Hardware Microkernels for Heterogeneous Manycore Systems

A Play in Three Acts

"And now for something completely different"…..MP

David Andrews
Mullins Endowed Chair in Computer Engineering
University of Arkansas
dandrews@uark.edu
http://hthreads.csce.uark.edu

Play Bill
(Todays Agenda)

- Enter Stage Left
  - –Hthreads for CPU/FPGA Hybrid Components
  - –Unified Programming Model for Custom R.T. Platforms
- Enter Stage Right
  - –The Manycore Movement
  - –Enabling Technologies
  - –Challenges
- The Final Act
  - –Fusing hthreads on Manycore Platforms
- Epilogue
  - – Moving the Hardware/Software Boundary
  - – Will it Catch On ?

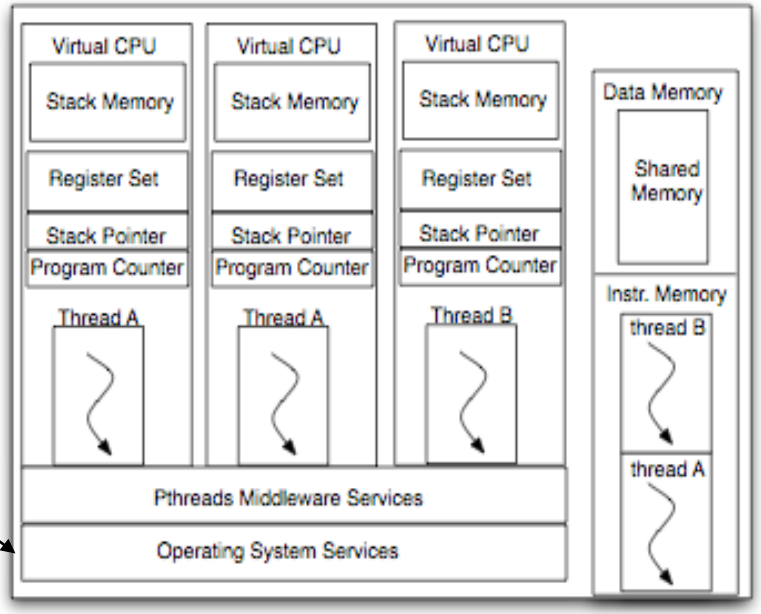
Act 1: Original Hthreads

- NSF Project Originally Developed as Unifying Programming Model for CPU/FPGA Hybrid Embedded Systems
  - Enable Programmers to Specify Computations that Seamlessly run on CPU/FPGA
    - Adopt Familiar Programming Models for hw/sw co-design
    - Open up custom hardware design to Software Engineers/Domain Scientists
  - Pthreads Model Adopted
    - Thread bodies synthesized and mapped into hardware
      - VHDL or C->VHDL
      - Also have used HandelC and Haskell-> VHDL
  - Operating System is Unifying Framework
    - Abstracts Interface
    - Enables Uniform API Calls from hardware/software threads

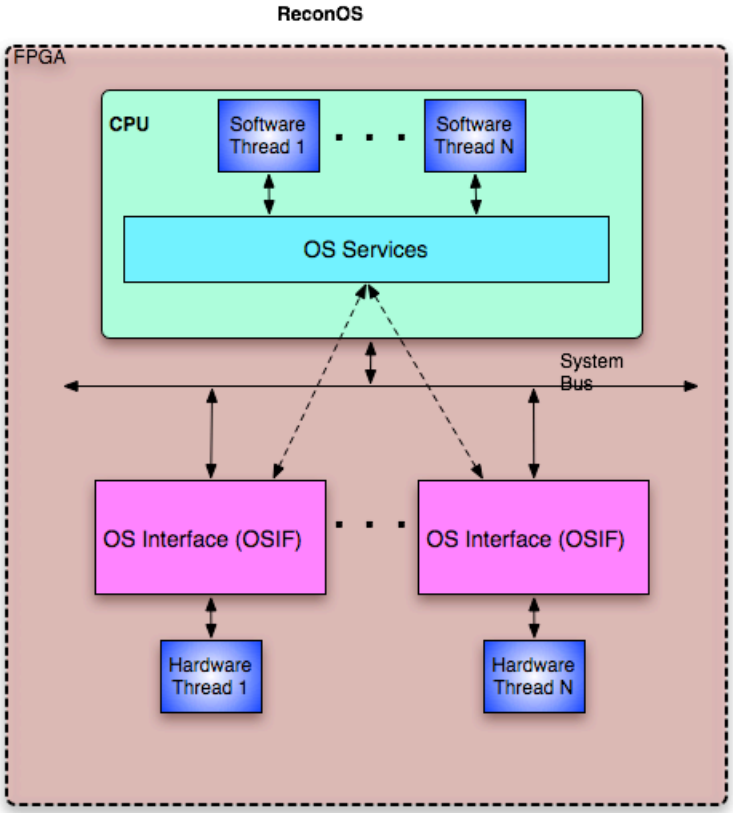
Programmers Perspective: Virtual CPU's Ala Multithreading

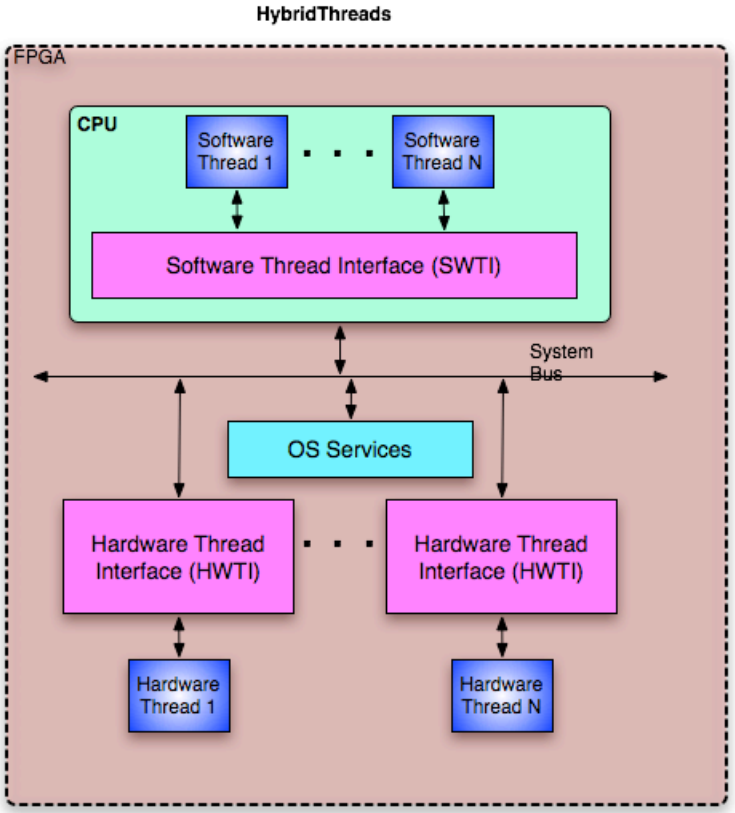
- Programmer Currently Specify 1,000's of Virtual CPU's
  - How Do We Turn Each VM (SIMD/MIMD) into Physical Circuits ?
    - Migrate from Special Purpose towards GP
      - Microblaze, SIMD-Microblaze, Extensible Processor
  - Need Support for Overarching Asynch Concurrency
    - OS/Middleware Define System Services
      - Refinements Driven By Services
        - » Interconnect Networks
        - » Memory Hierarchies
        - » Synchronization/Control

Todays Key To Abstraction:
Synch, Control API's, not
Pragmas in Sequential Language

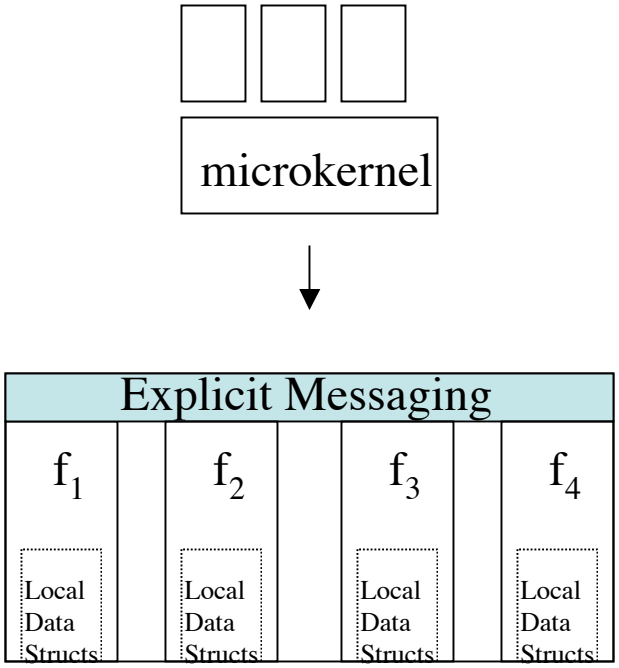


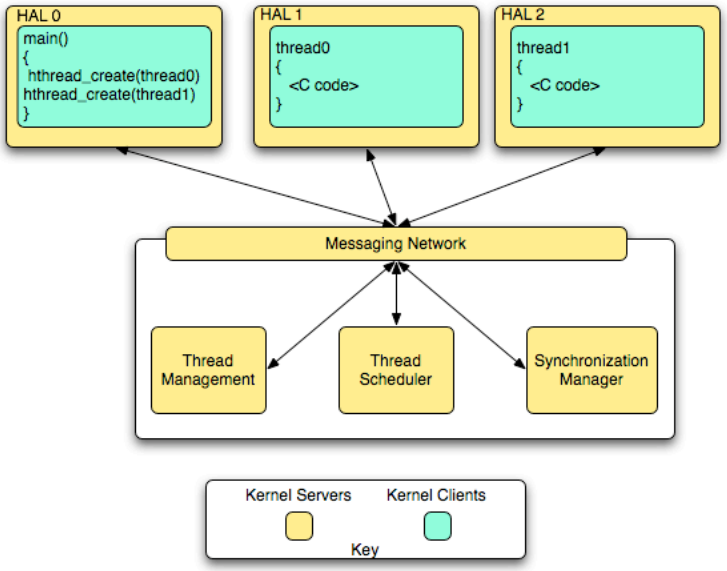
Multithreaded Programming Model
Asynchronous Concurrency





Approach





Explicit Partition = $f_x()$ + Data Structs

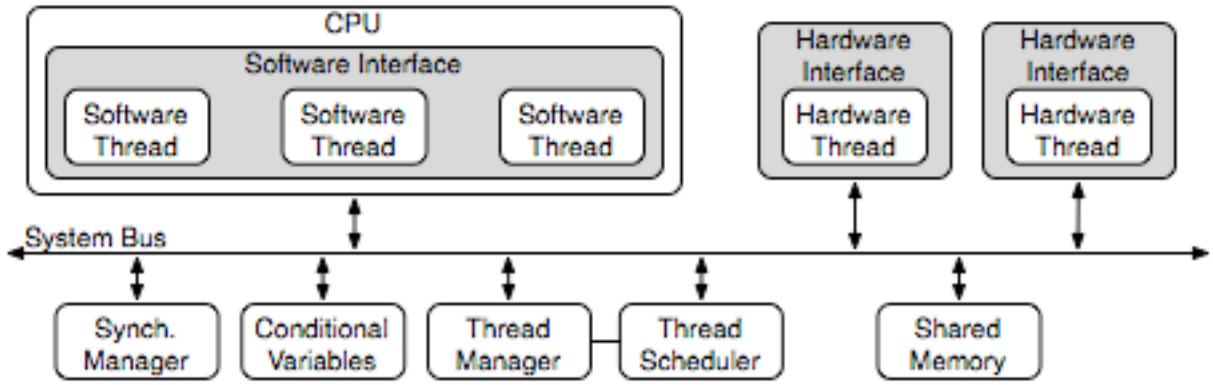
Explicit Messages = fast ld/st single ops

Functions off Application Processors

Hw/Sw Codesign

1. Performance
2. Heterogeneity
3. Scalability

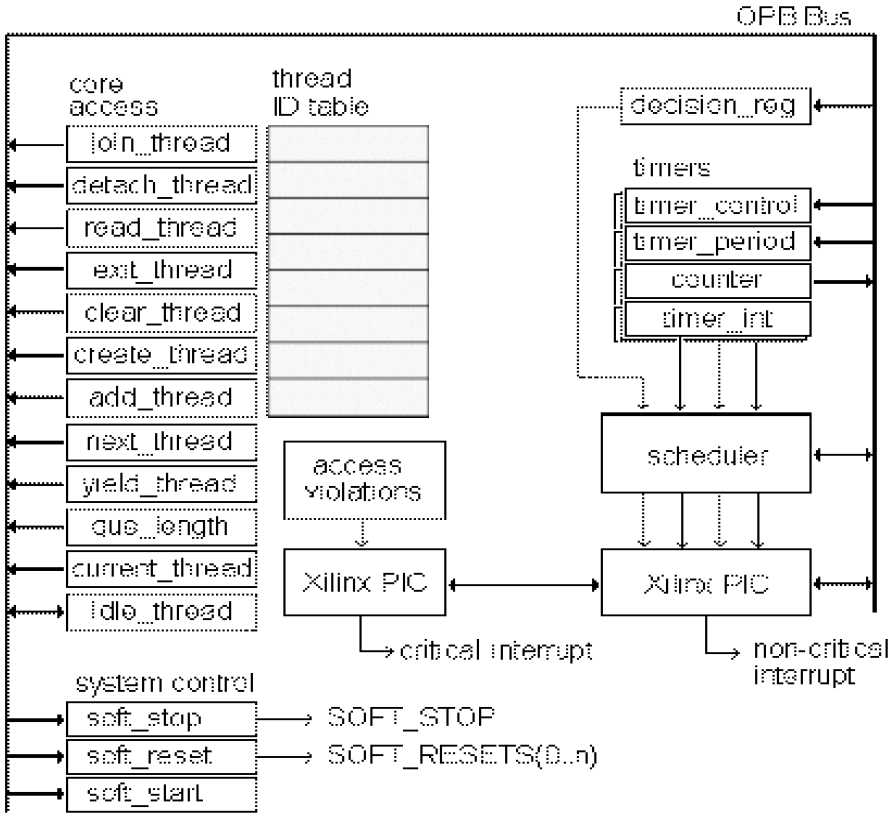
Original hthreads System



- Separate Cores Form Microkernel
  - Breaks up Monolithic Kernel Bottleneck
    - Fast lightweight messaging between cores (load/store)
      - Breaks up Global Data Structures
      - Allows Parallel Operations
      - Resolves Heterogeneity

Thread Manager

- Thread State Migrated Into FPGA (BRAM Key)
  - Parallel State Machines Allow Fast, Concurrency
  - Interface to all "System" Threads Eliminates Interrupt Invocations (Jitter & Overhead)
  - All operations in 12 or Less Clock Cycles
  - All Operations Single Read or Write for Synchronization Between HW/SW Components



Thread API Execution

Atomic Hardware Operations

```
hthread_create()
{
    if( attr->detached )
        threadStatus = create_thread_detached;

    else
        threadStatus = create_thread_joinable;

    if( !hasError(threadStatus) )    {
        threadID = extractID(threadStatus);
        update software data structures
        addStatus = add_thread(threadID);

        if( hasError(addStatus) )
        {
            clrStatus = clear_thread(threadID);

            update software data structures
            return RUN_QUEUE_FULL;
        } else
            thread->id = threadID
    } else
        return NO_THREADS_AVAILABLE;

    return SUCCESS;
}
```

⟹

```
if( thread unavailable ) return 0 + ERR_BIT
thd's status=used,~exited,~queued,~joined,detached
thd's  pid = 0
return thread's ID
```

⟹

```
if( thread unavailable ) return 0 + ERR_BIT
thd's status=used,~exited,~queued,~joined,~detached
thd's pid = current_thread
return thread's ID
```

⟹

```
if( threadID->status=used,~exited,~queued ) {
    threadID->status = queued
    add threadID to R2R_QUEUE
    update R2R_QUEUE
    return 0;
} else return threadID->pid,status,ERR_BIT
```

⟹

```
if( threadID->pid == current_thread ) {
    threadID->status = ~used,~exited,~queued,
                       ~joined, ~detached
    threadID->pid = 0;
    return 0;
} else return threadID->pid,status,ERR_BIT
```

API Timings

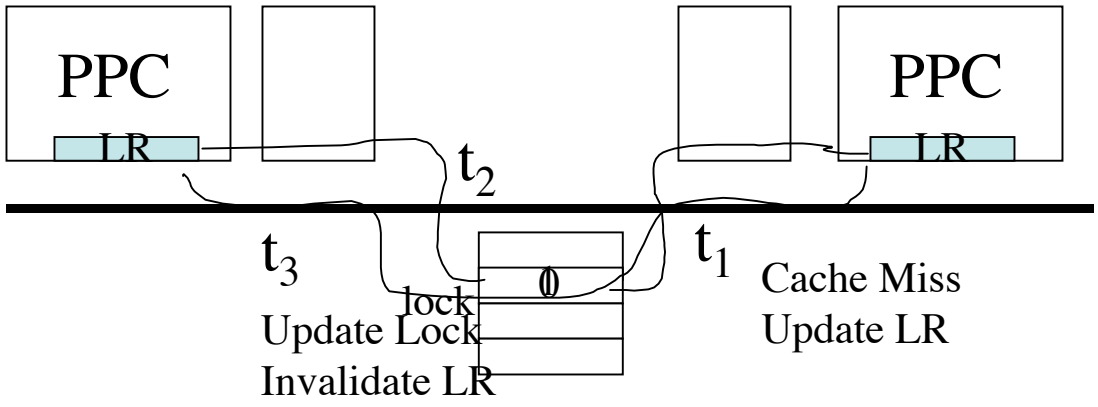
Operation	Time (clock cycles)
Add_Thread	10 + ENQ
Clear_Thread	10
Create_Thread_Joinable	8
Create_Thread_Detached	8
Current_Thread	3
Detach_Thread	10
Exit_Thread	17
Join_Thread	10
Next_Thread	10 + DEQ
Yield_Thread	10 + ENQ + DEQ

DRAM Access Takes 17+ Clock Cycles

Classic Synchronization

Classic SMP Synchronization Using LL/SC Atomic Pairs

t_1		ll R_x,lock
t_2	ll R_y, lock	bne R_x,again
t_3	bne R_y, again	sc R_x,lock
t_4	sc R_y,lock	Beq R_x, again
t_5	beq R_y, again	



Synchronization Manager

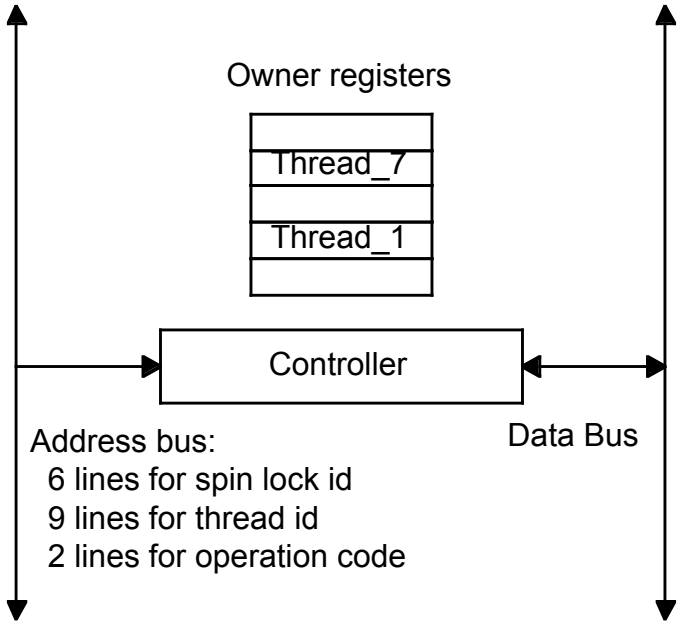
- Accepted "Atomic" Implementations are "pull" then "push"
  - Read semaphore variable (into CPU)
  - Conditionally write (based on active memory coherency circuits)
  - Appropriate for SMP's with smart CPU's/Snoopy Caches
  - Mechanisms on Access order of Variable Address, Not Thread Id

- New Method is Single Load Instruction "push + pull"
  - Mechanism on Thread_id, not access order of Variable Address
    - Thread_id Encoded in Address
    - Return of Load Provides Id of Owner
  - Eliminates need for Snoop Cache Protocols
    - Will be important for scalability !

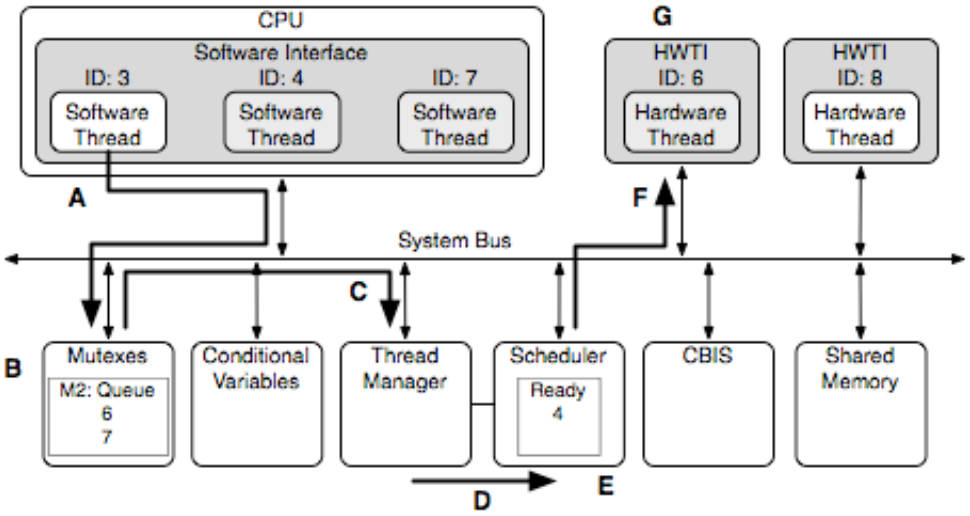
Semaphore Mechanisms

Core Base	Semaphore #	Thread_id	Op

- Thread Id Encoded into Address
  - Request register virtual: Address Range
  - Can Implement Atomic "Swap" in single Read
  - Accessible by any ISA with "Load" Instr.
- Only 1 IP Module for all locks
  - Use BRAM To store owner thread_id
- No Reliance On Cache Coherency



Hthreads in Action



A: Software thread 3 unlocks mutex M2 by calling hthread_mutex_unlock(M2), which sends signal to Mutex Manager.
B: Mutex Manager inspects M2's queue and decides ID 6 will own mutex next.
C: Mutex Manager sends add_thread(6) to Thread Manager.
D: Thread Manager gives ID 6 to Scheduler to add to Ready to Run queue.
E: Scheduler knows ID 6 is a hardware thread, does not add ID 6 to queue.
F: Scheduler instead sends RUN command to ID 6's HWTI.
G: Hardware thread 6 resumes execution, now owner of mutex M2.

Mutex Timings

TABLE V

HW TIMING OF SYNCHRONIZATION OPERATIONS

Operation	Time (cycles)
Lock Mutex	4
Unlock Mutex	5
Try Lock Mutex	3
Get Mutex Owner	3
Get Mutex Count	3
Get/Set Mutex Kind	3

TABLE VI

INTEGRATED TIMING OF SYNCHRONIZATION OPERATIONS

Operation	Avg Time (ns)	Std. Dev. (ns)
Lock Mutex	1524.54	52.19
Unlock Mutex	1097.63	27.33

Scheduler

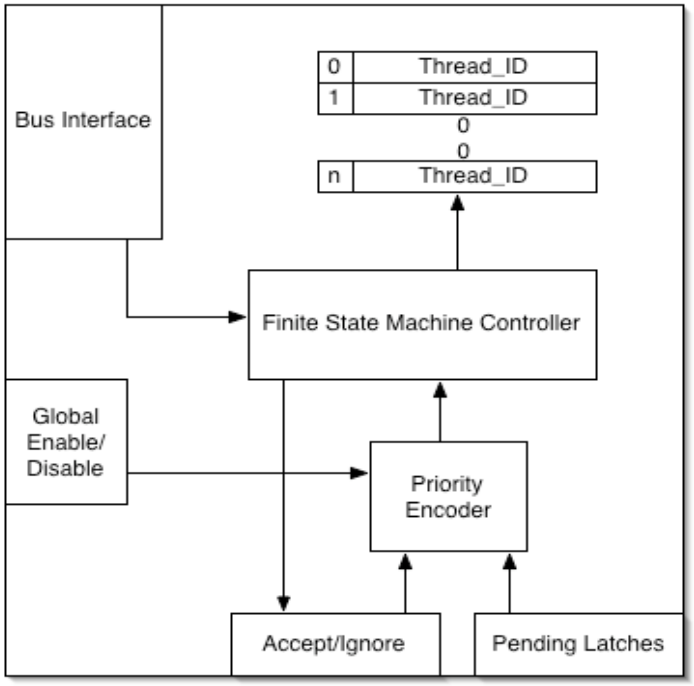
- Existing semantics
  - Timer Interrupt Expires, Thread Unblocked on Semaphore (Overhead)
  - Automatic Context Switch to Scheduler (Overhead)
  - Check for New Thread (Jitter + Overhead)
    - Variable Search Times in Scheduler Queue
    - May or may not choose different thread
  - Context Switch to New Thread (Overhead)
- Hthreads: All Scheduling Requests to HW Scheduler
  - External: Timer, Unblocked Semaphores, Interrupt Requests
  - Internal: Thread join, suspend, terminate, change priority
  - Scheduler runs in parallel with current thread
    - Will Only Generate Context Switch if Appropriate
      - Timer expiring may/may not cause context switch
      - Unblocked thread may/may not cause context switch
      - Change in a thread priority may/may not cause context switch
- Scheduler Decision Function 12 clock cycle Fixed Time
  - Independent of # Threads For Scalability

Scheduler Timings

Operation	Time (clock cycles)
Enqueue(SWthread)	28
Enqueue(HWthread)	20 + (1 Bus Transaction)
Dequeue	24
Get_Entry	10
Is_Queued	10
Is_Empty	10
Set_Idle_Thread	10
Get_Sched_Param	10
Check_Sched_Param	10
Set_Sched_Param(NotQueued)	10
Set_Sched_Param(Queued)	50

CBIS

Change Semantics of External Interrupts
Bring within Schedulers Envelope
  Register User Thread with Modified PIC
  Intr Rqst Causes Thread Scheduling Event to
   Thread Manager -> Scheduler



Act 2: Dawn of the Manycores

"Groundhog Day" Back to Parallelism Once Again

1. Dynamic ILP Run It's Course
   - Performance Scaling Ebbing
     - Not Much Juice to Be Squeezed in ILP
     - High Transistor Costs for Small Return
   - Power + Memory Wall = Brick Wall (View from Berkeley)
2. Manycore Architectures Following Moore's Law ?
   - Multiple Simple CPU's for Parallelism
     - MIMD/SIMD Heterogeneity
       - Better Use of Dense Interconnect
       - Will This Break Memory Wall ?
3. Manycores Good News for Real Time Embedded Systems
4. Parallelism + Power 1st Class Design Issues

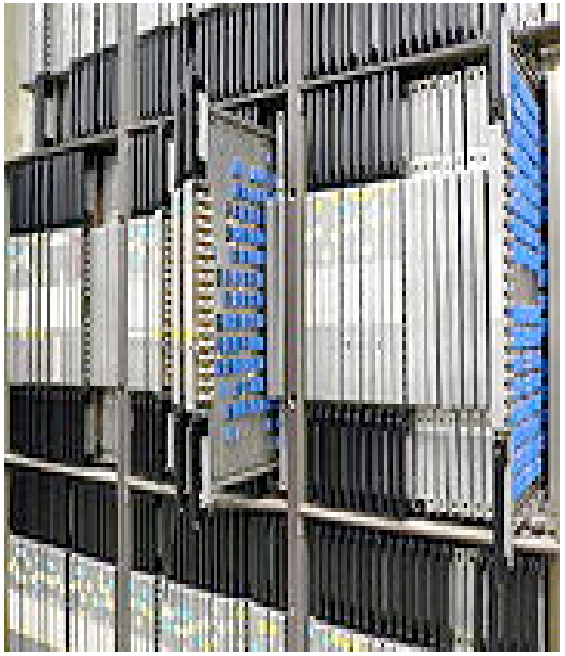
Also….It's Deja Vous All Over Again…Rebirth of Parallel Processing

Manycore Status

- Paradigm Shift Occurred Without Considering Software Infrastructure
  - Concerning as Prior Efforts a Failure
    - We did not Resolve Parallel Programming Models
    - We did not Resolve Run Time Systems
  - New Considerations
    - Magnitude of Parallelism Will Be Greater
    - Heterogeneity of New Applications
- Complete Technology Infrastructure Riding on Success

…..and We Are Not Quite Ready

Parallel Processing Era: Lots of Fun !
The war of the machines



Peak Performers, Usually Vector/SIMD Data Parallelism

-Hard to Program

-Expensive

Which PP Won ?
(Economics & Usability)

Video Killed the Radio Star The Buggles 1979 (First MTV Video)

Economics := Commodity
- Nodes
- Interconnects
- Sequential Languages
- Operating Systems

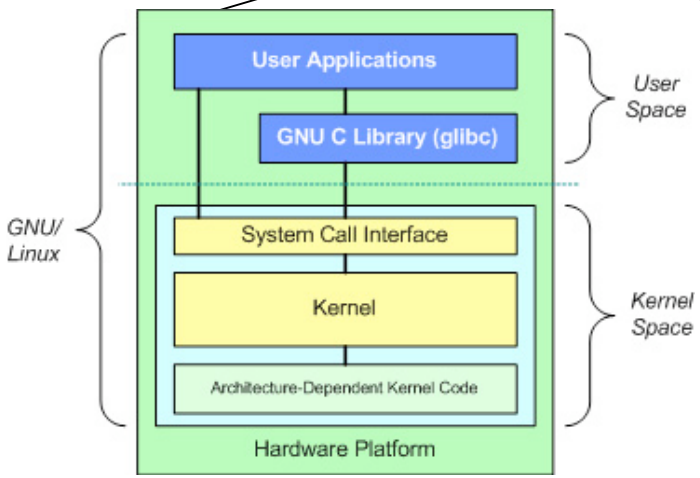
Usability := Familiarity
- MIMD Parallelism
- Linux
- Middleware

Victim of Our Own Success:

Side Effect was Broad Research in OS's Ebbed Research on Vaneered Middleware Layers

Image from Wikipedia

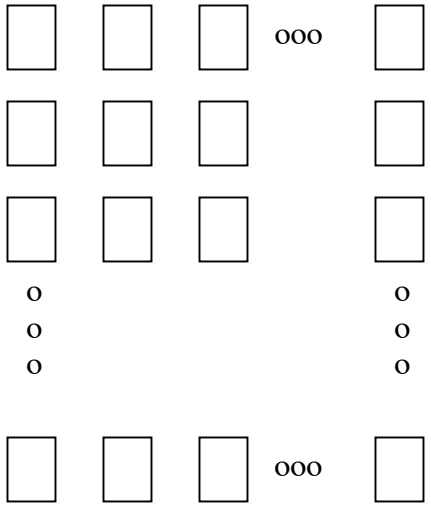
Thesis: *Classic Monolithic OS May be Retired Along with Power Hungry Dynamic ILP Processors*



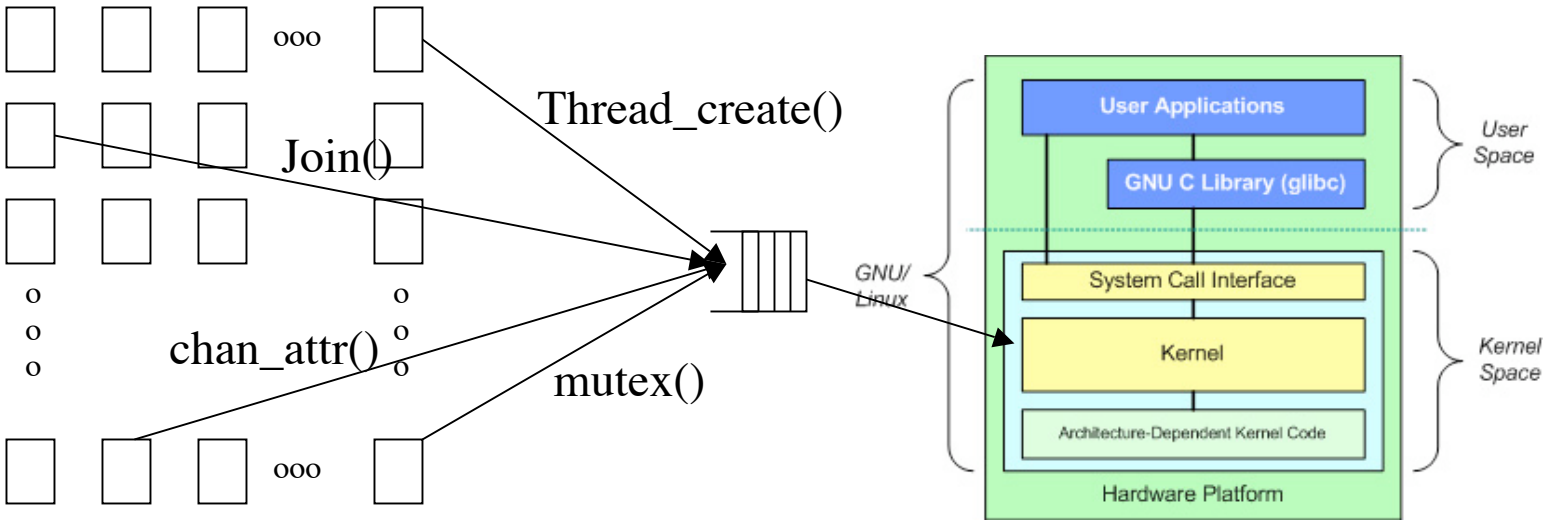
SCALABILITY

Seq to Parallel Chasm

HETEROGENEITY



Obvious Scalability Issues



1)Wasteful if large image replicated in memory hierarchy

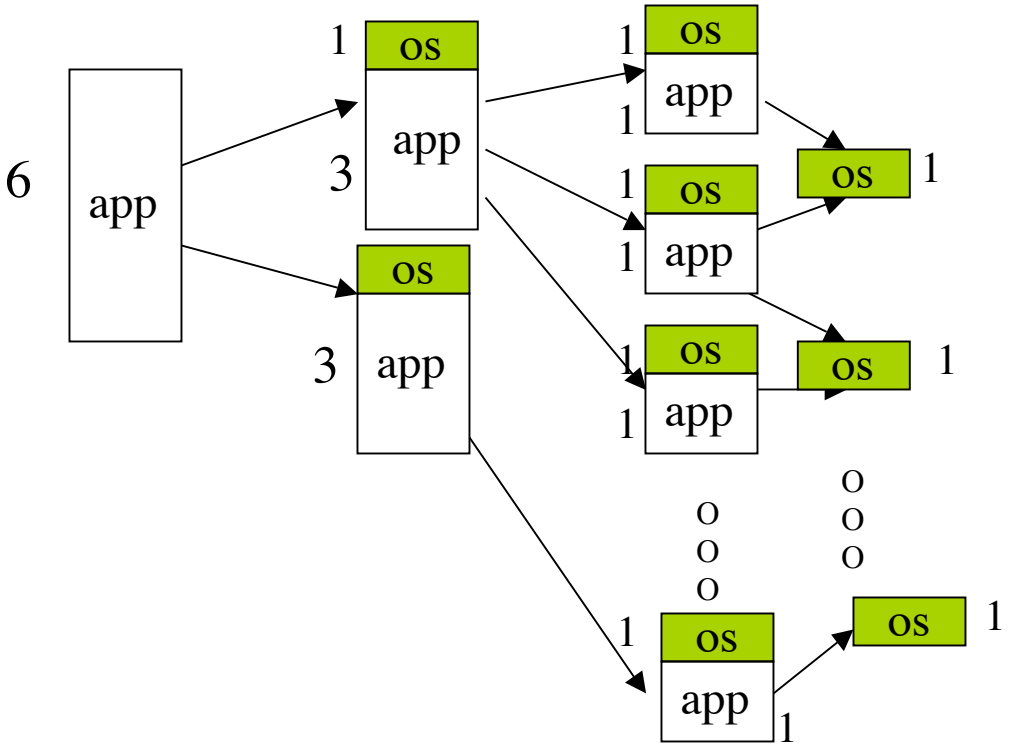
2) Shared data structures enforce sequentiality,
Implications on Contention, Latency,Caches

3) Scheduler focused on time and not space multiplexing

Image from Wikipedia

Scalability Brings Efficiency Issues

(A Little Scary to Me !)

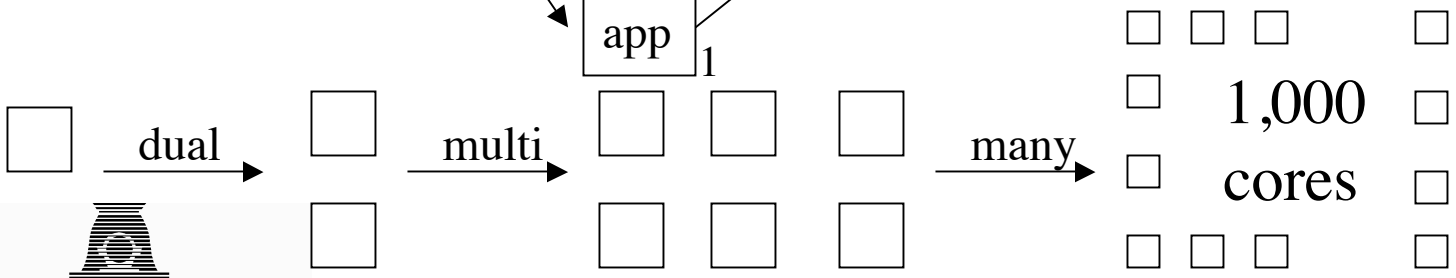


$$\text{Amdahl} = \frac{1}{(1-f) + \frac{f}{Sp}}$$

f => #threads > #Cores

$$OS_{eff} = \frac{app}{app + OS}$$

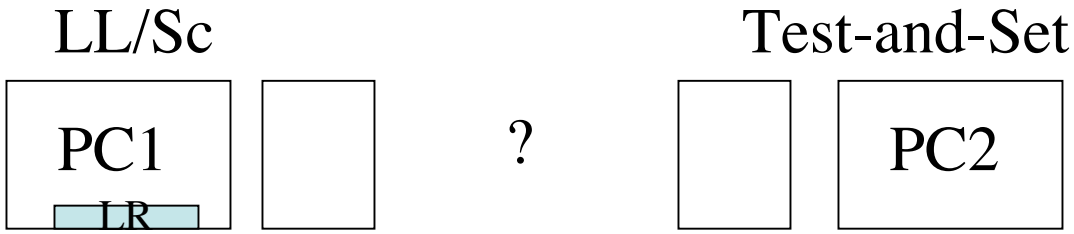
OS increasing



Heterogeneity Issues

- Amdahls law also pointing to heterogeneous cores
  - Scalar cores for threaded data processing
  - SIMD cores for audio/video/signal processing
- Heterogeneity Issues permeate abstractions
  - Unifying Programming Languages/Models
    - Not just Heterogeneous Accelerators
  - Run Time Systems
    - Scheduling, Synchronization, Thread Management
    - All Resources Under OS Scheduler Control
  - Compilation
    - Closely Related but Not Today

Heterogeneous Synchronization



Issues:

Different ISA's Collide

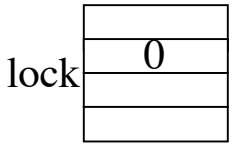
-LL/Sc versus TAS

Reliance on Snoopy Cache Protocol

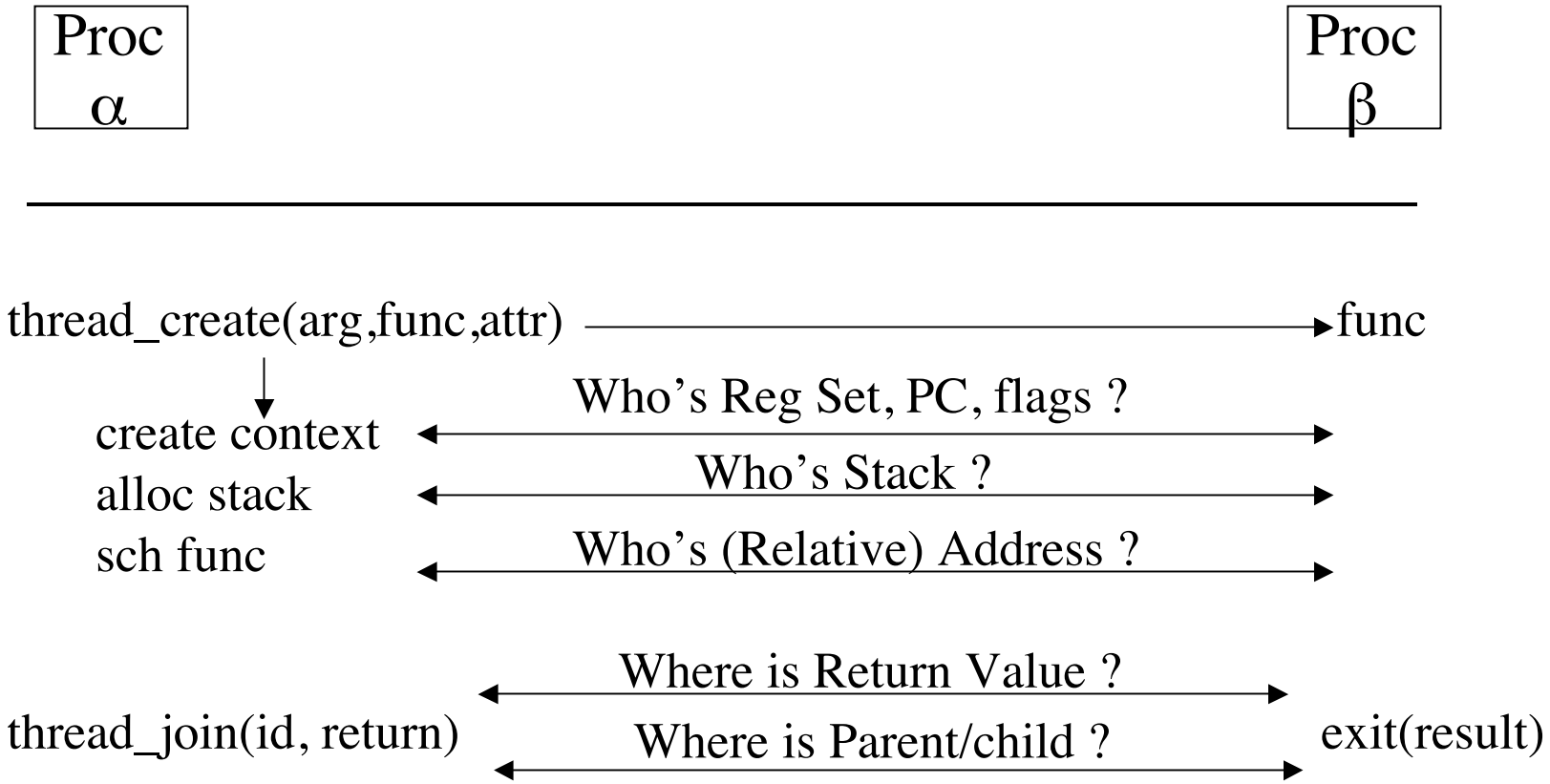
-Doesn't Support Hetero Semantics

-Doesn't Scale Well

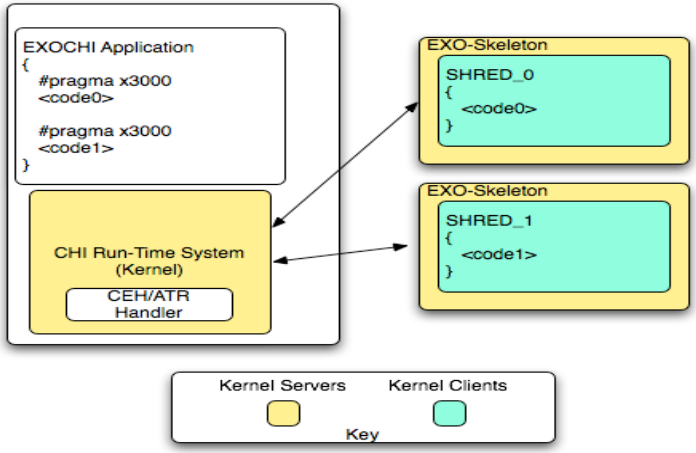
-May not Even Be in System (ala Cell)



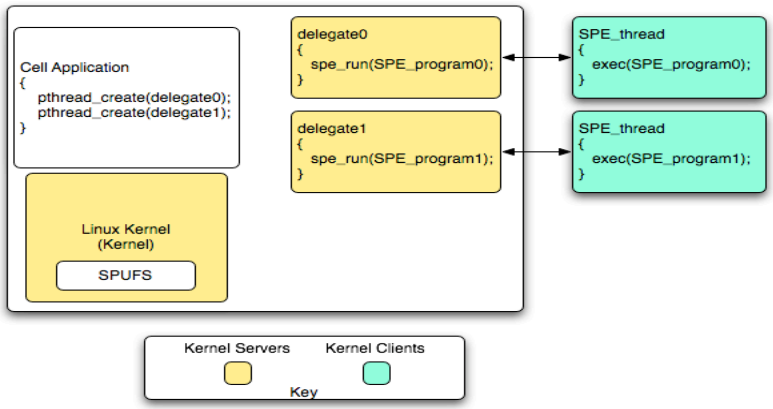
Thread Management Issues



Some Paths Followed So Far…
Client Server OS Models



EXOCHI



CELL

OpenMP Threaded Model
Application Managed Exosequencers
RPC for VM, Exception Handlers

pthreads delegate thread
SPE Thread Synchronous
RPC for OS Calls

Main OS Not Unifying Abstraction
Picks at Heterogeneity Issues
Not Addressing Scalability Issues

So Far….

Good Efforts at Bridging Abstractions

- Bringing in Heterocomponents Under thread Programming Model
- Some Bending/Breaking of Model

Efforts Represent Hierarchical (Subordinate) Operating Systems

- Not Yet Single Abstraction Most Familiar to Programmer
  1. Separate Scheduling Models
  2. Can This Even Be done ?

Efforts Not Addressing Scalability Issues

- 1,000's of Threads will Kill Performance

Act 3: Fusing hthreads with Manycores

Can We Use Hthreads as Unifying Framework ?

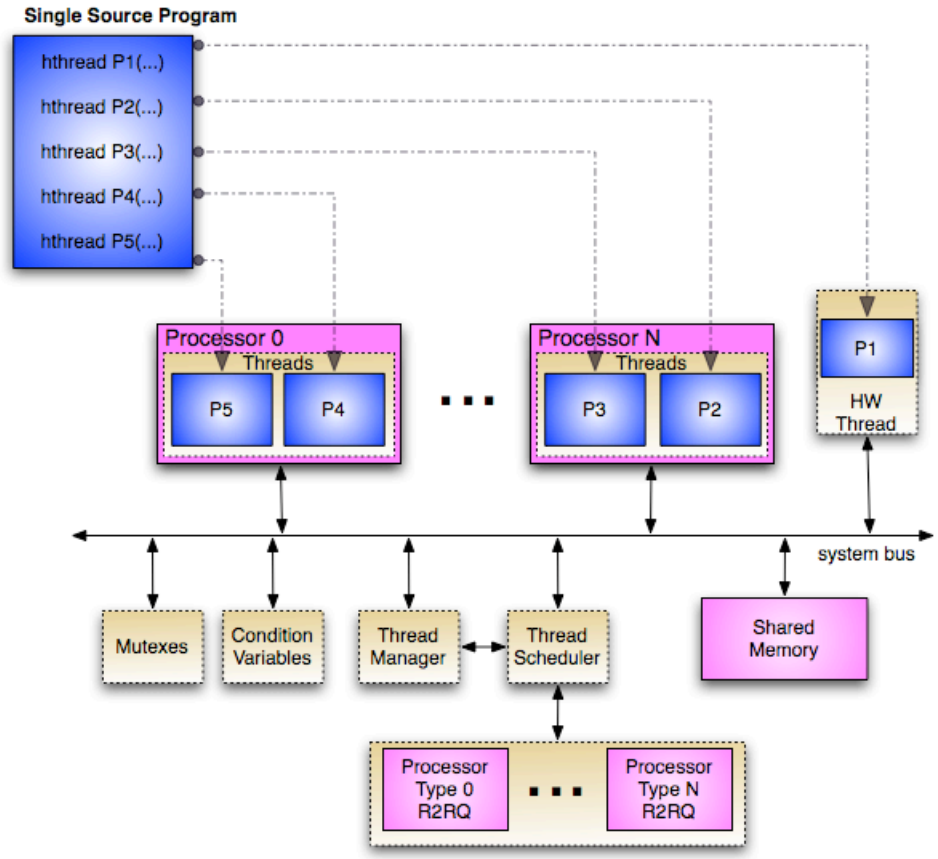
1) Already Abstracts Heterogeneous Differences
    1) Programmer Shouldn't Need to Know
    2) Uncouple Synchronization Primitives from ISA's
        1) Do Not Rely on LL/SC, test&set
2) Seamlessly Scales
    - No RPC Mechanisms
    - Minimal OS Overhead
    - Does Not Rely on Snoopy Cache Protocol
3) No Hetero Accelerator Model
    - Scheduler Treats All Resources First Class Schedulable Objects

hthreads for Heterogeneous Manycores

- Difference is largely within Computational Units
  - Substitute Processors for Custom Circuits
- Hthreads OS Cores Serve as Unifying Framework
  - Cores did not change !
  - Back to linkable libraries in place of FSM I'face
- Cores Interesting Enabling Technology
  - Resolves Heterogeneity (well almost…)
  - Provides Scalable Low Latency OS Services
  - Breaks up Monolithic Bottlenecks

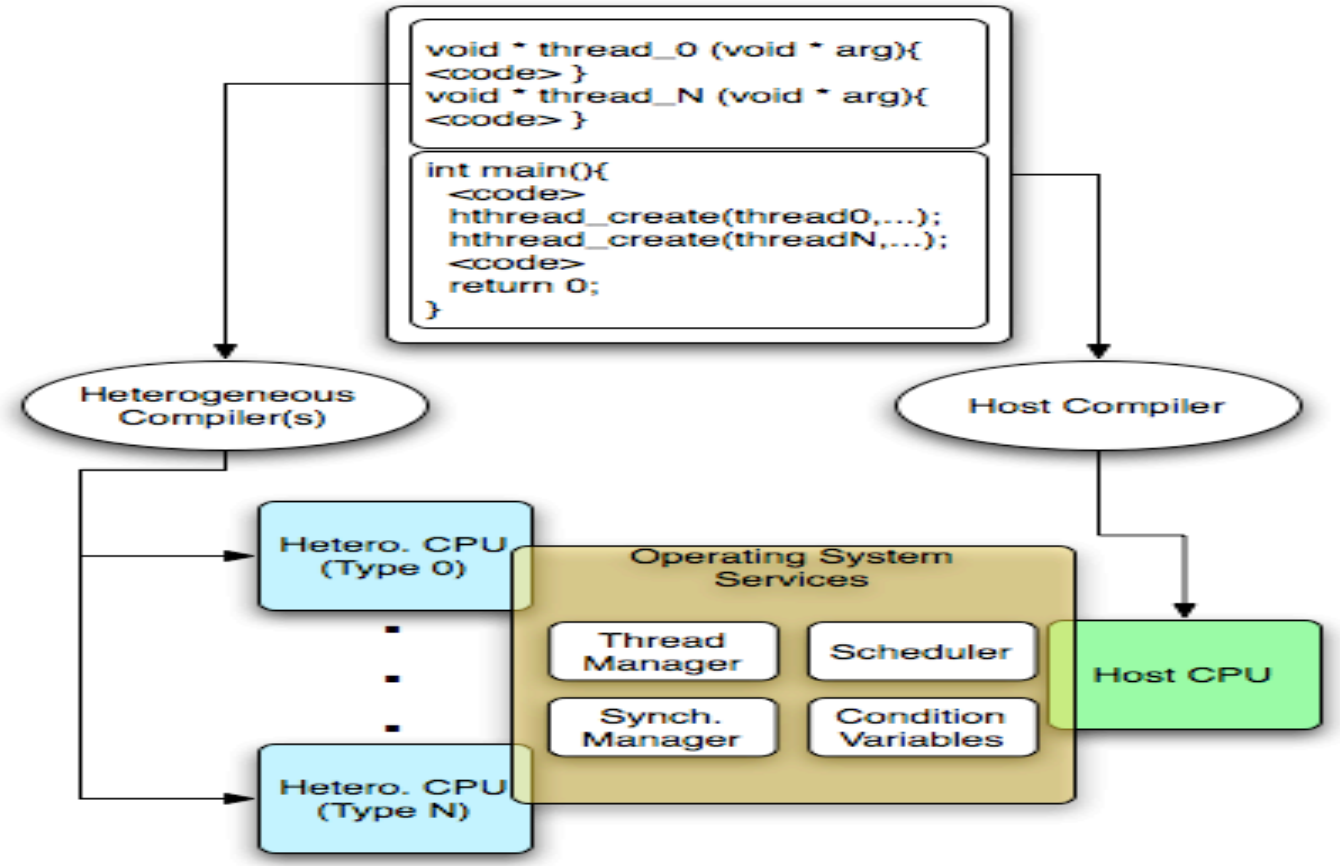
Keeping a Unified Thread Model

Eliminate Subordinate OS Model
- Scheduler Controls All Processors
- Single Program
- Scalable to 1,000's Cores
  - Maintain RT Performance



Need New Heterogeneous Compilation Framework

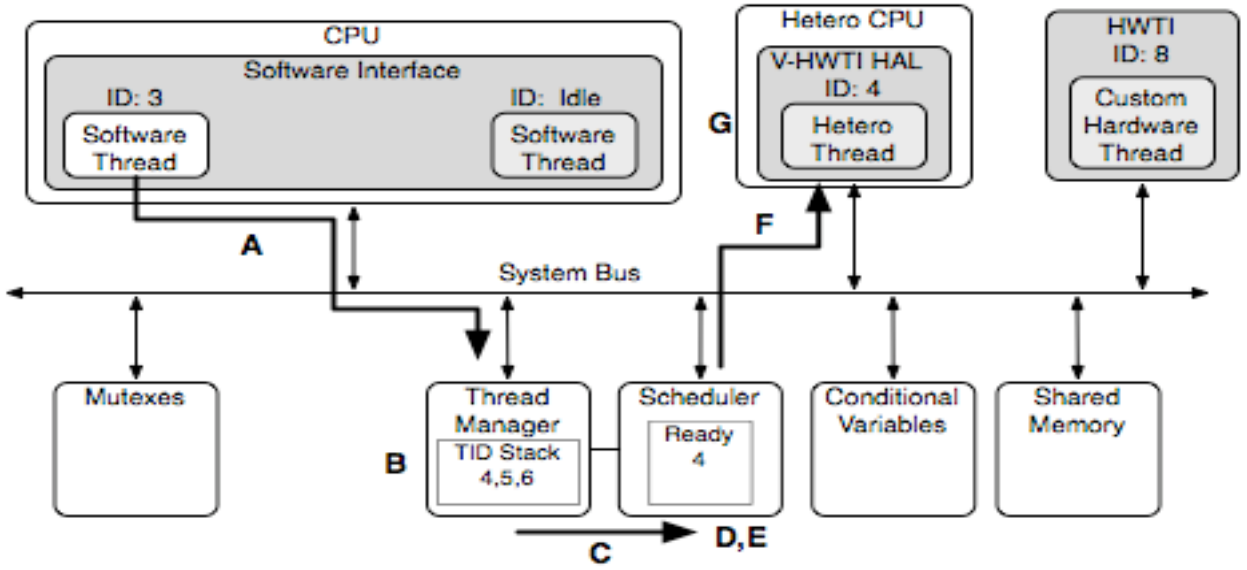
Our Experimental Ad Hoc Approach



Code Snipet

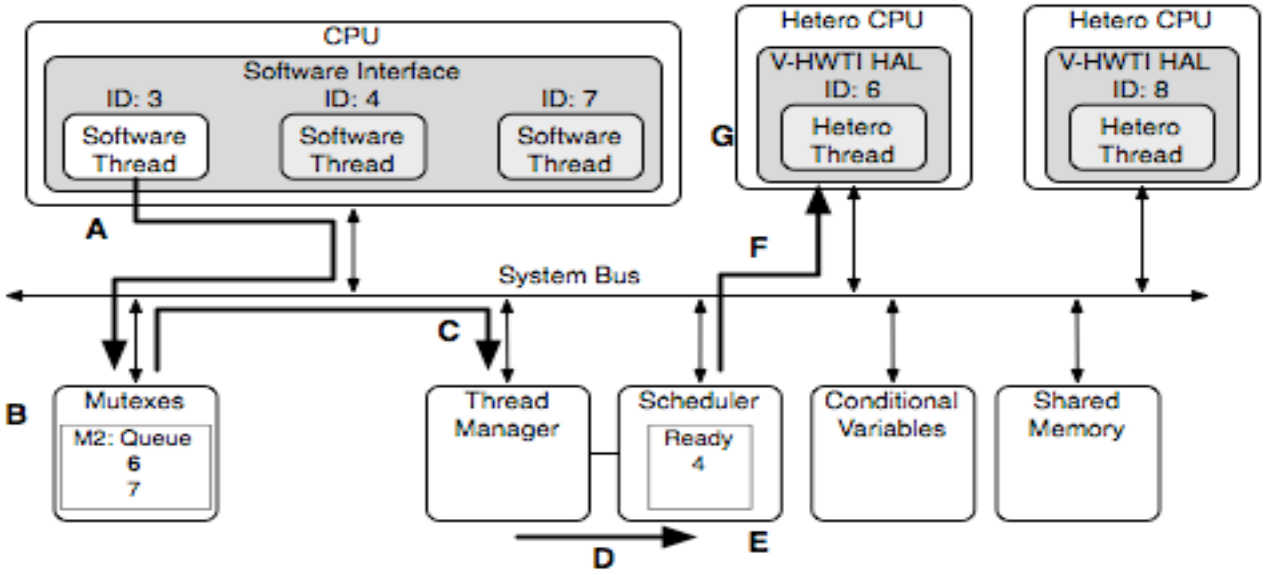
```
int main(){
    sortarg_t arg;
    int mutexnum = 0;
    int condnum = 0;
    hthread_t       tid[NUM_THREADS];
    hthread_attr_t attr[NUM_THREADS];
    // **********
    extern unsigned char intermediate[];
    extern unsigned int mbox_handle_offset;
    unsigned int mbox_handle = (mbox_handle_offset) + (unsigned int)(&intermediate);
    // ******************************************
    // Initialize thread argument and mailboxes
    arg.num_elements = CHUNK_SIZE;
    mailbox_init_no_globals(mutexnum++,condnum++, &arg.mb_start, NUM_CHUNKS);
    mailbox_init_no_globals(mutexnum++,condnum++, &arg.mb_done, NUM_CHUNKS );
    int i = 0;
    // Create threads
    for (i = 0; i < NUM_THREADS; i++)    {
        // Initialize attributes
        hthread_attr_init( &attr[i] );
        hthread_attr_sethardware( &attr[i], (void*)base_array[i] );
        // Spawn thread
#ifdef USE_HW_THREAD        hthread_create( &tid[i], &attr[i], (void*)mbox_handle, (void*)&arg );
#else                       hthread_create( &tid[i], NULL, mbox_thread, (void*)&arg );
#endif
        }
```

Creating A Heterogeneous Thread



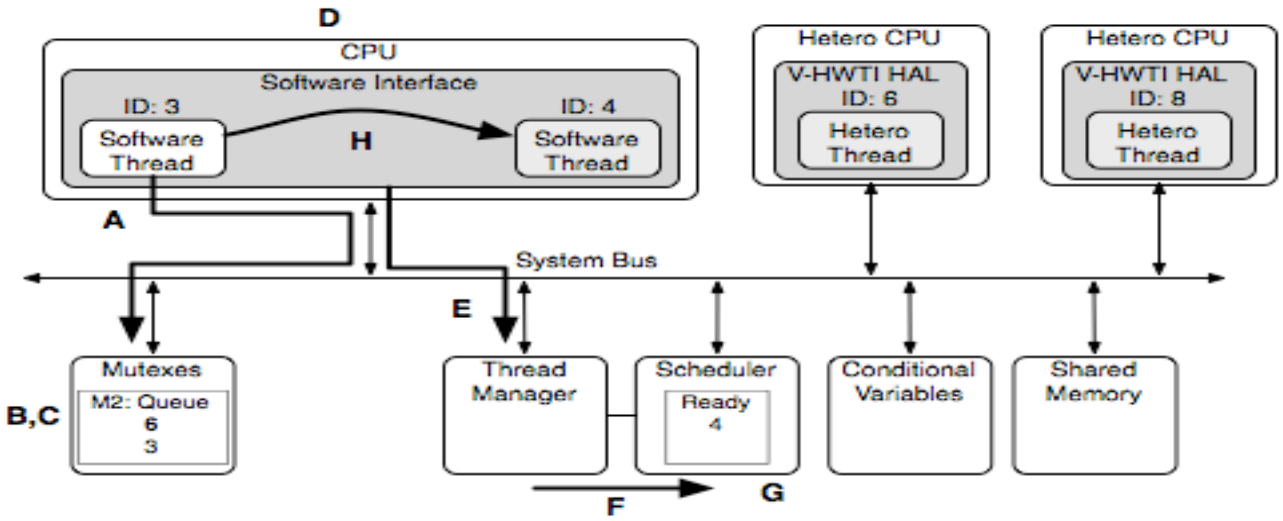
A: Thread 3 calls pthread_create which interacts with the Thread Manager.
B: Thread manager allocates a fresh thread identifier (TID 4).
C: Thread manager submits an ENQ request of TID 4 to the scheduler
D: Scheduler handles ENQ request by querying the attributes for TID 4.
E: Scheduler finds that ID 4 is a heterogeneous thread, and prepares to signal
F: Scheduler sends a SIGNAL to TID 4's V-HWTI.
G: Heterogeneous thread begins execution

Mutex Unlock



A: Thread 3 unlocks mutex M2 by calling hthread_mutex_unlock(M2), which sends signal to Mutex Manager.
B: Mutex Manager inspects M2's queue and decides ID 6 will own mutex next.
C: Mutex Manager sends add_thread(6) to Thread Manager.
D: Thread Manager gives ID 6 to Scheduler to add to Ready to Run queue.
E: Scheduler finds that ID 6 is a hetero. thread, does not add ID 6 to queue.
F: Scheduler instead sends *SIGNAL* command to ID 6's V-HWTI.
G: Hetero thread 6 resumes execution, now owner of mutex M2.

Mutex Lock



A: Thread 3 locks mutex M2 by calling hthread_mutex_unlock(M2), which sends signal to Mutex Manager.
B: Mutex Manager inspects M2's queue, sees that ID 6 owns the mutex.
C: Mutex Manager blocks TID 3 by placing it in M2's blocked queue, and returns blocked status to the caller.
D: Software interface receives blocked status, must now context switch.
E: Software interface asks for the next thread from Thread Manager.
F: Thread Manager requests a DEQ operation from the Scheduler.
G: Scheduler tells Thread Manager that TID 4 should run next.
H: Software Interface receives the next thread, context switches to TID 4.

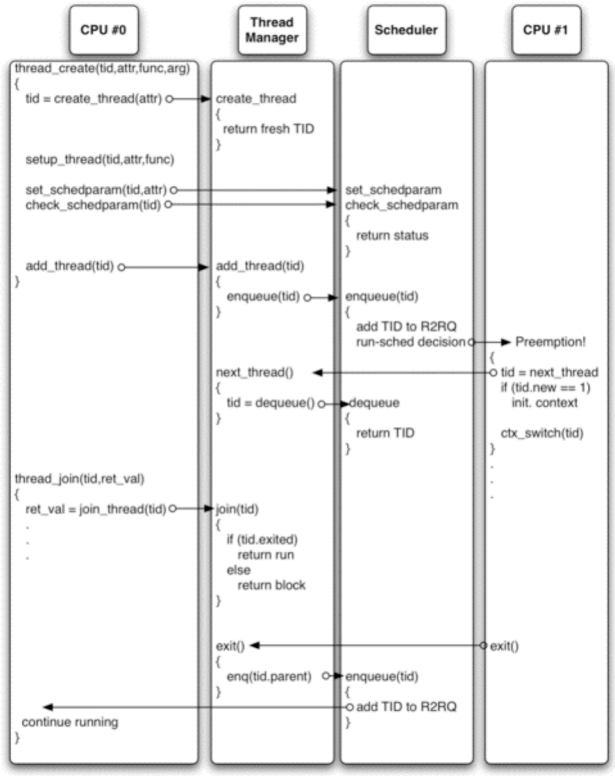
RPC/hthread Core Comparisons

System Call	Execution Time (from HW)	Execution Time (from PPC)	Execution Time (μBlaze)
`create`	160μs *	40.8μs	12.5μs
`join`	130μs *	65.7μs	13.9μs
`mutex_lock`	0.36μs	12.0μs	2.7μs
`mutex_unlock`	0.36μs	11.9μs	3.0μs

- RPC Call from Custom Circuit to OS on PPC
  - create_thread( )
    - 160usec versus 40.8usec PPC & 12.5 usec MBlaze
  - join( )
    - 130usec versus 65.7 usec PPC & 13.9 usec MBlaze

Current Work

- Thread Manager/Scheduler
  - Generic Thread Create
    - New Sequence of OS Messages
    - Return Values
  - > 2 Generic Processor Group Scheduling
    - O(1) on each Group



Epilogue: Will it Catch On ?

(Probably not with this Audience :-) !

- Does Require Moving the Hw/Sw Boundary
  - Targets Appropriate PP not Scalar Core Abstractions
- Historical Precedence
  - Early Days
    - Floating Point, Function Calls, Stacks
  - More Recently
    - VT Technology for Virtual Machines
    - EXOCGI -> Pangaea
- Programmers Will NEED Efficient Methods
  - Will Happen When Demand Dictates