Heterogeneous Work-stealing across CPU and DSP cores

Vivek Kumar¹, Alina Sbirlea¹, Ajay Jayaraj², Zoran Budimlić¹, Deepak Majeti¹, and Vivek Sarkar¹

¹ Rice University
² Texas Instruments
Accelerators in Top500

Lower power-to-performance ratio = $$

Source: http://www.slideshare.net/top500/top500-list-november-2014?related=1
Outline

• Background and motivation
• Our contributions
• Implementation
• Results
• Summary
if (! (GPGPU || CoProcessor) ) ?
if (! (GPGPU || CoProcessor) ) ?
if (! (GPGPU || CoProcessor) )?
DSP and HPC… Really?

- Traditionally DSPs existed in two different flavors
  - Fixed point operations
    - Integer arithmetic
    - Low cost
  - Floating point operations
    - Usage restricted to research, avionics
    - High cost

And Out of Thin Air a HPC Engine is Born…

And Out of Thin Air a HPC Engine is Born…

High capacity data transmission

Signal processing (e.g., MIMO)

High performance and accuracy

Floating point operations

And Out of Thin Air a HPC Engine is Born...

High capacity data transmission

Signal processing (e.g., MIMO)

High performance and accuracy

TI’s C66x DSPs – Integrated both fixed and floating point capabilities in the same DSP

November 2010

TI Keystone II SoC

- ARM A15
- C66x DSP

4MB ARM Shared L2

1MB L2 Per C66x Core

Multicore shared memory controller

6MB MSMC SRAM

DDR3

Total 2GB

Heterogeneous Work-stealing across CPU and DSP cores | Kumar et. al.
Existing Programming Model

- No special programming language required for DSP
  - Supports C language (TI’s C66x)
- Parallel programming using OpenMP
  - Stotzer et. al., OpenMP on the low-power TI Keystone-II ARM/DSP system-on-chip, IWOMP 2013
Existing Programming Model

• No special programming language required for DSP
  – Supports C language (TI’s C66x)

• Parallel programming using OpenMP
  – Stotzer et. al., OpenMP on the low-power TI Keystone-II ARM/DSP system-on-chip, IWOMP 2013
  – ARM dispatches OpenMP kernels to DSPs and wait for completion
    • Idle ARM cores
Contributions

✔ HC-K2H programming model
  Task parallel programming model for TI’s ARM+DSP SoC, which abstracts away hardware complexities from the user

✔ Hybrid work-stealing runtime
  That performs load balancing of tasks across ARM and DSP cores

✔ Detailed performance study
  Using standard work-stealing benchmarks

✔ Results
  That shows HC-K2H runtime can even outperform DSP only execution
HC-K2H Parallel Programming Model

// Task T₀ (Parent)
start_finish( );

end_finish( );

STMT3; //T₃
HC-K2H Parallel Programming Model

// Task T₀ (Parent)
start_finish( );

async (STMT1);    // T₁ (Child)
forasync (STMT2);  // T₂ (Child)
end_finish( );

STMT3;            // T₃
// Task T₀ (Parent)
start_finish( );

async (STMT1); //T₁ (Child)
forasync (STMT2); //T₂ (Child)
end_finish( );

STMT3; //T₃

//T₁ and T₂ are children of T₀.
// T₀, T₁, T₂, and T₃ are tasks.

async ↔ forasync
Compiling and Running

```c
main () {
}
```
Compiling and Running

```
main () { }

User main replaced using macros

main () {
    initialize_runtime ();
    _user_main ();
    finalize_runtime ();
}
_user_main () { }

ARM version

main () {
    initialize_runtime ();
    finalize_runtime ();
}
_user_main () { }

DSP version
```
Compiling and Running

$$\text{main}(\ ) \{ \}$$

User main replaced using macros

\begin{verbatim}
main ( ) {
    initialize_runtime ( )
    _user_main ( );
    finalize_runtime ( );
}
_user_main ( ) { }
\end{verbatim}

ARM version

\begin{verbatim}
main ( ) {
    initialize_runtime ( )
    finalize_runtime ( );
}
_user_main ( ) { }
\end{verbatim}

DSP version

\textbf{ARM version}

\textbf{DSP version}

\$ \text{ARM\_WORKERS=X\ DSP\_WORKERS=Y\ ./executable\ <command\ line\ args>}$
Hybrid Work-Stealing Implementation
HC-K2H Task Data-Structure

<table>
<thead>
<tr>
<th>ARMfunPtr</th>
<th>DSPfunPtr</th>
</tr>
</thead>
</table>

Valid at ARM only
Valid at DSP only

// ARM has access to symbol table manager
// which helps function pointer mapping between
// ARM and DSP

ARMfunPtr = lookup(DSPfunPtr);
DSPfunPtr = lookup(ARMfunPtr);
HC-K2H Task Data-Structure

<table>
<thead>
<tr>
<th>ARMfunPtr</th>
<th>DSPfunPtr</th>
<th>ARM_finish</th>
<th>DSP_finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid at ARM only</td>
<td>Valid at DSP only</td>
<td>Updated by ARM only</td>
<td>Updated by DSP only</td>
</tr>
</tbody>
</table>

// ARM has access to symbol table manager
// which helps function pointer mapping between
// ARM and DSP

ARMfunPtr = lookup(DSPfunPtr);
DSPfunPtr = lookup(ARMfunPtr);
HC-K2H Task Data-Structure

<table>
<thead>
<tr>
<th>ARMfunPtr</th>
<th>DSPfunPtr</th>
<th>ARM_finish</th>
<th>DSP_finish</th>
<th>args [SIZE]</th>
</tr>
</thead>
</table>

Valid at ARM only
Valid at DSP only
Updated by ARM only
Updated by DSP only
Function arguments packed in an array (current limitation in HC-K2H)

// ARM has access to symbol table manager
// which helps function pointer mapping between
// ARM and DSP

ARMfunPtr = lookup(DSPfunPtr);

DSPfunPtr = lookup(ARMfunPtr);
## Work-stealing Properties

<table>
<thead>
<tr>
<th></th>
<th>WS Data-structure</th>
<th>Task synchronization (task counter at each finish scope)</th>
<th>Push</th>
<th>Pop</th>
<th>Steal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>Software deques at each core</td>
<td>Atomic operations</td>
<td>Tail</td>
<td>Tail</td>
<td>Head</td>
</tr>
</tbody>
</table>
## Work-stealing Properties

<table>
<thead>
<tr>
<th></th>
<th>WS Data-structure</th>
<th>Task synchronization (task counter at each finish scope)</th>
<th>Push</th>
<th>Pop</th>
<th>Steal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>Software deques at each core</td>
<td>Atomic operations</td>
<td>Tail</td>
<td>Tail</td>
<td>Head</td>
</tr>
<tr>
<td>DSP</td>
<td>Hardware queues at each core</td>
<td>Using single hardware semaphore (total hardware semaphores = 32 only)</td>
<td>Head or tail</td>
<td>Tail (only)</td>
<td>== Pop</td>
</tr>
</tbody>
</table>
Hybrid Work-Stealing Design
Hybrid Work-Stealing Design
Hybrid Work-Stealing Design

Steals within ARM cores

Push
Pop

Head
Tail
if (Idle_DSPs) {
    task->DSPfunPtr = lookup(ARMfunPtr);
    increment(task->ARM_finish);
    push(task, SQ1);
}
Hybrid Work-Stealing Design

if (Idle_DSPs) {
    task->DSPfunPtr = lookup(ARMfunPtr);
    increment(task->ARM_finish);
    push(task, SQ1);
}

Steal(task, SQ1); start_new_DSP_finish();
Hybrid Work-Stealing Design

Steals within ARM cores

```c
if (Idle_DSPs) {
    task->DSPfunPtr = lookup(ARMfunPtr);
    increment(task->ARM_finish);
    push(task, SQ1);
}
```

Steals within DSP cores

```c
Steal(task, SQ1); start_new_DSP_finish();
```
Hybrid Work-Stealing Design

if (Idle_DSPs) {
    task->DSPfunPtr = lookup(ARMfunPtr);
    increment(task->ARM_finish);
    push(task, SQ1);
}

Steals within ARM cores

Steals within DSP cores

Steal(task, SQ1); start_new_DSP_finish(
end_DSP_new_finish(
push(task, SQ3);
Heterogeneous Work-stealing across CPU and DSP cores | Kumar et. al.

Steal(task, SQ3); decrement(task->ARM_finish);

Steals within ARM cores

if (Idle_DSPs) {
    task->DSPfunPtr = lookup(ARMfunPtr);
    increment(task->ARM_finish);
    push(task, SQ1);
}

start_new_DSP_finish( ); push(task, SQ3);

Steal(task, SQ3); decrement(task->ARM_finish);

Steals within DSP cores

Steal(task, SQ1); start_new_DSP_finish( );

end_DSP_new_finish( ); push(task, SQ3);
Hybrid Work-Stealing Design

Steal(task, DSPs); task->ARMfunPtr = lookup(DSPfunPtr); start_new_ARM_finish();

Steal(task, SQ3); decrement(task->ARM_finish);

Steals within ARM cores

if (Idle_DSPs) {
    task->DSPfunPtr = lookup(ARMfunPtr);
    increment(task->ARM_finish);
    push(task, SQ1);
}

end_DSP_new_finish(); push(task, SQ3);

Steal(task, SQ1); start_new_DSP_finish();

Steals within DSP cores
Hybrid Work-Stealing Design

Steal(task, DSPs); task->ARMfunPtr = lookup(DSPfunPtr); start_new_ARM_finish();

Steal(task, SQ3); decrement(task->ARM_finish);

Steals within ARM cores

Steal(task, DSPs); task->ARMfunPtr = lookup(DSPfunPtr); start_new_ARM_finish();

Steal(task, SQ3); decrement(task->ARM_finish);

Steals within DSP cores

if (Idle_DSPs) {
  task->DSPfunPtr = lookup(ARMfunPtr);
  increment(task->ARM_finish);
  push(task, SQ1);
}
end_ARM_new_finish(); push(task, SQ2);

Steal(task, SQ1); start_new_DSP_finish();

Steal(task, DSPs); task->ARMfunPtr = lookup(DSPfunPtr); start_new_ARM_finish();

Steal(task, SQ3); decrement(task->ARM_finish);
Hybrid Work-Stealing Design

Steal(task, DSPs); task->ARMfunPtr = lookup(DSPfunPtr); start_new_ARM_finish();

Steal(task, SQ3); decrement(task->ARM_finish);

Steal(task, SQ3); decrement(task->ARM_finish);

if (Idle_DSPs) {
    task->DSPfunPtr = lookup(ARMfunPtr);
    increment(task->ARM_finish);
    push(task, SQ1);
}

end_ARM_new_finish(); push(task, SQ2);

Steal(task, DSPs); task->ARMfunPtr = lookup(DSPfunPtr); start_new_ARM_finish();

end_DSP_new_finish(); push(task, SQ3);

Steal(task, SQ2); decrement(task->DSP_finish);

Steal(task, DSPs); task->ARMfunPtr = lookup(DSPfunPtr); start_new_ARM_finish();

end_DSP_new_finish(); push(task, SQ3);

Steal(task, SQ2); decrement(task->DSP_finish)
Experimental Evaluation

- Work-stealing performance
  - DSP only \textit{v/s} hybrid

![Graph showing work-stealing performance](image)

- DSP only work-stealing performance
- Hybrid work-stealing performance
Experimental Evaluation

- Work-stealing performance
  - Understanding anomalies
Summary

• Current Work
  – Parallel programming model for TI Keystone II SoC
    • Abstracts away hardware complexities
  – Hybrid work-stealing across ARM and DSP
  – Detailed experimental evaluation
    • Optimal load balancing, even outperforming DSP only executions

• Future work
  – Shared memory allocations from DSP cores
  – More benchmarks
Backup Slides
HC-K2H Parallel Programming Constructs

• Asynchronous tasks
  
  // Execute only on DSP cores
  asyncDSP (void* funPtr, void* args, int args_size);

  // Can execute on ARM or DSP core
  async (void* funPtr, void* args, int args_size);

  // Can execute on ARM or DSP core
  forasync (void* funPtr, void* args, int args_size, int loop_dimension, loop_domain_t* domain, int mode);

• Synchronization over asynchronous tasks
  
  // start a finish scope
  // variable argument function call
  start_finish (int total_shared_vars, shared_pointer_i, ...);

  // end a finish scope
  end_finish ( );
HClib Programming Model

```c
#include "hclib.h"
main () {
    arm_init (N);
    parallel_sum ( );
}

void arm_init (int N) {
    size = N;
    A = malloc(N * sizeof(int));
    initialize_A( );
    /* similarly for B and C */
}

void parallel_sum ( ) {
    loop_domain_t loop = {low, high, stride, tile};
    start_finish( );
    forasync(kernel, NULL, 0, 1, &loop, WS_RECURSION);
    end_finish( );
}

void kernel(void* args, int i) {
    C[i] = A[i] + B[i];
}
```

Parallel array addition in HClib

More information: http://habanero-rice.github.io/hclib/
#include "hc-k2h.h"
main () {
    arm_init (N);
    parallel_sum ();
}

void arm_init (int N) {
    size = N;
    A = ws_malloc(N * sizeof(int));
    initialize_A();

    /* similarly for B and C */
}

void parallel_sum () {
    loop_domain_t loop = {low, high, stride, tile};
    start_finish();
    forasync(kernel, NULL, 0, 1, &loop, WS_RECURSION);
    end_finish();
}

void kernel(void* args, int i) {
    C[i] = A[i] + B[i];
}

Parallel array addition in HC-K2H
#include "hc-k2h.h"
main () {
    arm_init (N);
    parallel_sum ( );
}

void arm_init (int N) {
    size = N;
    A = ws_malloc(N * sizeof(int));
    initialize_A( );
    ws_cacheWbInv (A);
    /* similarly for B and C */
}

void parallel_sum ( ) {
    loop_domain_t loop = {low, high, stride, tile};
    start_finish( );
    forasync(kernel, NULL, 0, 1, &loop, WS_RECURSION);
    end_finish( );
}

void kernel(void* args, int i) {
    C[i] = A[i] + B[i];
}

Parallel array addition in HC-K2H
#include “hc-k2h.h”
main ( ) {
    arm_init (N);
    dsp_init ( );
    parallel_sum ( );
}

void arm_init (int N) {
    size = N;
    A = ws_malloc(N * sizeof(int));
    initialize_A( );
    ws_cacheWbInv (A);
    /* similarly for B and C */
}

void dsp_init ( ) {
    /* pointer translation */
    int in[ ] = {N,
                 ws_dspPtr (A),
                 ws_dspPtr (B),
                 ws_dspPtr (C)};
    start_finish (0);
    /* DSP only async task */
    /* dsp_init_func( ) à A = (int*) in [1]; ………… */
    asyncDSP (dsp_init_func, in, sizeof(in));
    end_finish ( );
}

void parallel_sum ( ) {
    loop_domain_t loop = {low, high, stride, tile};
    start_finish ( );
    forasync (kernel, NULL, 0, 1, &loop, WS_RECURSION);
    end_finish ( );
}

void kernel(void* args, int i) {
    C[i] = A[i] + B[i];
}

Parallel array addition in HC-K2H
```c
#include "hc-k2h.h"
main ( ) {
    arm_init (N);
    dsp_init ( );
    parallel_sum ( );
}

void arm_init (int N) {
    size = N;
    A = ws_malloc(N * sizeof(int));
    initialize_A( );
    ws_cacheWbInv (A);
    /* similarly for B and C */
}

void dsp_init ( ) {
    /* pointer translation */
    int in[ ] = {N, ws_dspPtr(A), ws_dspPtr(B), ws_dspPtr(C)};
    start_finish (0);
    /* DSP only async task */
    /* dsp_init_func( ) \rightarrow A = (int*) in [1]; .......... */
    asyncDSP (dsp_init_func, in, sizeof(in));
    end_finish ( );
}

void parallel_sum ( ) {
    loop_domain_t loop = {low, high, stride, tile};
    start_finish(
    forasync(kernel, NULL, 0, 1, &loop, WS_RECURSION);
    end_finish( );
}

void kernel(void* args, int i) {
    C[i] = A[i] + B[i];
}
```

Parallel array addition in HC-K2H
HC-K2H Programming Model

```c
#include "hc-k2h.h"
main ( ) {
    arm_init (N);
    dsp_init ( );
    parallel_sum ( );
}

void arm_init (int N) {
    size = N;
    A = ws_malloc(N * sizeof(int));
    initialize_A( );
    ws_cacheWbInv (A);
    /* similarly for B and C */
}

void dsp_init ( ) {
    int in[ ] = {N, ws_dspPtr(A), ws_dspPtr(B), ws_dspPtr(C)};
    start_finish (0);
    /* DSP only async task */
    /* dsp_init_func( ) à A = (int*) in [1]; .......... */
    asyncDSP (dsp_init_func, in, sizeof(in));
    end_finish ( );
}

void parallel_sum ( ) {
    loop_domain_t loop = {low, high, stride, tile};
    ws_args_t t1 = {C}; /* result array */
    start_finish(1, &t1);
    forasync(kernel, NULL, 0, 1, &loop, WS_RECURSION);
    end_finish( );
}

void kernel(void* args, int i) {
    C[i] = A[i] + B[i];
}
```

Parallel array addition in HC-K2H
Avoiding False Sharing

- ARM cache line
  - 64 bytes
- DSP cache line
  - 128 bytes

Allocate writable shared buffers with sizes in multiple of 128 bytes

// Specifying information on for-loop in forasync task
loop_domain_t loop_info = { lowBound, highBound, stride, tile_size };  

uint32_t writable_shared_array[1024]; /* Option: use tile_size = 32 */
Multicore ARM+DSP TI’s Keystone-II SoC

• Software configuration
  – ARM
    • Standard Linux
  – DSP
    • Custom real-time O.S. called as SYS/BIOS™
      – Support for inter processor communication
      – Custom runtime library to support thread management, scheduling and synchronization
      – Supports C99 C language
      – No pthread libraries or GCC built-in atomic functions