Enabling Task Level Parallelism in HandelC

Thamer AbuYasin

Acknowledgements, and many thanks

- I would like to thank my committee members
  - Dr. Andrews
  - Dr. Alexander
  - Dr. Agah

  For their patience, guidance, assistance … for everything

- Also, I would like to thank the entire population of the CSD Lab for their help and the friendly atmosphere, especially my teammates in the Hthreads project, Jason, Erik, Jim, Fabrice, Wes, Seth, Shane, Elias and the original cast that I never got to meet

Academic and Professional Background

- Received a bachelor of science in electrical engineering from the University of Jordan, spring 2003
- Worked in the field of electronic and information systems for two years after graduation
- Joined KU in fall 2005 to pursue a master of science in computer engineering, under a Fulbright scholarship

Agenda

- Problem Statement
- Background & Related Work
- HCthreads: Design and Implementation
- Support Utilities
- Results
- Conclusions and Future Work

Problem Statement

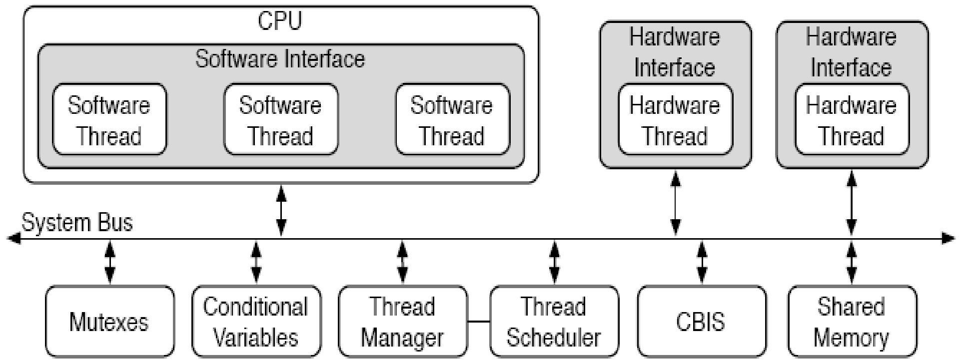
Instruction Level Parallelism vs. Task Level Parallelism

- Majority of efforts up to date focused on ILP
  - PRISM and DISC first attempts to accelerate applications through instruction set metamorphosis, limited ILP
  - GARP one of first studies to directly address ILP, processor and reconfigurable fabric on the same chip, gains less than overhead
  - General purpose processors followed similar approach
  - Final conclusion ILP is limited

Problem Statement

Instruction Level Parallelism vs. Task Level Parallelism

- New emphasis on TLP
  - Multiprocessor Systems on Chip, Parallel Computing
  - FPGA solutions such as Hthreads, Milan's, ReconOS and Thread Warping



Hthread system block diagram   Erik Anderson

Problem Statement

- The objective of this thesis is to merge the capabilities of modern TLP with the existing ILP capabilities of HandelC
- HandelC has a large domain of users
- HCthreads is to bring modern programming techniques and model to that base
- Enhancing the programming model through combining ILP and TLP capabilities will bring additional performance

Problem Statement

Contributions of This Thesis

- First contribution is a threading library called HCthreads based on Pthreads, major components:
    - The Dispatcher
    - The Terminator
    - The Functional Units

- Second contribution is a support library that enables the use of HandelC cores on platforms not supported by Celoxica using the Hthreads system

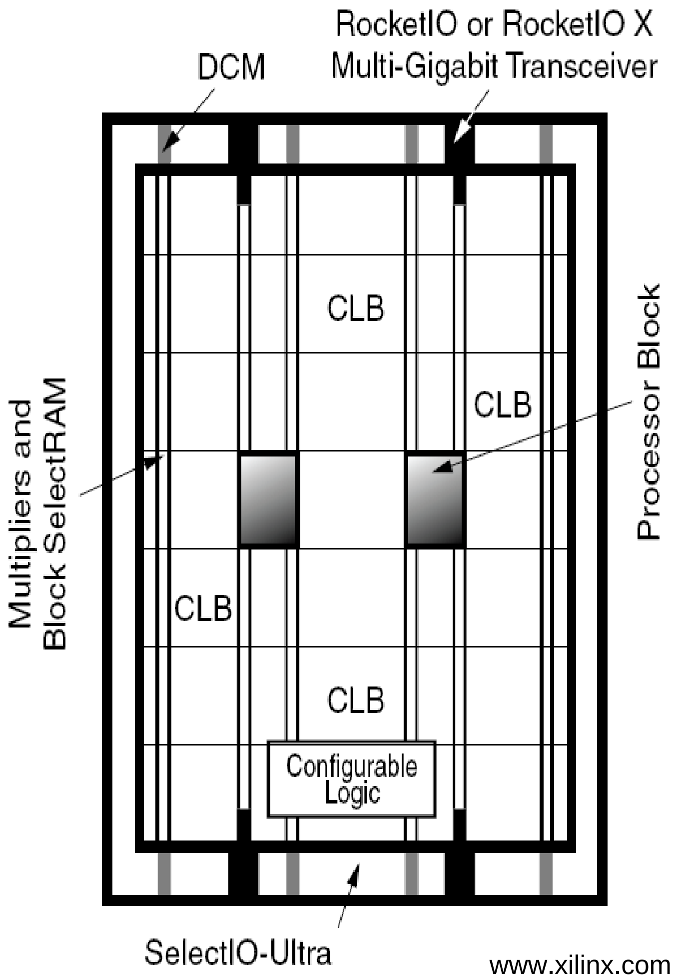
Agenda

- Problem Statement
- Background & Related Work
- HCthreads: Design and Implementation
- Support Utilities
- Results
- Conclusions and Future Work

Background & Related Work

Field Programmable Gate Arrays

- Started with Estrin in 1959
- Popular nowadays in different fields



www.xilinx.com

Background & Related Work

C 2 Hardware

- Many available Academic and Commercial Tools
  - HandelC is one of the most popular
- Main objective is to bridge the HW/SW boundary
- Most target a SIMD computational model
  - extends ILP approach
- Support a subset of ANSI C, pointers and recursion are not supported
- Add pragmas to guide the translation process

Background & Related Work

HandelC

- HandelC is based on the CSP algebra
- Each assignment must occur in one clock cycle

  A = (C + V + D) / G

  - This will generate deep logic
- Provides the “par” construct to express SIMD operations
- Does provide some TLP level primitives but no runtime support (counter intuitive for programmers)
  - Spinning semaphores and channels
  - User can create multiple main functions

Agenda

- Problem Statement
- Background & Related Work
- HCthreads: Design and Implementation
- Support Utilities
- Results
- Conclusions and Future Work

HCthreads: Design and Implementation

HCthreads Design

- Pthreads provides support to implement heterogeneous threads on different platforms, very comprehensive set of features
- Many of the Pthreads features are not required
  - HCthreads targets homogeneous threads
  - Processing cores in HCthreads are truly parallel not pseudo concurrent

- pthread_create()
- pthread_exit()
- pthread_cancel()
- pthread_join()
- pthread_detach()
- pthread_kill()
- pthread_mutex_destroy ()
- pthread_mutex_lock ()
- pthread_mutex_trylock ( )
- pthread_mutex_unlock ()
- pthread_cond_signal()
- pthread_cond_wait()

HCthreads: Design and Implementation

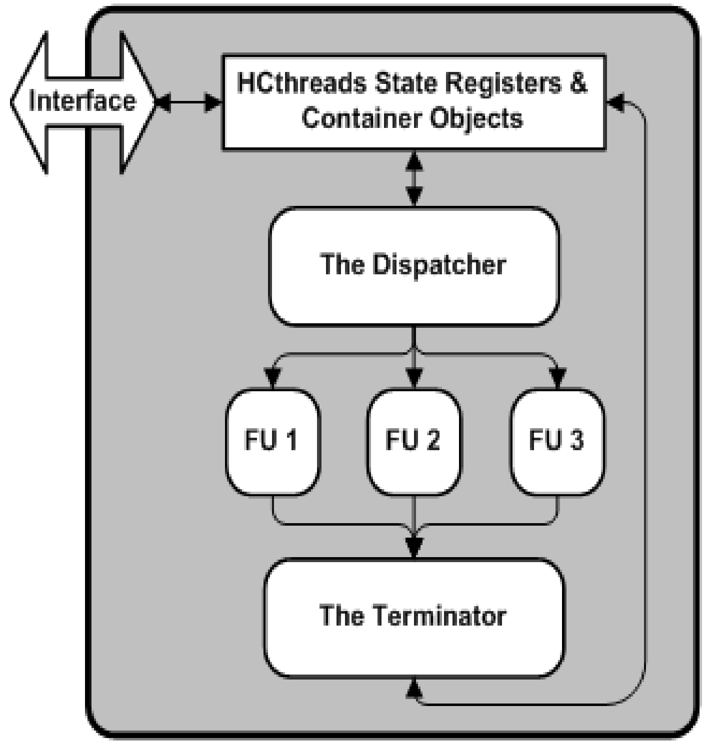
HCthreads Implementation: Attributes

- DETACHED is to replace pthread_detach and related attributes, defines if all threads in the systems are detached or joinable
- CONTAINER_SIZE, defines the number of entries in the ready to run container, different applications require different number of entries
- R2RSTACK, defines if the ready to run container will behave like a stack or a queue, solves the breadth first search problem
- NO_FNUNITS, defines the number of parallel functional units in the system

HCthreads: Design and Implementation

HCthreads Implementation: Components

- The Dispatcher, a light weight scheduler responsible for assigning threads to functional units,
- The Terminator, a central location where all functional units report when the current thread has completed its computation
- The Functional Units, multiple engines each running a separate copy of the accelerated function



HCthreads: Design and Implementation

HCthreads Implementation: Interface

- All previously mentioned attributes should be defined by the programmer
- Programmer needs to define the accelerated function
- Programmer needs to define the input argument structure
- hcthread_create is used to create threads, comes with two signatures depending on the employed joinable or detached threading scheme
- hcthread_join is used to join on threads only if a joinable scheme is used

HCthreads Implementation: Data Structures

- Threads and functional units state,
  - bit fields with each bit representing a thread or a unit,
  - a high bit indicates a free resource and a low bit indicates a busy resource
  - Simpler circuits to check for free resources and to update state

- Ready to run container, keeps order of created threads, can behave like a stack or a queue

- Input argument array, parallels the ready to run container and maintains a copy of the input argument for created threads

Agenda

- Problem Statement
- Background & Related Work
- HCthreads: Design and Implementation
- Support Utilities
- Results
- Conclusions and Future Work

Support Utilities

First prototype: Simple Data Streaming

- Integration with Hthreads can extend the use of HandelC cores on platforms not supported by Celoxica
  - VHDL wrapper required to interface HandelC cores into the HWTI
  - HandelC cores act as slaves to VHDL wrappers
  - the VHDL wrapper marginalized this approach to only support a streaming model

Support Utilities

Current Solution: Full integration with Hthreads

- In this approach the HandelC core assumes the responsibilities of the VHDL wrapper
- All services and abstractions of the Hthreads system are now accessible to the HandelC core
- HWTI services encapsulated within HandelC library functions

Enabling Task Level Parallelism in HandelC

Agenda

- Problem Statement
- Background & Related Work
- HCthreads: Design and Implementation
- Support Utilities
- Results
- Conclusions and Future Work

Results

Simulator Results

- Both Solutions have same code in accelerated functions
- When introducing two or more units HCthreads has less overhead, better scheduling in HCthreads when compared to par invocations
- Some irregular results due to known bug in semaphore arbitration

Simulator Total Clock Cycles

par construct					
	1	2	3	4	5
QS	53,338	39,370	36,901	34,833	33,594
NQ	225,063	119,712	82,810	63,557	52,099
MM	9,888	5,373	4,083	5,241	4,676
HCthreads					
	1	2	3	4	5
QS	55,335	34,124	30,523	29,746	29,761
NQ	236,791	118,627	79,383	59,662	47,926
MMJ	9,961	5,423	4,313	4,313	4,313
MMD	9,947	5,402	4,107	4,107	4,084

J stands for joinable designs

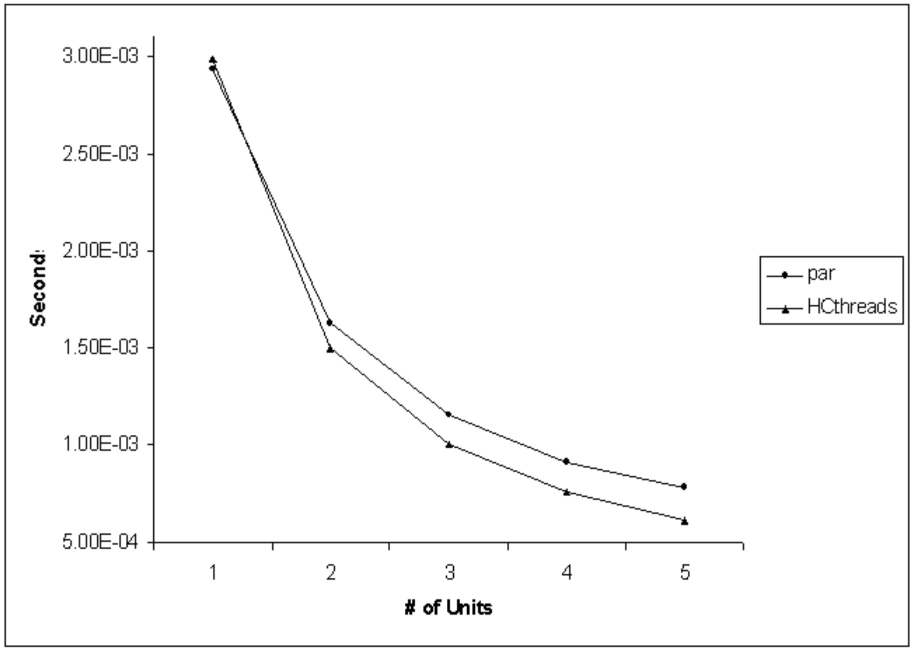
D stands for detached designs

Results

ML310 Results

- HCthreads produces better timing but requires additional resources

- Though no speedups could be achieved in memory intensive applications when having multiple units, HCthreads can be used to implement recursion with minimal overhead



NQueens Total Execution Time, ML310

Results

HandelC and Hthreads Integration

- The ML310 test cases incorporated the Hthreads support library with requests to HWTI services:
  - Load, Store
  - Push, Pop
  - Malloc, Free
  - Thread exit
- Current test setup does not employ all Hthreads services such as mutexes and thread operations
  - Separate test cases constructed to verify such cases
  - More testing is needed

Results

Enhancing the programming model

- For this section, no quantitative results to present but can state that HCthreads makes the coding of TLP in HandelC easier
- HCthreads provides free support for recursion even if no TLP is warranted

```
structAddr = pop();
size  = load(structAddr);

/* initialize input */
arg.startIndex = 0;
arg.endIndex   = size[16:0] - 1;

/* create initial thread */
hcthread_create(arg);

/* block till all terminate */
while(LiveThreads) delay;

push(ticks);
threadexit();
```

Agenda

- Problem Statement
- Background & Related Work
- HCthreads: Design and Implementation
- Support Utilities
- Results
- Conclusions and Future Work

Conclusions and Future Work

- HCthreads managed to combine ILP and TLP capabilities in HandelC enhancing the programming model
- HCthreads succeeded in providing the same speedups in computationally intensive applications with no overhead
- More testing is needed for the Hthreads support library
- Incorporate the globally distributed local memory offered by Hthreads into the HandelC address space to enhance the programming model further

Questions

?