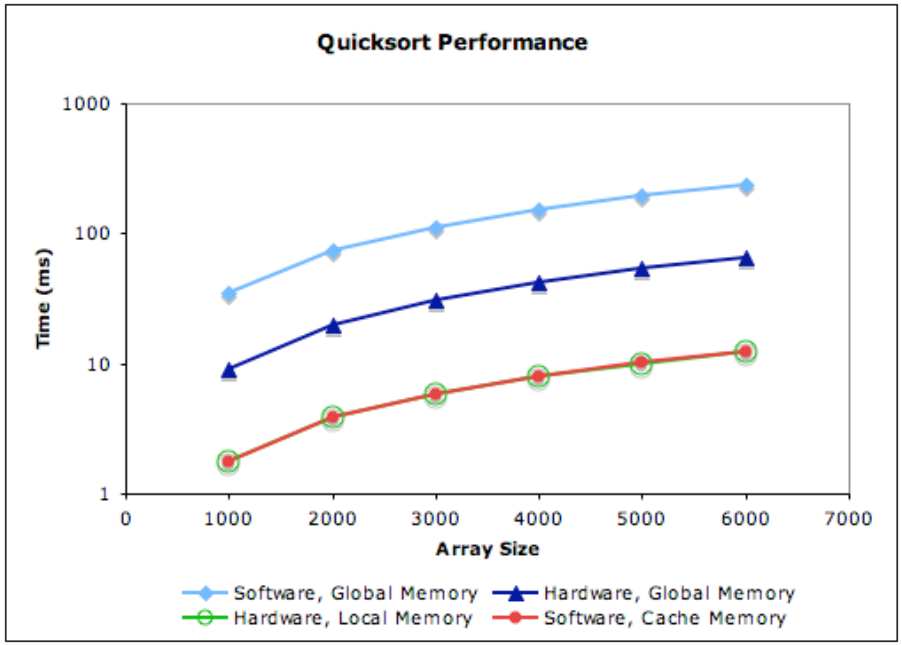
- POSIX Test-suite adapted for Hthreads.
- Conformance tests
  - Version for SW, HW, and mixed.
- Stress tests
  - Version for SW, HW, and mixed.
- Abstractions held across SW/HW.

Demonstrating HLL Constructs

Algorithm	**Demonstrates**
Quicksort	Recursion, local variables, access to shared memory.
Factorial	Recursion, local variables.
Huffman	Sharing data between HW and SW
Haar DWT	Local array access, access to shared memory.
IDEA	Task level parallelism, local variables.

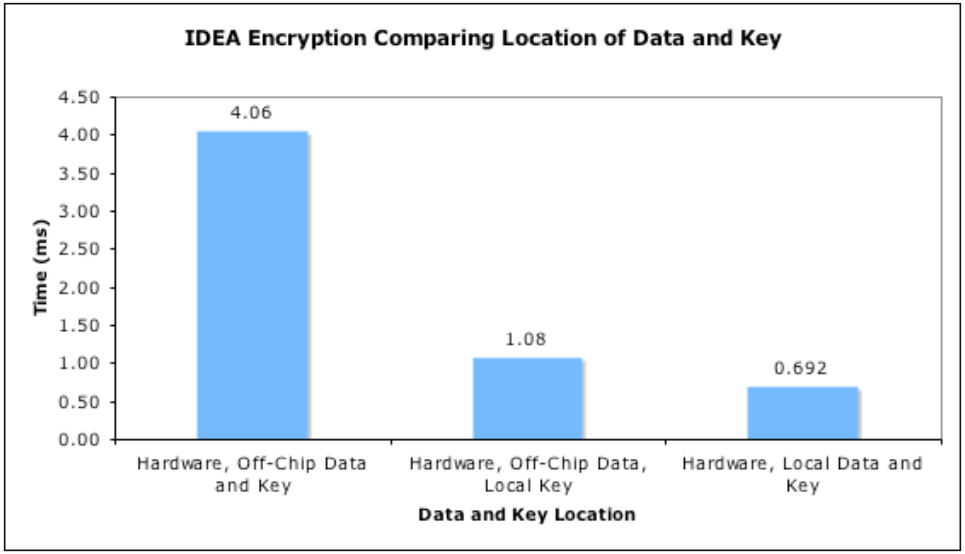
Function Call Stacks and Recursion

- Quicksort
  - Recursive
  - *O(nlogn)* performance



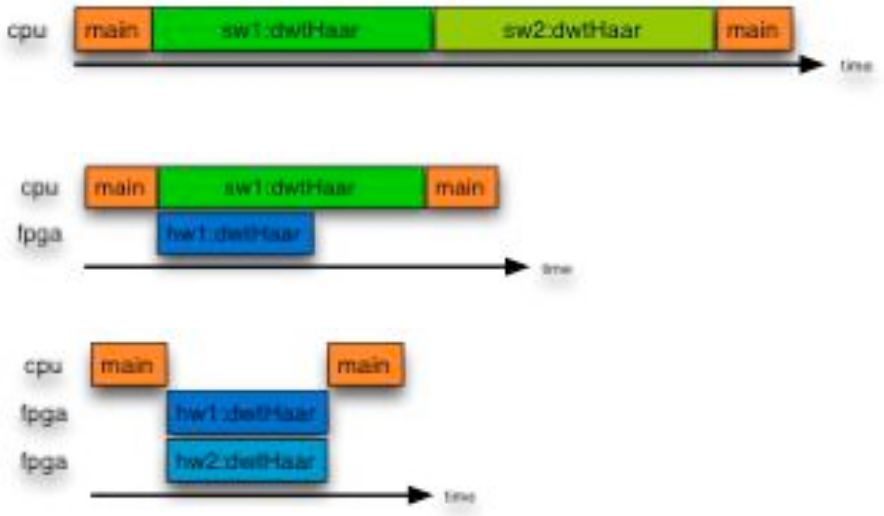
Memory Latency

- IDEA encryption
  - Key and data location comparison.



Task Level Parallelism

- Haar DWT
  - Software's pseudo-concurrency.
  - Hardware's true concurrency.
- Performance
  - 2 SW = 31.1ms
  - 1HW/SW = 16.5ms
  - 2 HW = 16.6ms



Future Work

- Memory latency for hardware threads.
- Leveraging reconfigurable computing.
- High level language to hardware descriptive language translation.

Conclusions

- Parallel programming models may be used to abstract CPU/FPGA boundary.
  - Threads communicate and synchronize with other threads without regard to location.
- Abstract virtual machine can be implemented in either HW or SW.
  - Created a framework for HLL to HDL.

Questions?

Supplemental Material

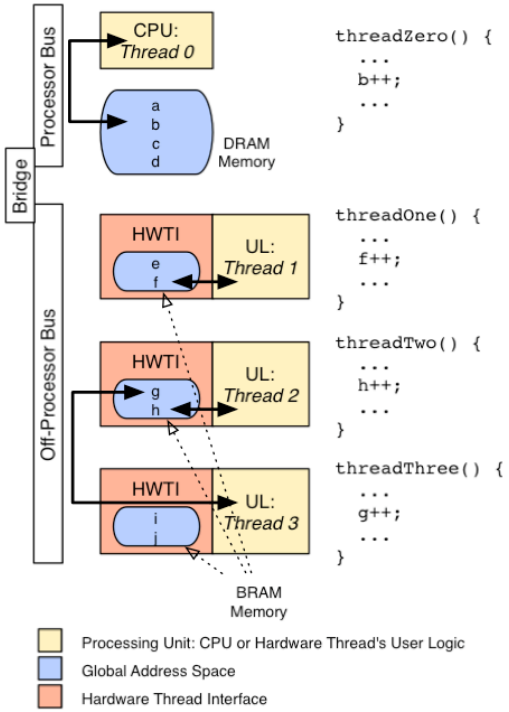
Function Call Stack Example

```
void * threadFunction(int * argument) {
  int a;
  int b;
  int c;
  foo( &a, &b );
  //return state = x0102
  ...
}

void foo( int *a, int *b ) {
  int d;
  int e;
  //Stack shown here
  ...
}
```

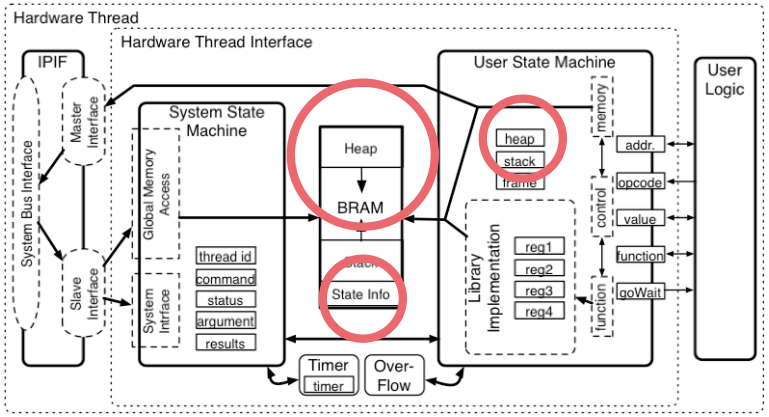
	Address	Value	Meaning
SP →	x0088		next declared variable or parameter push
	x0084	E	declared variable E
FP →	x0080	D	declared variable D
	x007A	x0000 0102	user logic's return state
	x0078	x6300 0060	frame pointer restore value
	x0074	2	number of parameters passed to foo()
	x0070	x6300 0060	first parameter passed to foo(), address of A
	x006A	x6300 0064	second parameter passed to foo(), address of B
	x0068	C	declared variable C
	x0064	B	declared variable B
	x0060	A	declared variable A
	x005A	x0000 0000	user logic's return state
	x0058	x0000 0000	frame pointer restore value
	x0054	1	number of parameters passed to the thread
	x0050	x0000 2340	argument of thread

Globally Distributed Local Memory



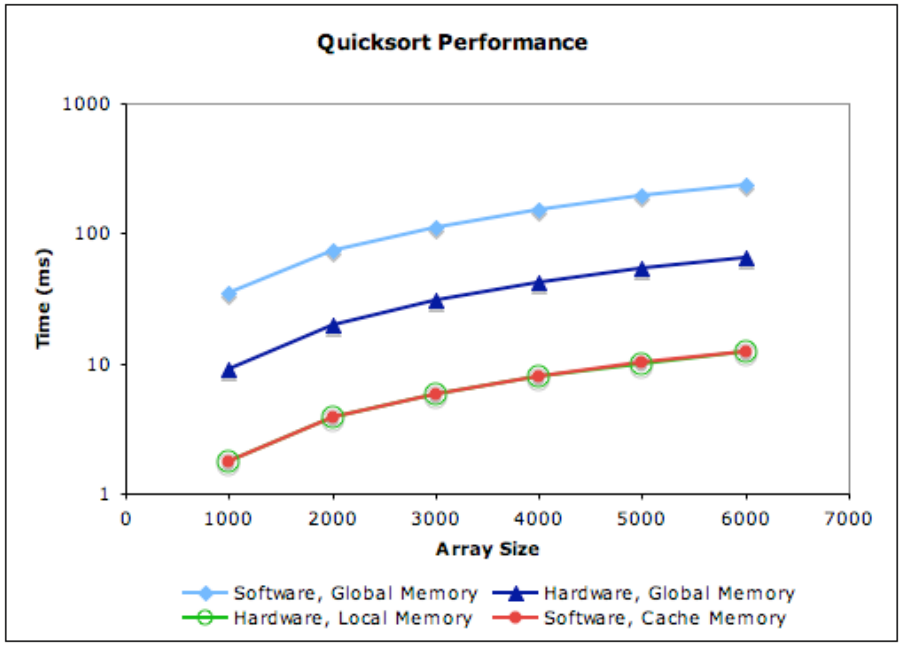
Dynamic Memory Allocation

- Pre-allocated Heap.
- Light version of malloc, calloc, and free.



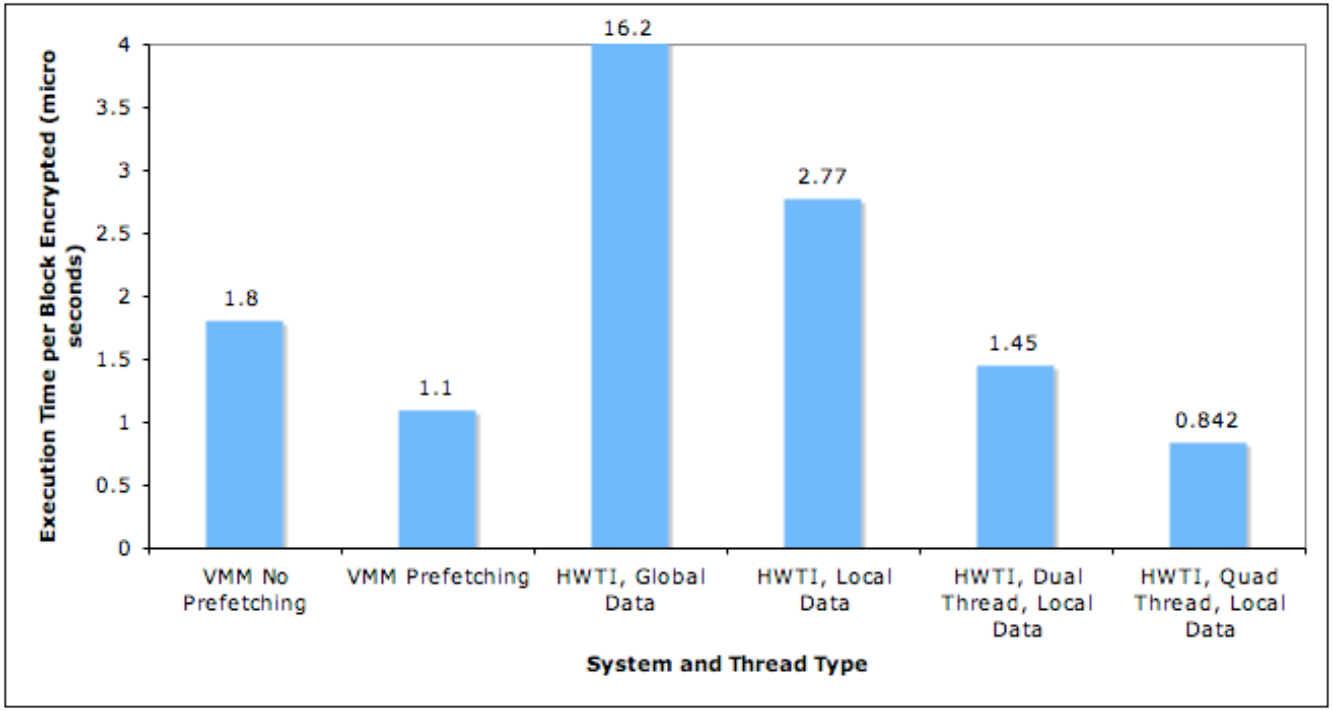
Demonstration HLL Constructs: Quicksort

- HWTI maintains *O(nlogn)* behavior.
- Cache-like performance.



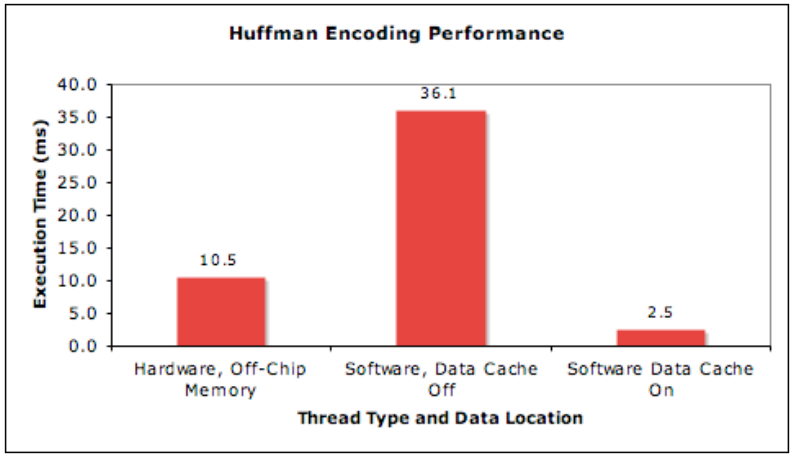
Demonstration HLL Constructs: IDEA

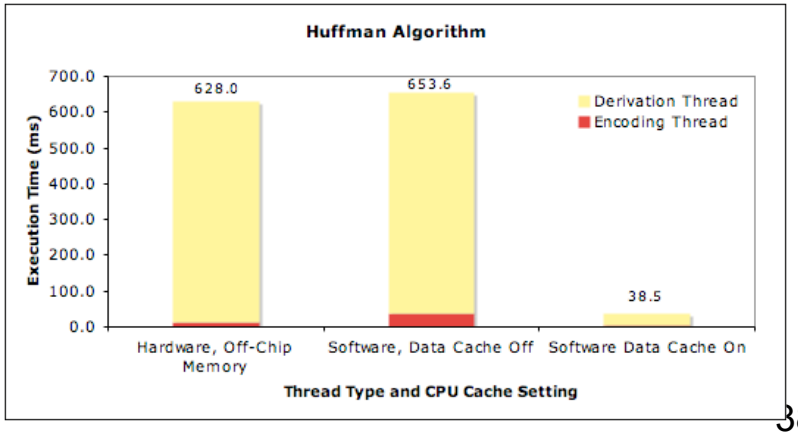
- Benefits of task level parallelism.
- Comparison with Vuletic's hardware threads.



Demonstration HLL Applicability: Huffman

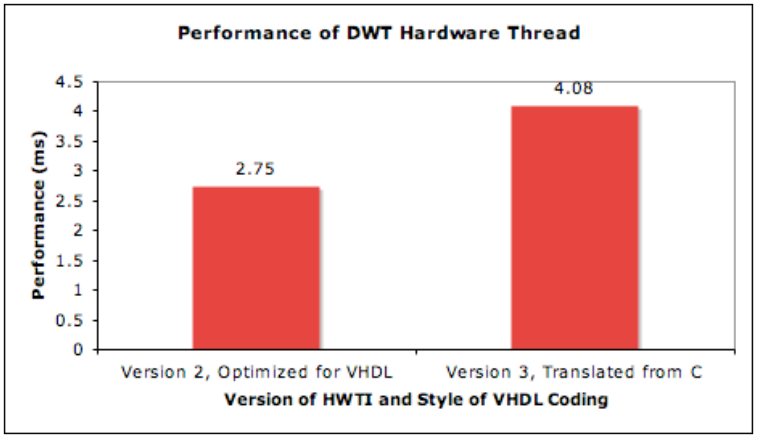
- Abstract data passing between SW and HW threads.
- Data cache on CPU.

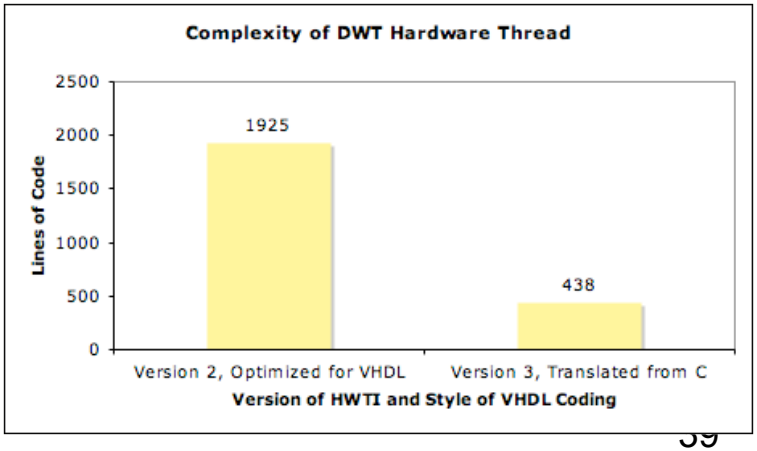




Demonstration HLL Applicability: Haar DWT

- Abstract interface vs meaningful abstract interface.
- Performance.
- Complexity.



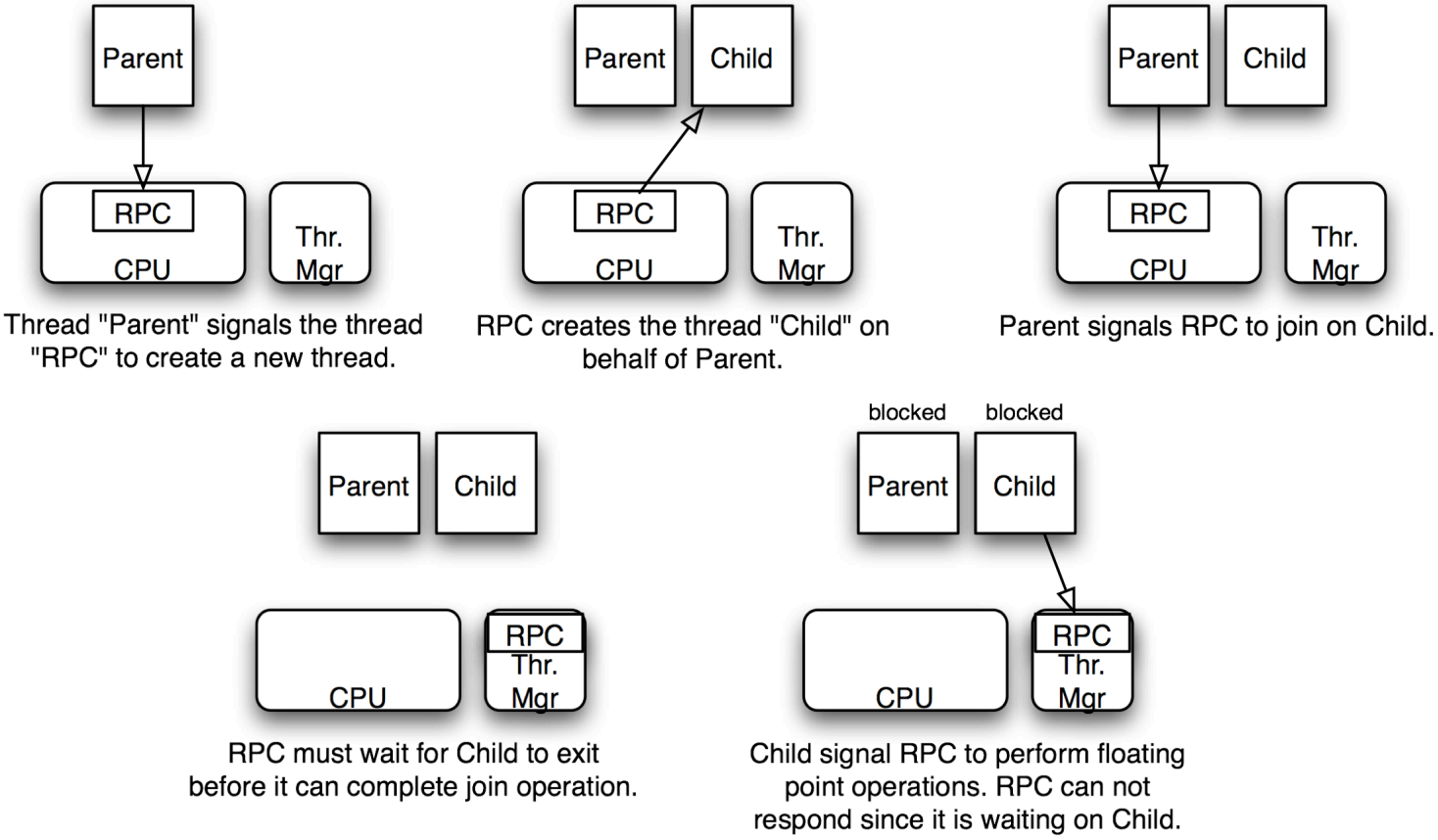


Globally Distributed Local Memory

- “Cache like” performance.
- Maintains shared memory model.
- User access without bus transactions.

Operation	Global	Local	HWTI
Load	51	3	19
Store	28	1	19

Join Danger


Parent
RPC
CPU
Thr.
Mgr
Thread "Parent" signals the thread "RPC" to create a new thread.
Parent
Child
RPC
CPU
Thr.
Mgr
RPC creates the thread "Child" on behalf of Parent.
Parent
Child
RPC
CPU
Thr.
Mgr
Parent signals RPC to join on Child.
Parent
Child
CPU
RPC
Thr.
Mgr
RPC must wait for Child to exit before it can complete join operation.
blocked
blocked
Parent
Child
CPU
RPC
Thr.
Mgr
Child signal RPC to perform floating point operations. RPC can not respond since it is waiting on Child.

Remote Procedural Calls

- Advantages:
  - HW Access to shared libraries.
  - Complete support for hthread APIs.
- Disadvantages:
  - Interrupts the CPU.
  - Comparatively slow.

Library Call	**Execution**
hthread_create (hthread.h)	160μs
hthread_join (hthread.h)	130μs
malloc (stdlib.h)	122μs
free (stdlib.h)	120μs
printf (stdio.h)	1.66ms
cos (math.h)	450μs
strcmp (string.h)	114μs