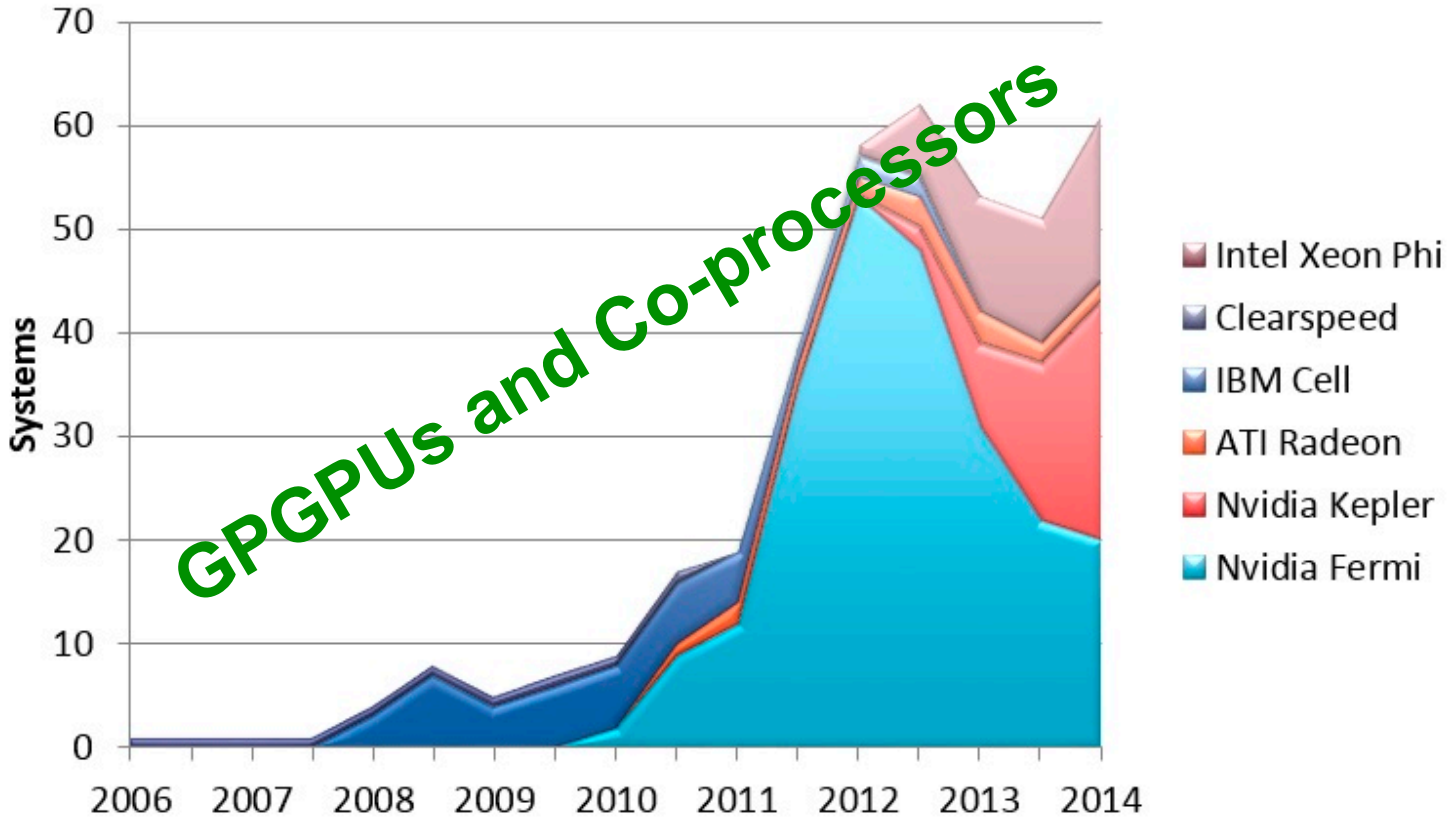
Heterogeneous Work-stealing across CPU and DSP cores

Vivek Kumar[1], Alina Sbîrlea[1], Ajay Jayaraj[2],
Zoran Budimlić[1], Deepak Majeti[1], and Vivek Sarkar[1]

[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

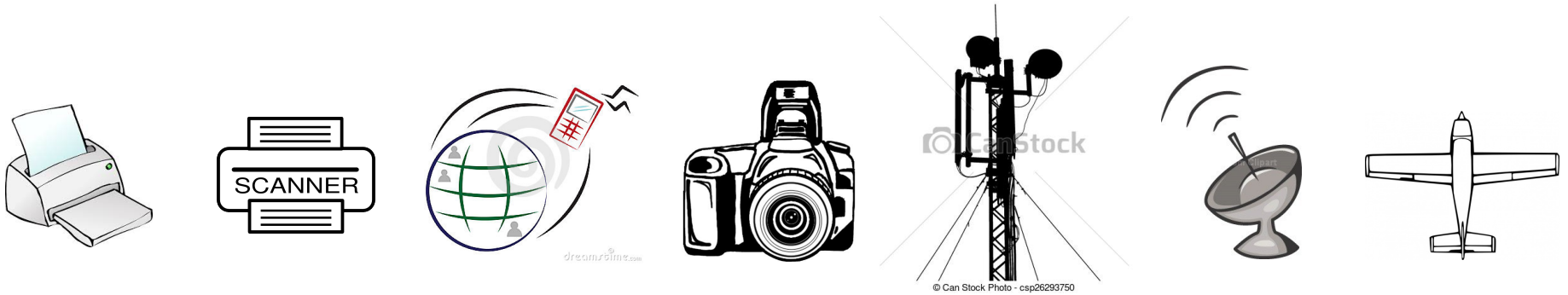
RICE TEXAS INSTRUMENTS

Outline

- Background and motivation
- Our contributions
- Implementation
- Results
- Summary

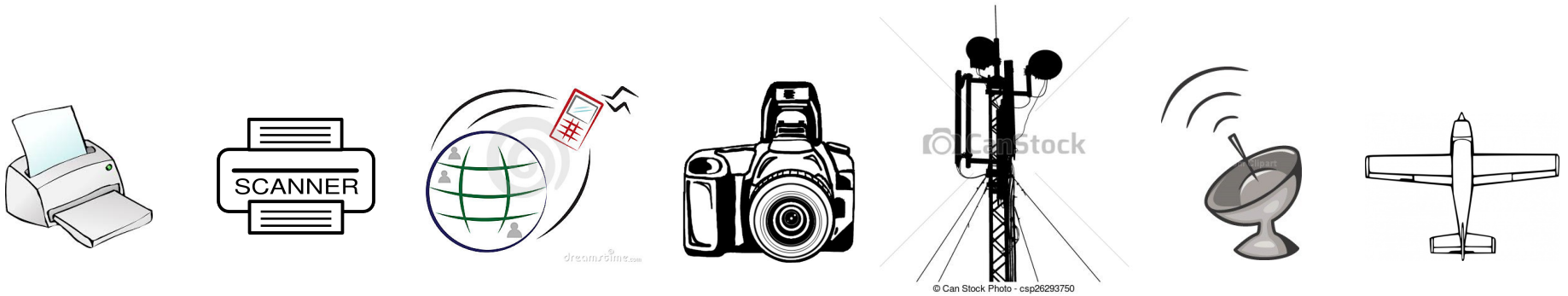
if (! (GPGPU || CoProcessor)) ?

if (! (GPGPU || CoProcessor)) ?

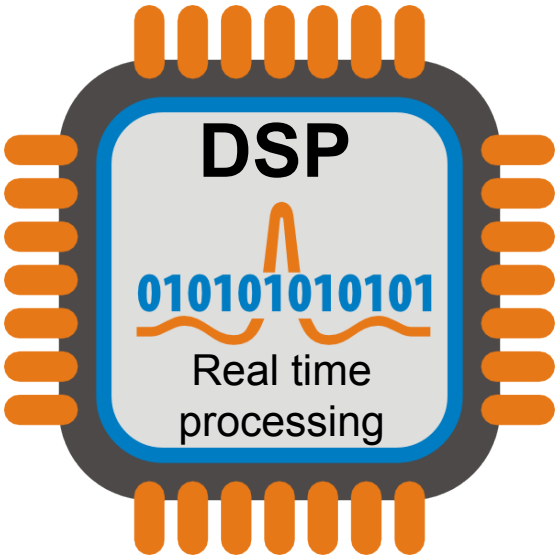


RICE TEXAS INSTRUMENTS

if (! (GPGPU || CoProcessor)) ?



Continuous analog signals as electric pules of varying amplitude



Digital signals in binary format

RICE TEXAS INSTRUMENTS

DSP and HPC… Really ?

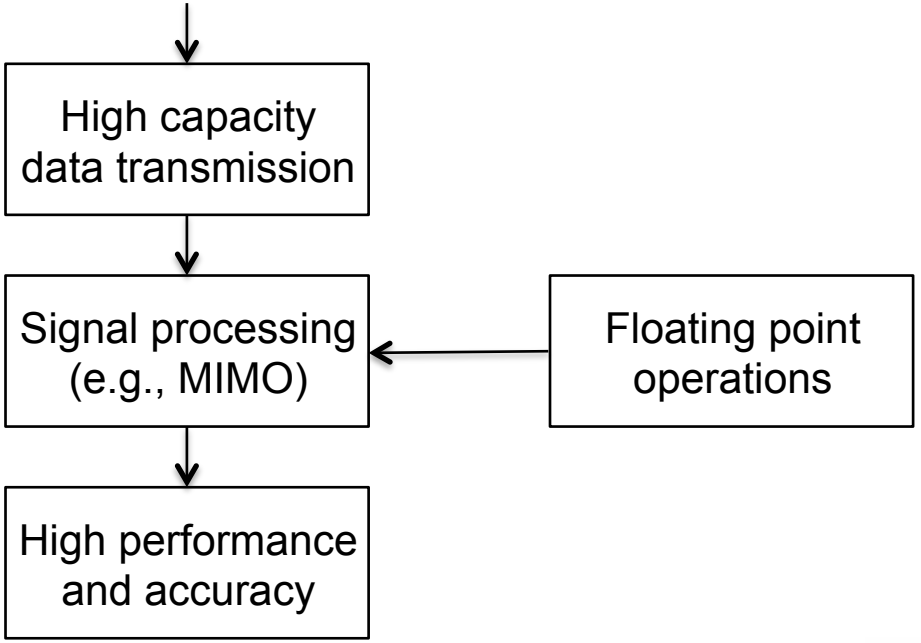
- Traditionally DSPs existed in two different flavors
  - Fixed point operations
    - Integer arithmetic
    - Low cost
  - Floating point operations
    - Usage restricted to research, avionics
    - High cost

Source: http://www.ti.com/lit/wp/spry061/spry061.pdf

And Out of Thin Air a HPC Engine is Born…

Source: http://www.ti.com/lit/wp/spry147/spry147.pdf

And Out of Thin Air a HPC Engine is Born…



Source: http://www.ti.com/lit/wp/spry147/spry147.pdf

RICE TEXAS INSTRUMENTS

And Out of Thin Air a HPC Engine is Born…

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

November 2010

High capacity data transmission

Signal processing (e.g., MIMO)

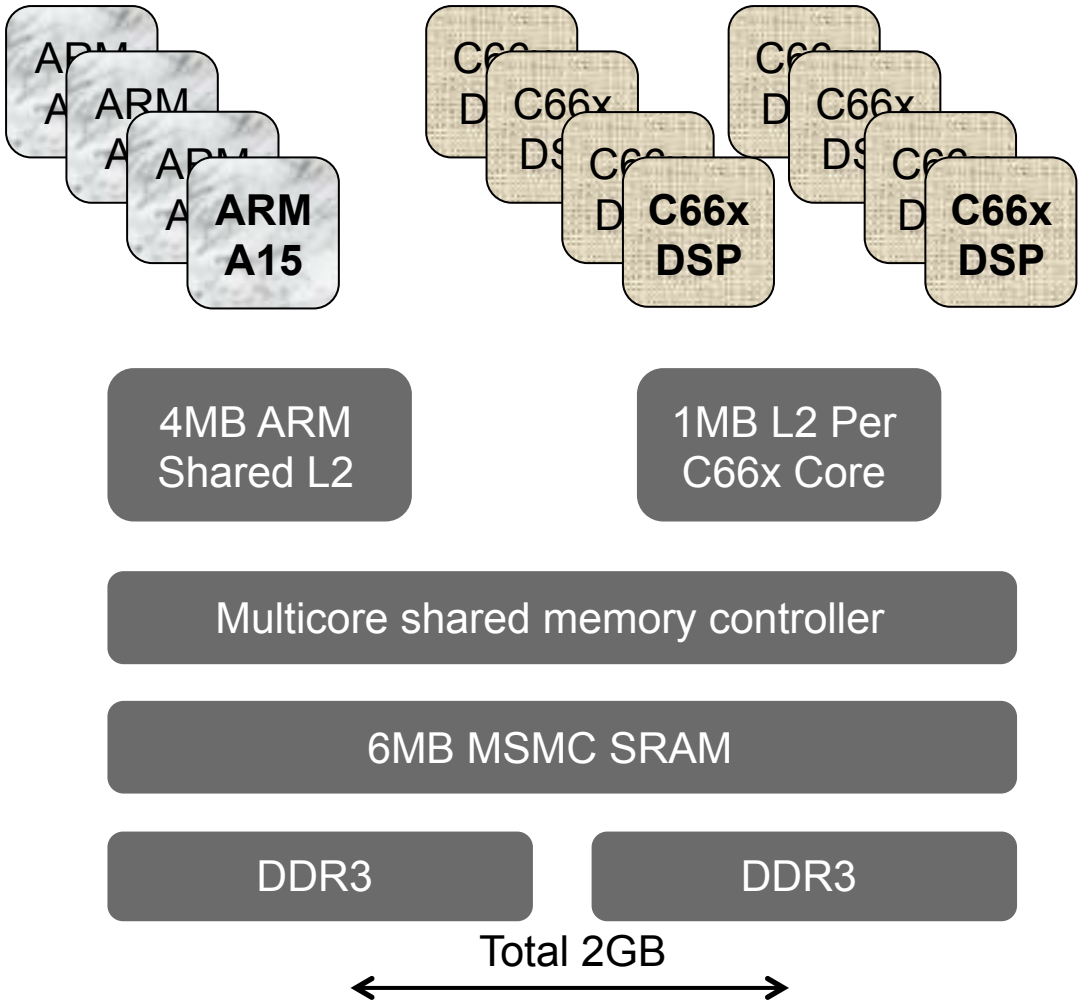
Floating point operations

High performance and accuracy

Source: http://www.ti.com/lit/wp/spry147/spry147.pdf

RICE Texas Instruments

TI Keystone II SoC



RICE TEXAS INSTRUMENTS

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

RICE TEXAS INSTRUMENTS

HC-K2H Parallel Programming Model

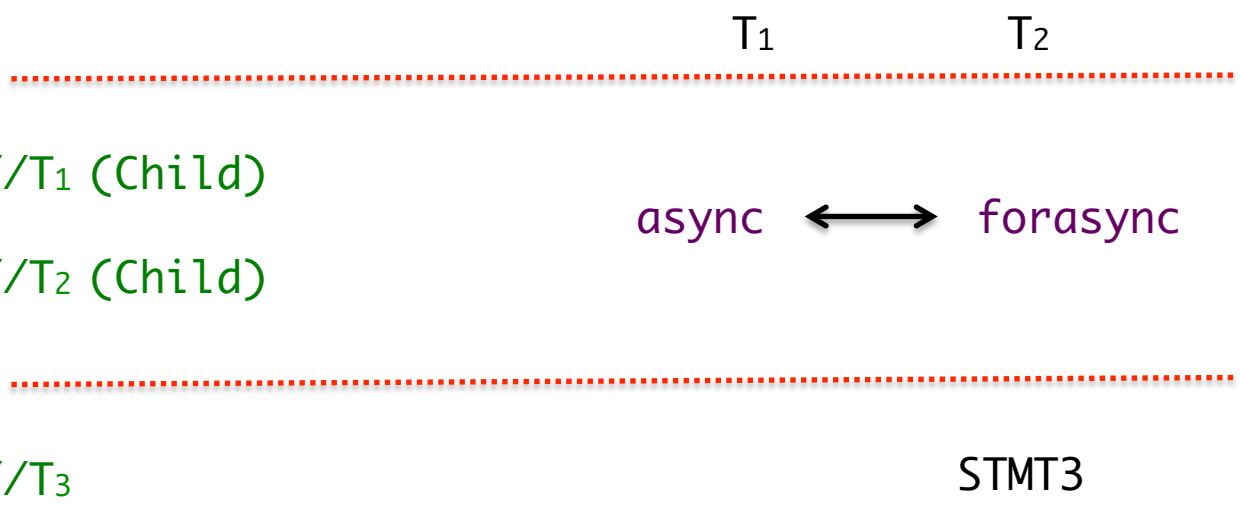
```
// Task T0 (Parent)
start_finish( );




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
```

HC-K2H Parallel Programming Model

```
// Task T0 (Parent)
start_finish( );

async (STMT1);     //T1 (Child)

forasync (STMT2); //T2 (Child)

end_finish( );

STMT3;             //T3
```



RICE TEXAS INSTRUMENTS

Compiling and Running

```
main ( ) { }
```

Compiling and Running

```
main ( ) { }
```


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

ARM version

```
main ( ) {
  initialize_runtime ( )

  finalize_runtime ( );
}
_user_main ( ) { }
```

DSP version

Compiling and Running

```
main ( ) { }
```

User main replaced using macros

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

ARM version

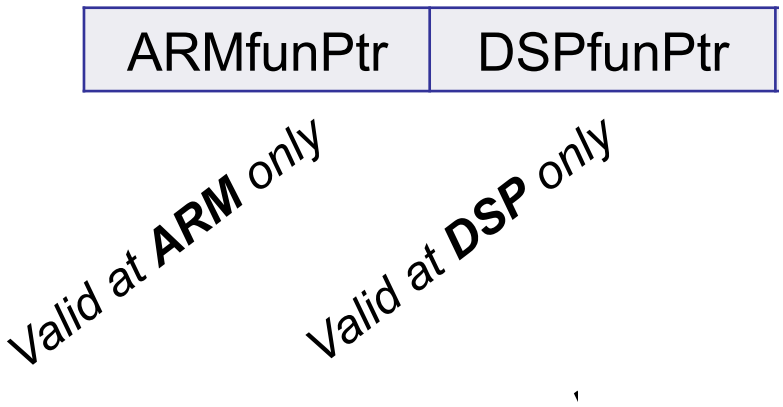
```
main ( ) {
  initialize_runtime ( )

  finalize_runtime ( );
}
_user_main ( ) { }
```

DSP version

$ ARM_WORKERS=**X** DSP_WORKERS=**Y** ./executable <command line args>

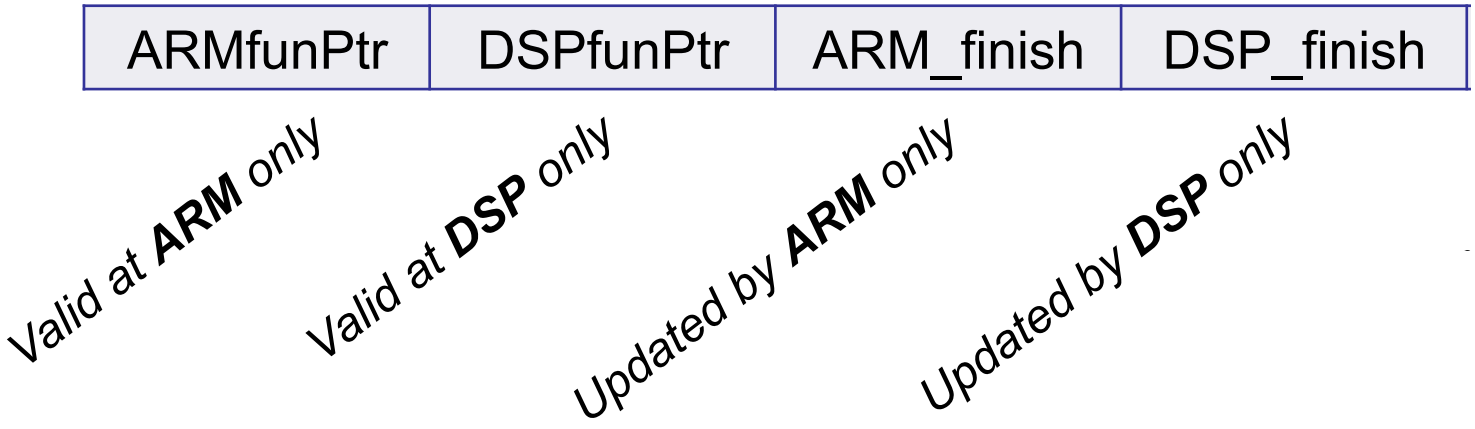
Hybrid Work-Stealing Implementation

HC-K2H Task Data-Structure



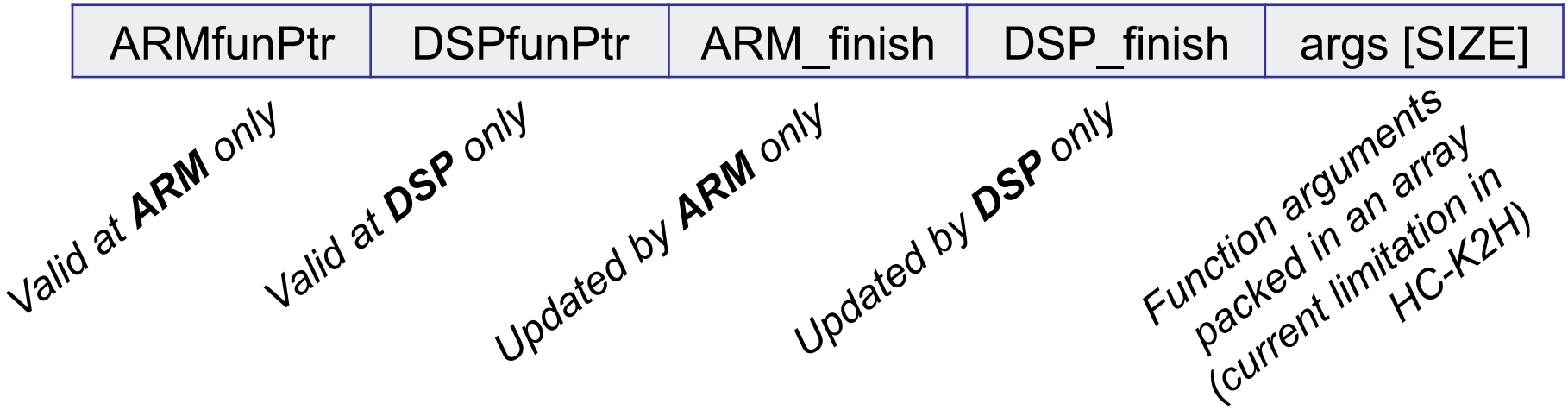

```
// ARM has access to symbol table manager
// which helps function pointer mapping between
// ARM and DSP

ARMfunPtr = lookup(DSPfunPtr);

DSPfunPtr = lookup(ARMfunPtr);
```

HC-K2H Task Data-Structure




```
// 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




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

ARMfunPtr = lookup(DSPfunPtr);

DSPfunPtr = lookup(ARMfunPtr);
```


RICE TEXAS INSTRUMENTS

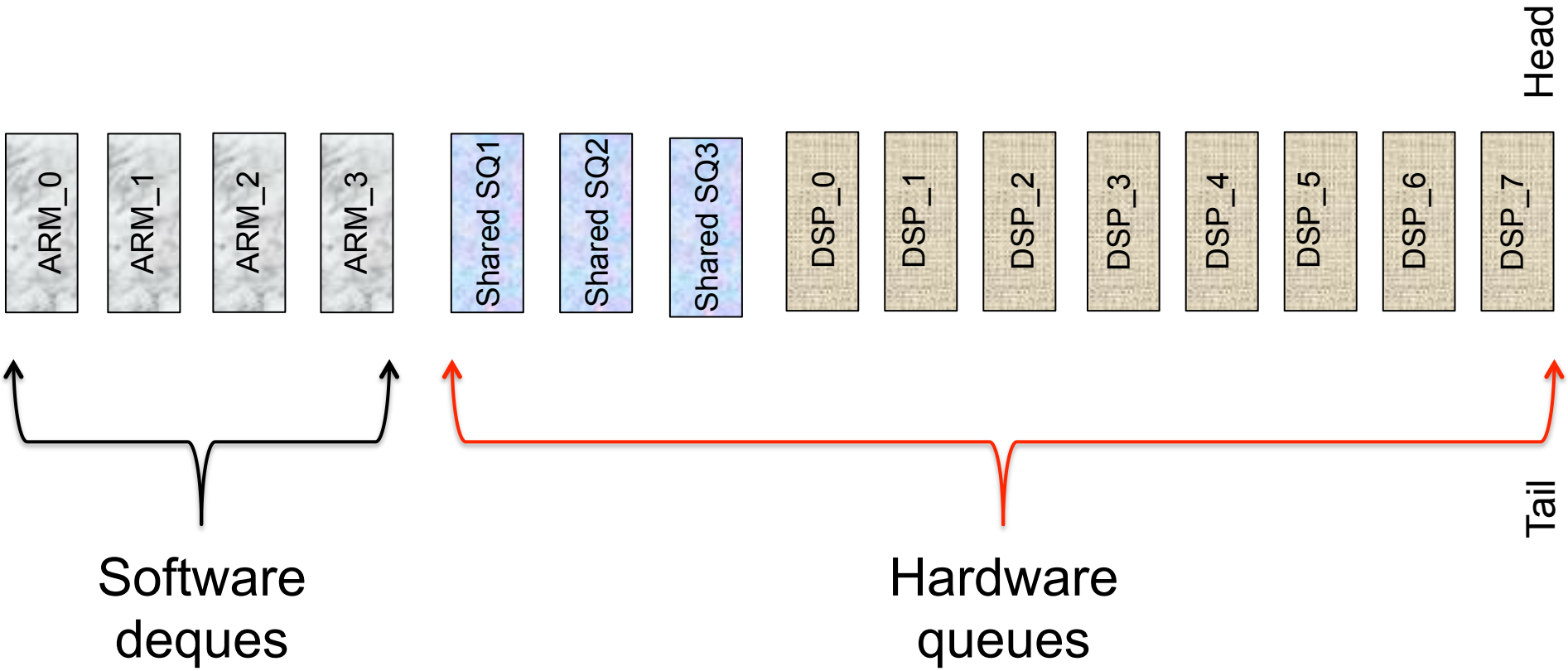
Work-stealing Properties

	WS Data-structure	Task synchronization (task counter at each finish scope)	Push	Pop	Steal
ARM	Software deques at each core	Atomic operations	Tail	Tail	Head

Work-stealing Properties

	WS Data-structure	**Task synchronization (task counter at each finish scope)**	**Push**	**Pop**	**Steal**
ARM	Software deques at each core	Atomic operations	Tail	Tail	Head
DSP	Hardware queues at each core	Using single hardware semaphore (total hardware semaphores = 32 only)	Head or tail	Tail (only)	== Pop

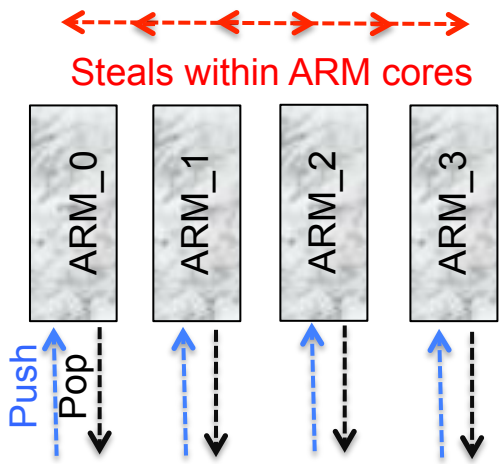
Hybrid Work-Stealing Design



RICE TEXAS INSTRUMENTS

Hybrid Work-Stealing Design

Hybrid Work-Stealing Design

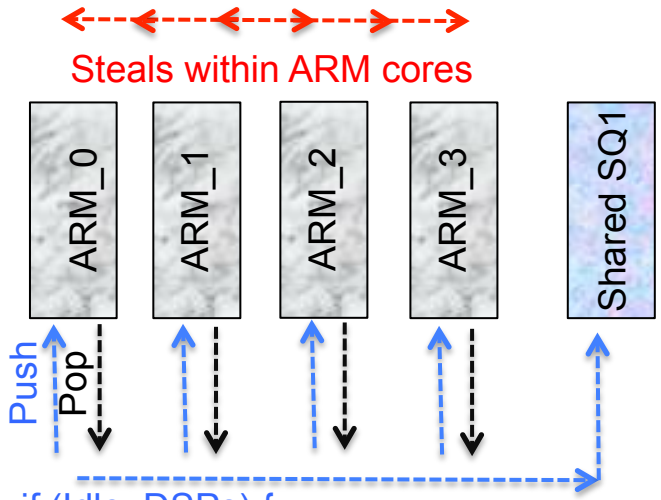


Head

Tail

RICE TEXAS INSTRUMENTS

Hybrid Work-Stealing Design



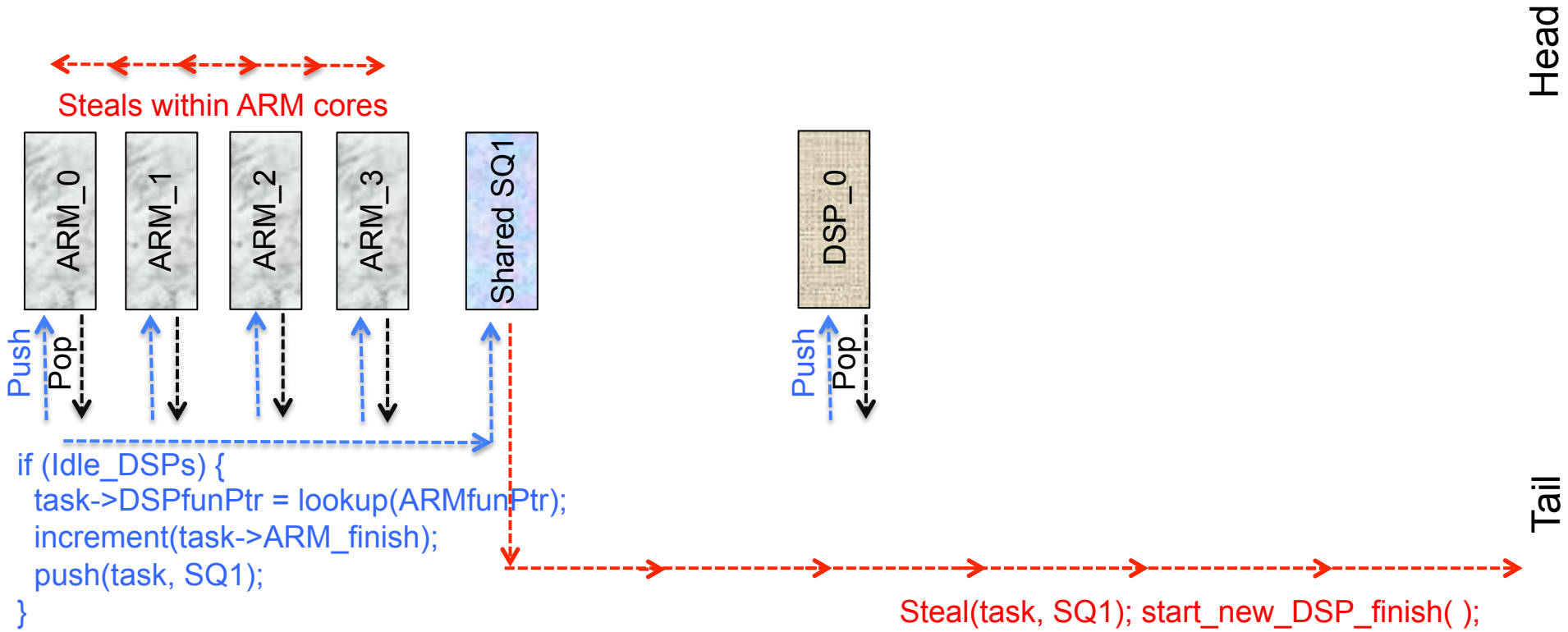

```
if (Idle_DSPs) {
  task->DSPfunPtr = lookup(ARMfunPtr);
  increment(task->ARM_finish);
  push(task, SQ1);
}
```

Head

Tail

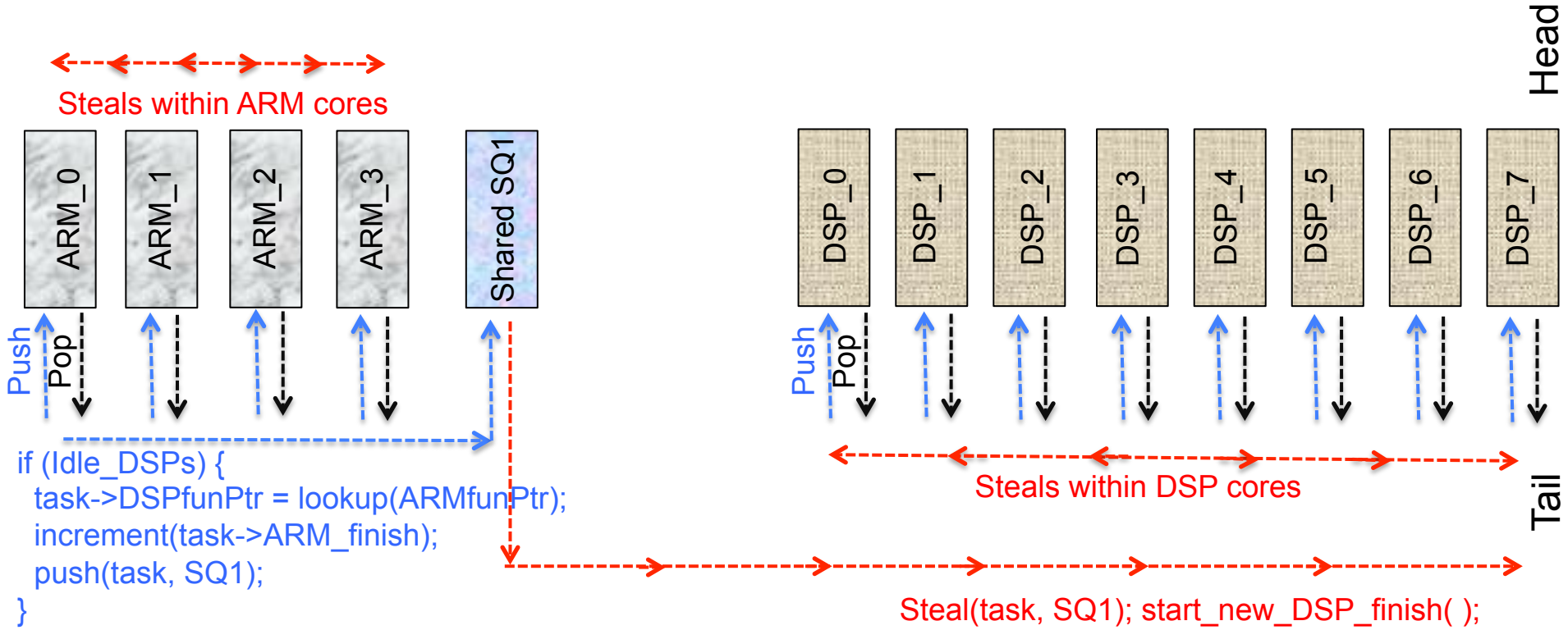
RICE TEXAS INSTRUMENTS

Hybrid Work-Stealing Design



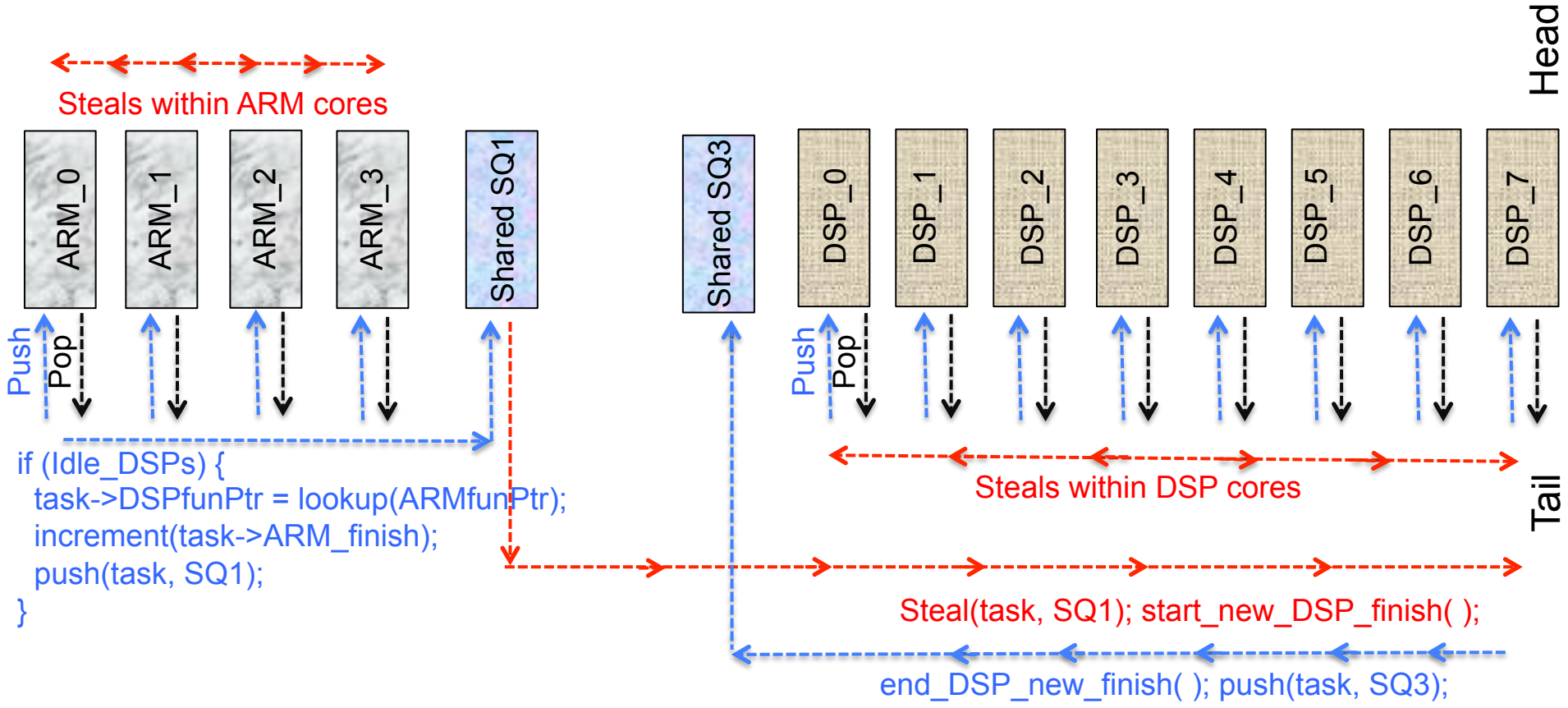
RICE TEXAS INSTRUMENTS

Hybrid Work-Stealing Design



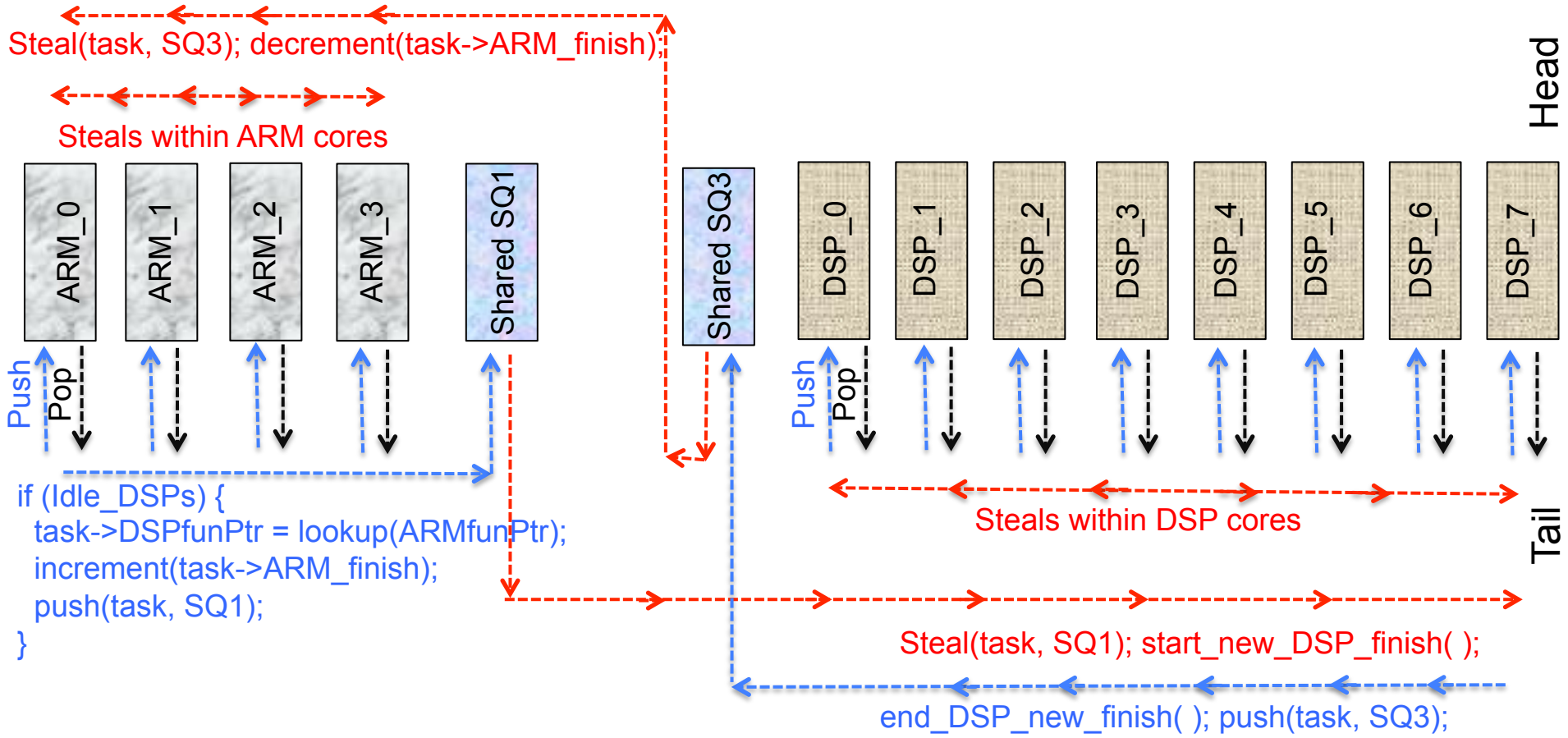
RICE TEXAS INSTRUMENTS

Hybrid Work-Stealing Design



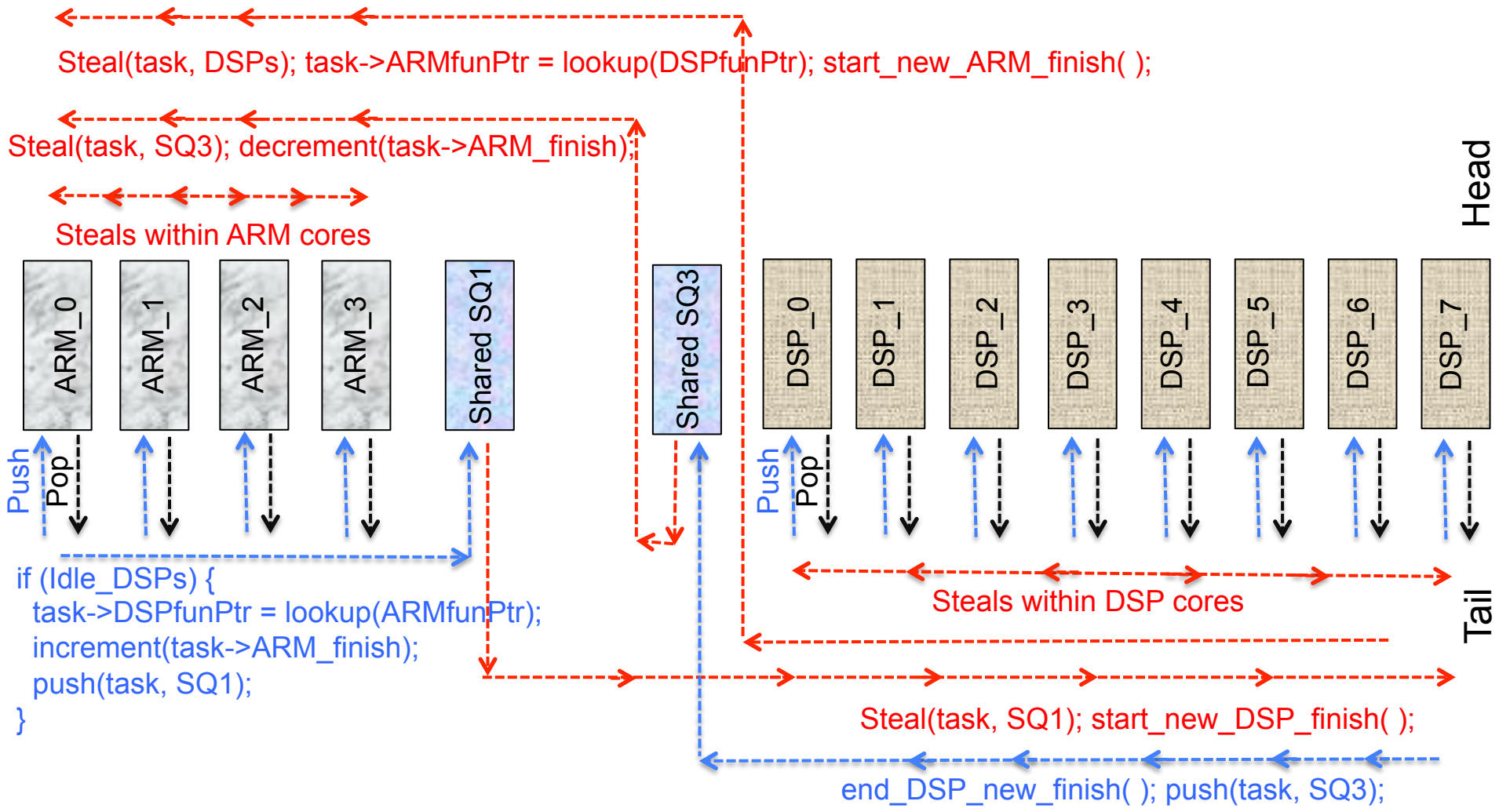
RICE TEXAS INSTRUMENTS

Hybrid Work-Stealing Design



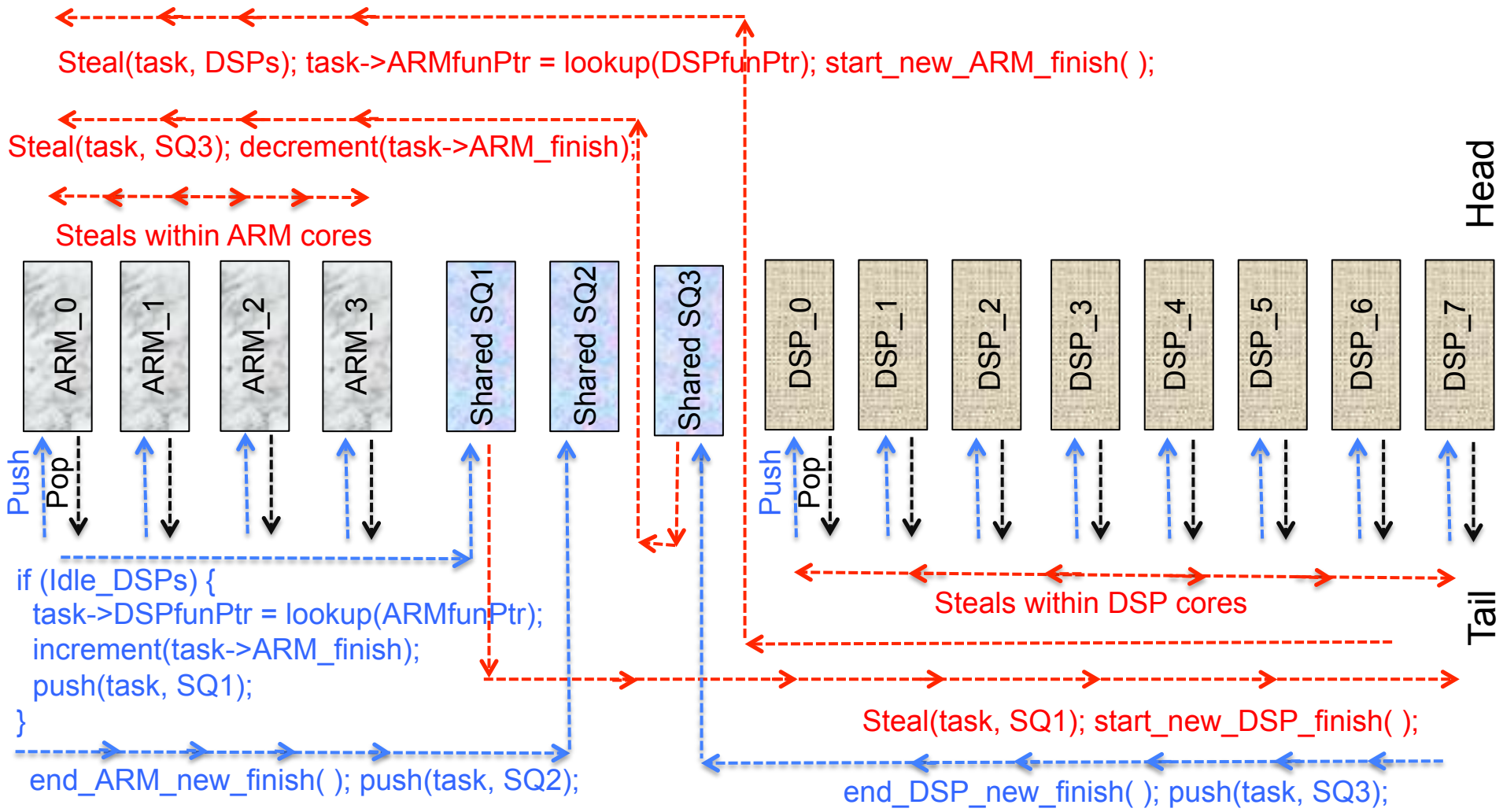
RICE TEXAS INSTRUMENTS

Hybrid Work-Stealing Design



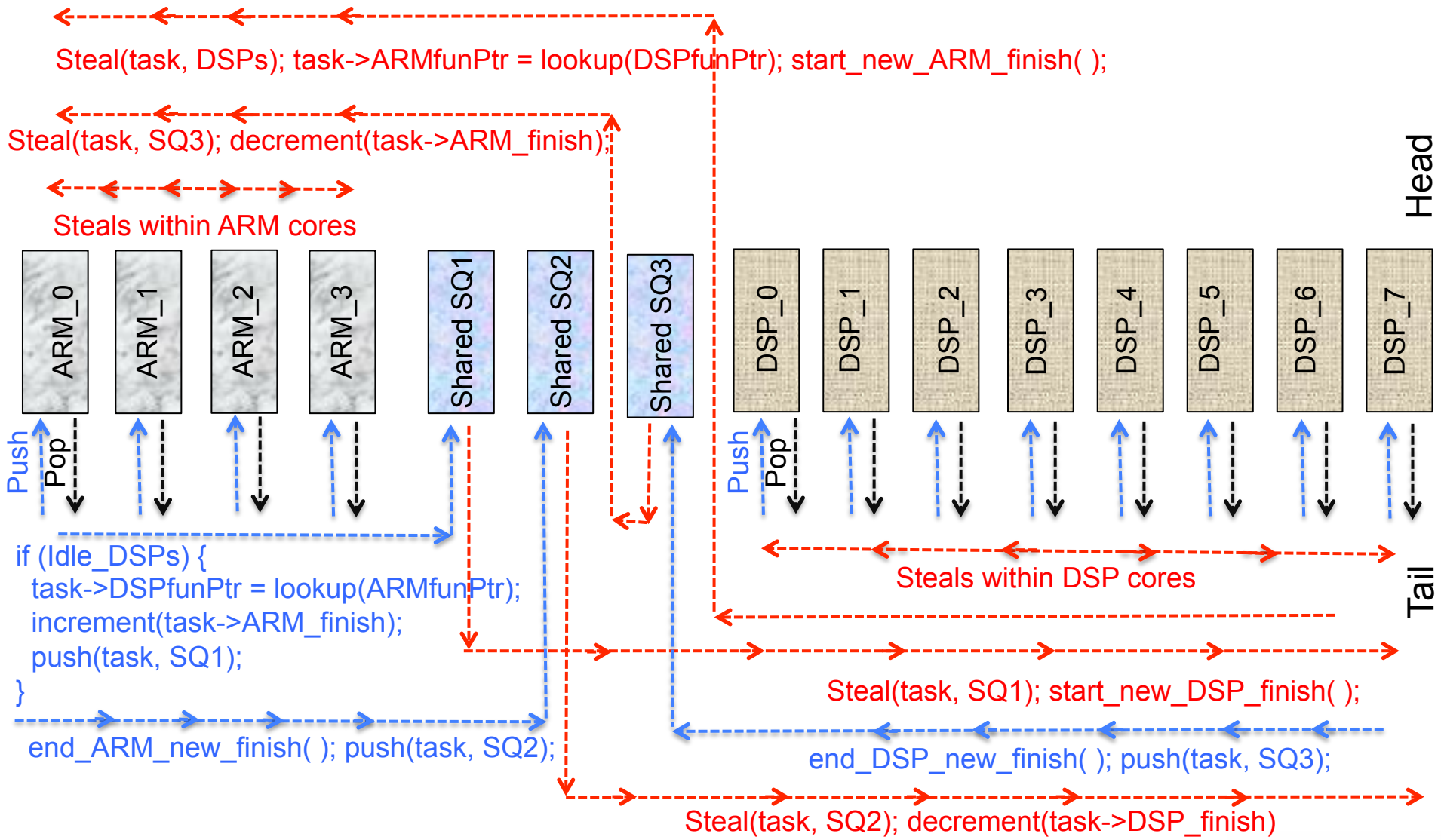
RICE TEXAS INSTRUMENTS

Hybrid Work-Stealing Design



RICE TEXAS INSTRUMENTS

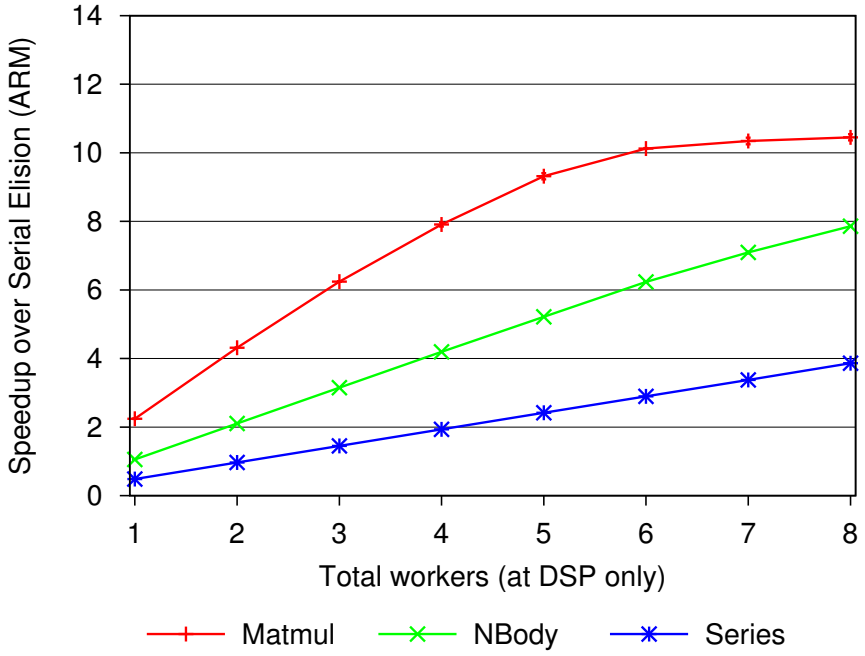
Hybrid Work-Stealing Design



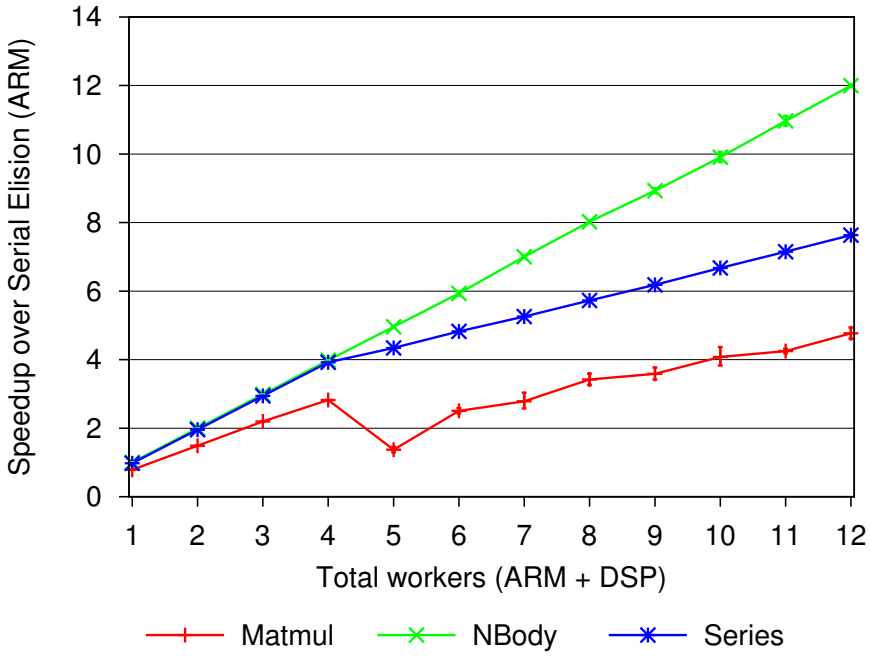
RICE TEXAS INSTRUMENTS

Experimental Evaluation

- Work-stealing performance
  - DSP **only** v/s hybrid



DSP only work-stealing performance

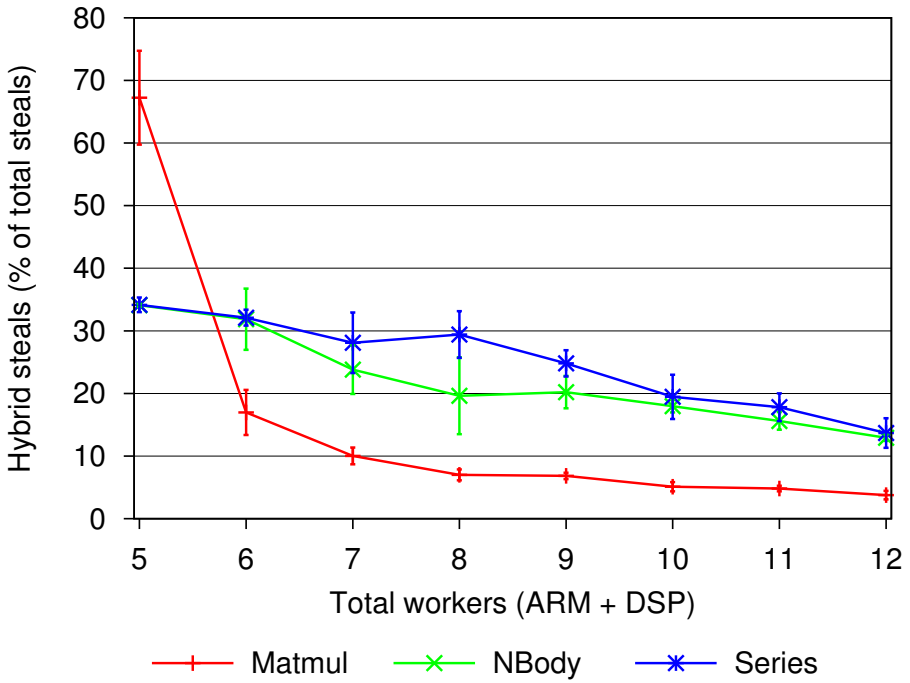


Hybrid work-stealing performance

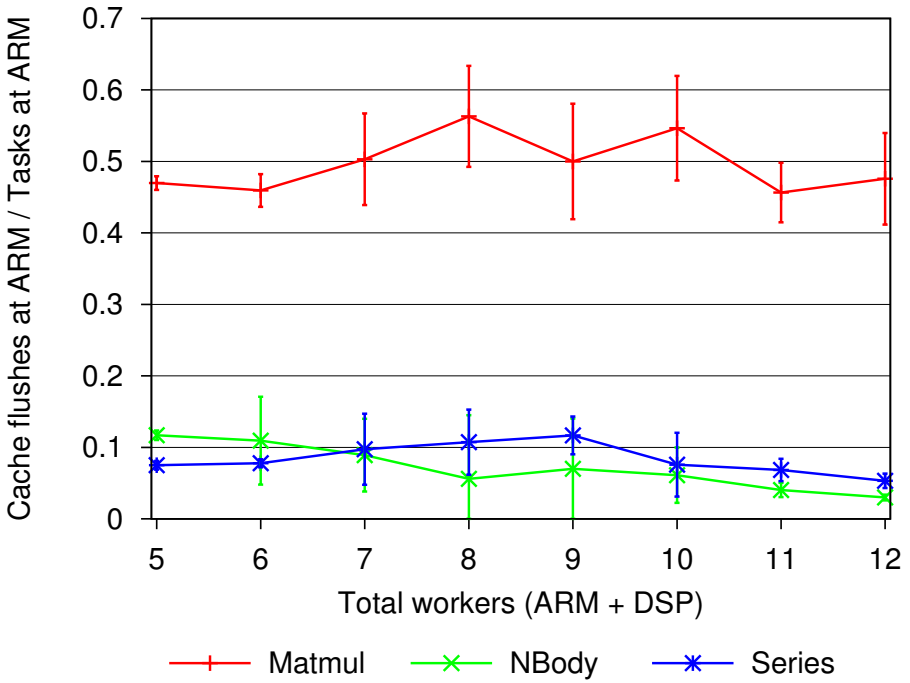
RICE TEXAS INSTRUMENTS

Experimental Evaluation

- Work-stealing performance
  - Understanding anomalies



Total hybrid steals (% of total steals)



Ratio of cache flushes at ARM to total async tasks generated at ARM

RICE TEXAS INSTRUMENTS

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

```
#include ”hclib.h”
int size, *A, *B, *C;
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/

HC-K2H Programming Model

```
#include “hc-k2h.h”
int size, *A, *B, *C;
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 HC-K2H

HC-K2H Programming Model

```
#include “hc-k2h.h”
int size, *A, *B, *C;
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

HC-K2H Programming Model

```
#include “hc-k2h.h”
int size, *A, *B, *C;
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 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

```
#include “hc-k2h.h”
int size, *A, *B, *C;
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( ) → 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

```
#include “hc-k2h.h”
int size, *A, *B, *C;
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( ) → 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