Structured Parallelism for High Productivity and High Performance

Vivek Kumar

Assistant Professor

Department of Computer Science & Engineering

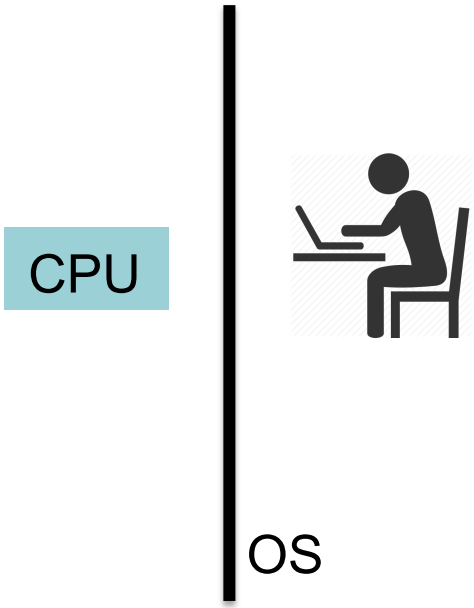
IIIT Delhi

Outline

- Background
- Structured Parallelism using Async-Finish Programming Model
- Load Balancing using Work-Stealing
- High Productivity using Async-Finish
- High Performance using Async-Finish
  - Heterogeneous Computing over MPSoC
  - Distributed Computing
  - JVM Supported Work-Stealing
- Future Research

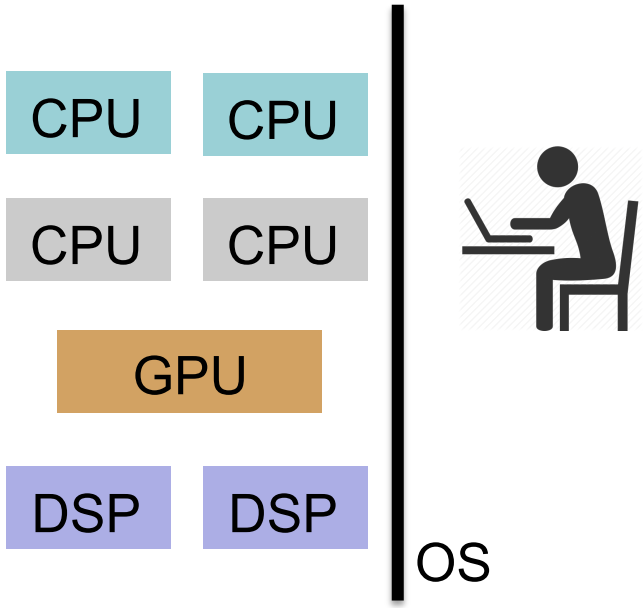
Hardware and Software Trend

1990s, early 2000



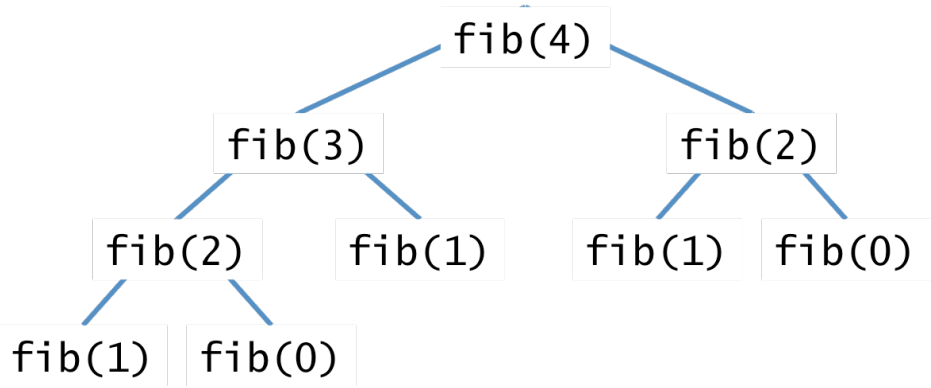
Today

(No more free lunch)



Let's Parallelize Fibonacci Program

```c
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>

uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x = fib(n-1);
    uint64_t y = fib(n-2);
    return (x + y);
  }
}

int main(int argc, char *argv[]) {
  uint64_t n = atoi(argv[1]);
  uint64_t result = fib(n);
  printf("Fibonacci of %" PRIu64 "
         is %" PRIu64 ".\n", n, result);
  return 0;
}
```



Key idea for parallelization

The calculations of `fib(n-1)` and `fib(n-2)` can be executed simultaneously without mutual interference.

Background

Let's Parallelize Fibonacci Program

```
1. uint64_t fib(uint64_t n) {
2.   if (n < 2) {
3.     return n;
4.   } else {
5.     uint64_t x = fib(n-1);
6.     uint64_t y = fib(n-2);
       return (x + y);
7.   }
8. }
9. int main(int argc, char *argv[]) {
10.   uint64_t result = fib(40);
11.   printf("Result is %" PRIu64 ".\n", result);
12.   return 0;
13. }
```

Pthreads

```
1. uint64_t fib(uint64_t n) {
2.   if (n < 2) {
3.     return n;
4.   } else {
5.     uint64_t x = fib(n-1);
6.     uint64_t y = fib(n-2);
       return (x + y);
7.   }
8. }
9. typedef struct {
10.   uint64_t input;
11.   uint64_t output;
12.} thread_args;
13.
14.void *thread_func(void *ptr) {
15.   uint64_t i = ((thread_args *) ptr)-
   >input;
16.   ((thread_args *) ptr)->output = fib(i);
17.   return NULL;
18.}
19.int main(int argc, char *argv[]) {
20.   pthread_t thread;
21.   thread_args args;
22.   int status;
23.   uint64_t result;
24.   args.input = n-1;
25.   status = pthread_create(&thread,
    NULL,thread_func,&args);
26.   if (status != NULL) { return 1; }
27.   result = fib(n-2);
28.   // Wait for the thread to terminate.
29.   status = pthread_join(thread, NULL);
30.   if (status != NULL) { return 1; }
31.   result += args.output;
32.   printf("Result is %" PRIu64 ".\n", result);
33.   return 0;
34. }
```

http://classes.engineering.wustl.edu/cse539/web/lectures/lec01_intro.pdf

Background

Let's Parallelize Fibonacci Program

```
1. uint64_t fib(uint64_t n) {
2.   if (n < 2) {
3.     return n;
4.   } else {
5.     uint64_t x = fib(n-1);
6.     uint64_t y = fib(n-2);
       return (x + y);
7.   }
8. }
9. int main(int argc, char *argv[]) {
10.   uint64_t result = fib(40);
11.   printf("Result is %" PRIu64 ".\n", result);
12.   return 0;
13. }
```

EASY

Pthreads

```
1. uint64_t fib(uint64_t n) {
2.   if (n < 2) {
3.     return n;
4.   } else {
5.     uint64_t x = fib(n-1);
6.     uint64_t y = fib(n-2);
       return (x + y);
7.   }
8. }
9. typedef struct {
10.   uint64_t input;
11.   uint64_t output;
12. } thread_args;
13.
14. void *thread_func(void *ptr) {
15.   uint64_t i = ((thread_args *) ptr)->input;
16.   ((thread_args *) ptr)->output = fib(i);
17.   return NULL;
18. }
19. int main(int argc, char *argv[]) {
20.   pthread_t thread;
21.   thread_args args;
22.   int status;
23.   uint64_t result;
24.   args.input = n-1;
25.   status = pthread_create(&thread, NULL,thread_func,&args);
26.   if (status != NULL) { return 1; }
27.   result = fib(n-2);
28.   // Wait for the thread to terminate.
29.   status = pthread_join(thread, NULL);
30.   if (status != NULL) { return 1; }
31.   result += args.output;
32.   printf("Result is %" PRIu64 ".\n", result);
33.   return 0;
34. }
```

HARD

- **Issues?**
  - Scalability
    - This code is only for 2 cores. Rewrite for more cores
  - Modularity
    - Logic no more neatly encapsulated
  - Overhead
    - Recreating thread >10^4 cycles

Other well-known options?

	Threads	Java Fork/Join	Cilk	OpenMP
Productivity	Low	Better than threads	High	High
Performance	Low	Limited	Limited	Limited

Only supported on multicore processors

- Supports both multicores and accelerators
- No support for NUMA
- Does not support hybrid execution simultaneously across multicores and accelerators
- Loosely integrated with communication libraries (MPI, UPC, etc.)

Productively Parallelizing Fibonacci Program

```
uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x, y;
    finish([&]() {
      async([&](){ x = fib(n-1);});
      y = fib(n-2);
    });
    return (x + y);
  }
}
```

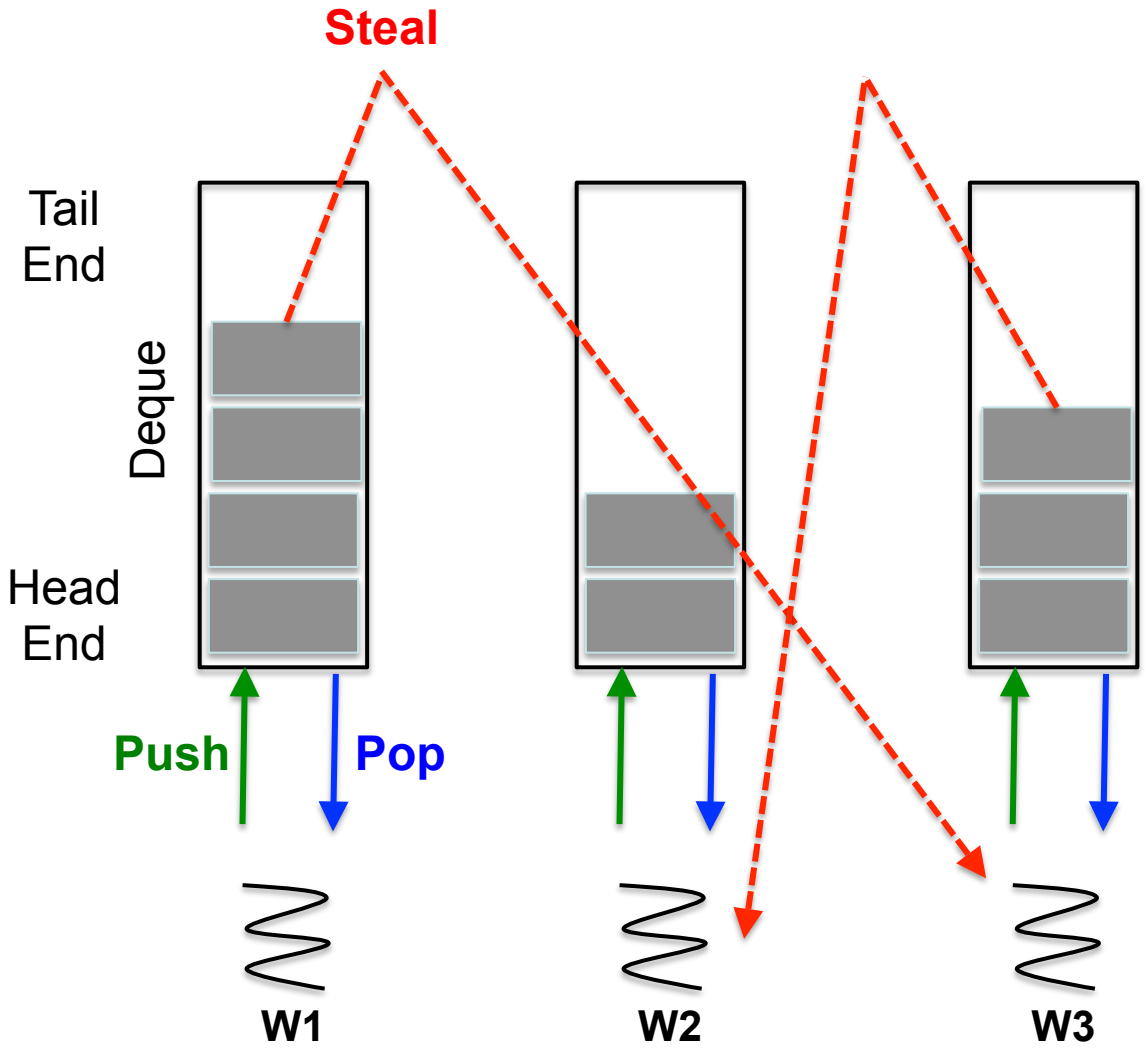
HClib (Habanero C/C++ Library)

- Open-sourced C++11/C library based implementation [1]
- Originated from Rice University
- Being used for **research** at various universities (including IIIT Delhi)
- Being used for **teaching** a parallel programming course at IIIT Delhi

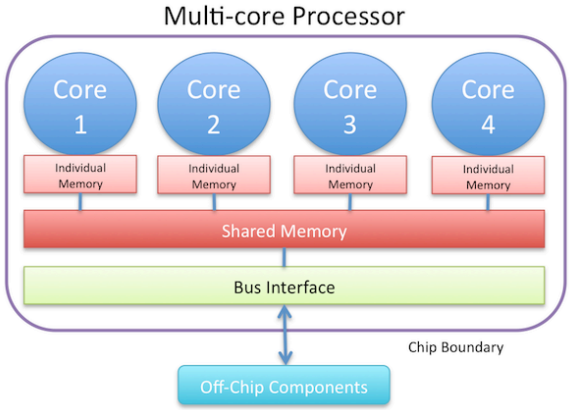
- High productivity due to **serial elision** (as in Cilk)
  - Removing all async and finish constructs results in a valid sequential program

[1] V Kumar, and V Sarkar, Tutorial on HClib, HiPC 2018 [https://github.com/habanero-rice/hclib/tree/master/tutorial/hipc18]

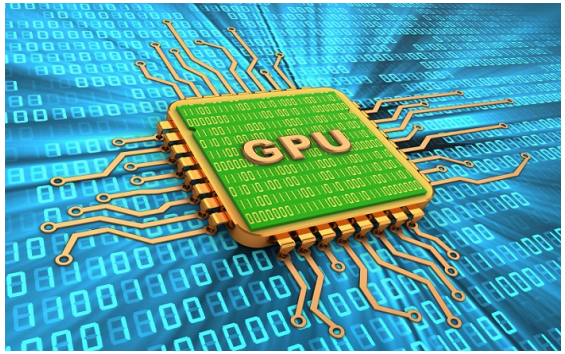
Dynamic Load Balancing using Work-Stealing



Supported on Wide Range of Architectures



Multicores

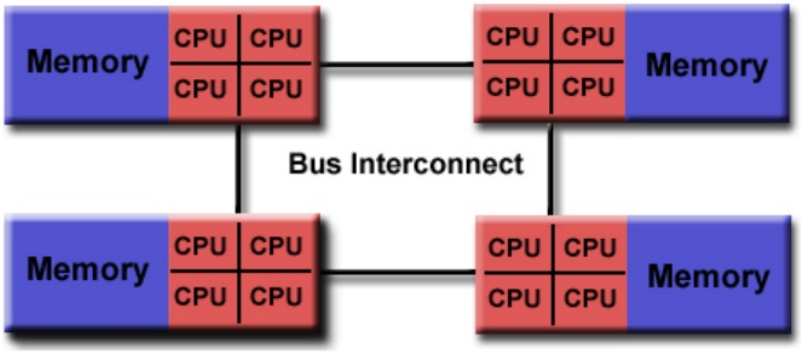


CUDA GPUs

Multiprocessor System-on-Chip

Supercomputers

Today's Agenda



NUMA processors

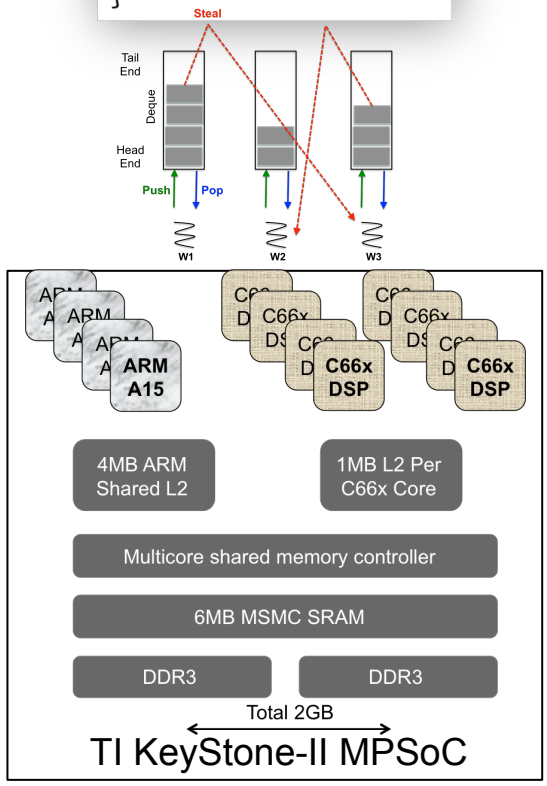
HClib: Unified Programming Model

```
finish_spmd([=]() { // global synchronization scope
        // "intra-node" asynchronous tasks
        finish([=](){ .... }); // local synchronization
        async(..., [=]() {...});
        forasync(..., [=]() {...});
        forasyncGPU(..., [=]() {...});
        async_await(..., [=]() {...});
        async_future(..., [=]() {...});

        // "inter-node" asynchronous tasks
        async_copy(src, dst, size, ...);
        async_at(remote_rank, [=]() { ... });

         // "locality-free" asynchronous tasks
         // uses distributed work-stealing
         asyncAny([=]() { ...});
});
```

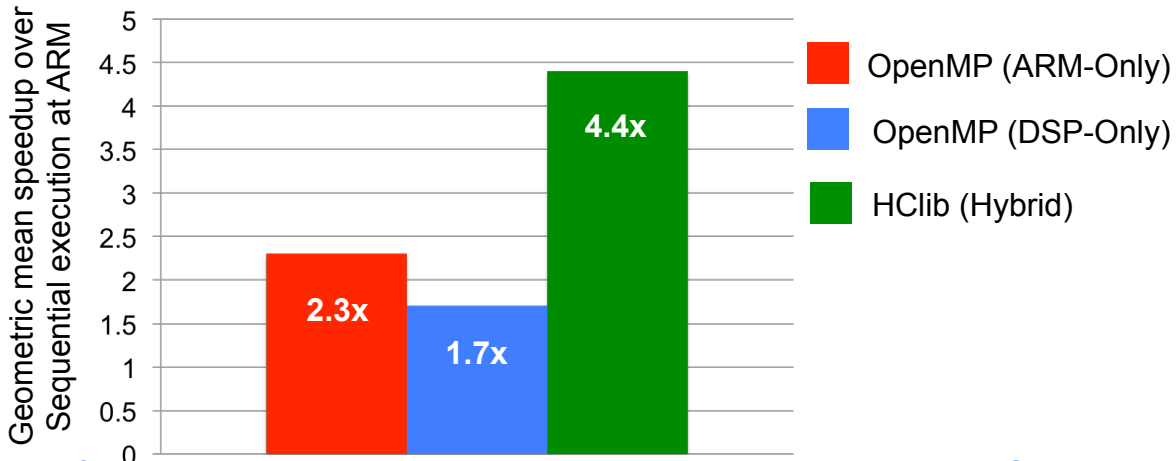
High Performance using Async-Finish

Heterogeneous Computing over MPSoC

```
foo() {
    start_finish();
    async (S1);
    forasync (S2);
    end_finish();
}
```



- Complex architecture
  - Different OS at ARM and DSP
  - No cache coherency at DSP and in between ARM & DSP
  - Only hardware queues and hardware semaphores at DSP
  - Different cache line sizes at ARM and DSP
- Unified programming model that abstracts away all hardware complexities
- Hybrid work-stealing for simultaneous execution across all 4 ARM and 8 DSP cores

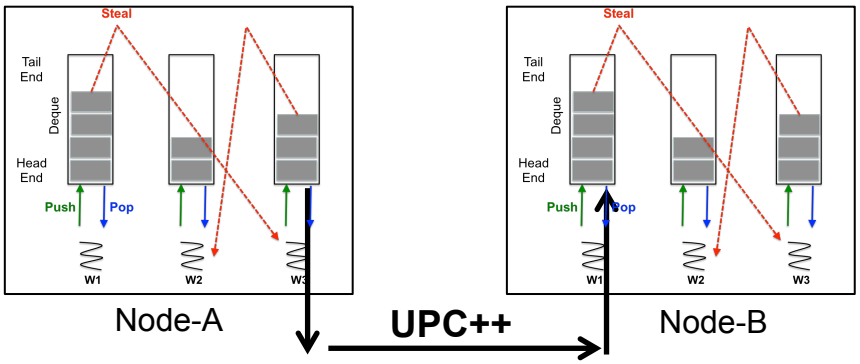


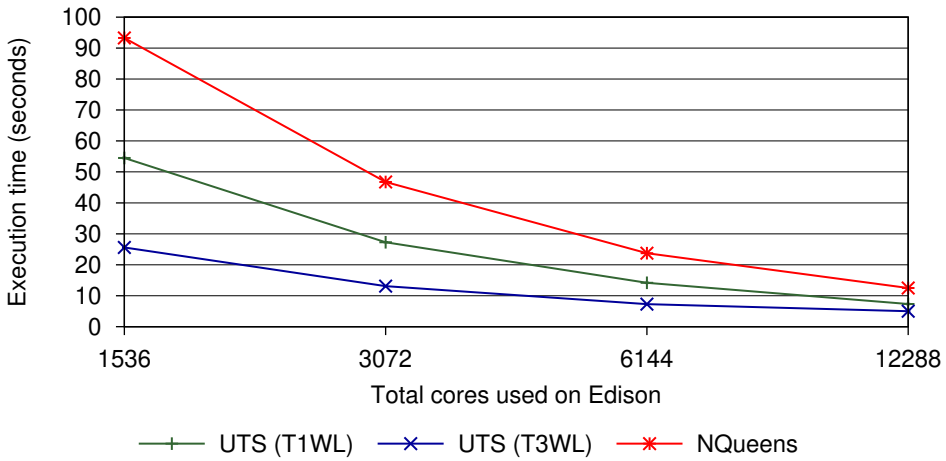
*[1] **V Kumar**, A Tiwari, G Mitra, HetroOMP: OpenMP for Hybrid Load Balancing Across Heterogeneous Processors, IWOMP 2019*

*[2] **V Kumar**, A Sberlia, Z Budimlic, and V Sarkar, Heterogeneous Work-Stealing across CPU and DSP Cores, HPEC 2015*

High Performance using Async-Finish

Distributed Computing





- Unified programming model based on tight integration between HClib and UPC++ libraries
- All **inter**-node communications (UPC++) routed through a dedicated communication worker (one) at each node [1]
  - Allows overlapping of computations and communications without using a thread-safe UPC++
- Inter- and intra-node work-stealing of locality free tasks (asyncAny) [2]

*[1] **V Kumar**, Y Zheng, V Cave, Z Budimlic, and V Sarkar, HabaneroUPC++: a Compiler-free PGAS Library, PGAS 2014*

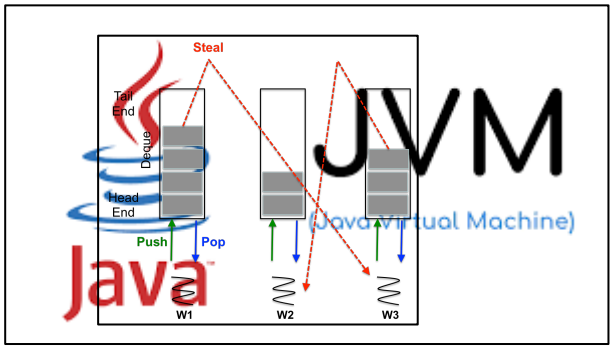
*[2] **V Kumar**, K Murthy, V Sarkar, and Y Zheng, Optimized Distributed Work-Stealing, IA^3 2016*

JVM Supported Work-Stealing (1/2)

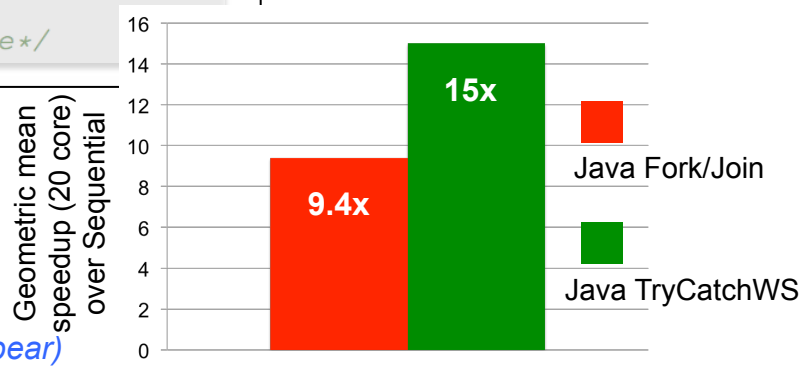
```
1.  class UTS {
2.      boolean found = false;
3.      void search() {
4.          finish_abort recurse(root);
5.      }
6.      void recurse(Node n) {
7.          if(n.equals(goal)) {
8.              found = true;
9.              abort;
10.             return;
11.         }
12.         for(int i=0; i<n.nChild; i++) {
13.             async recurse(n.child[i]);
14.         }
15.     } } Java TryCatchWS
```

```
1  class UTS { boolean found=false;
2   ForkJoinPool pool=new ForkJoinPool(2);
3   void search(){
4    try {
5     pool.invoke(new RecursiveAction(){
6      public void compute() {
7       new Recurse(root).fork();
8       helpQuiesce();
9      }
10    });
11   } catch(CancellationException e){}
12  }
13  class Recurse extends RecursiveAction{
14   Node n;
15   public Recurse(Node _n) {n=_n;}
16   public void compute() {
17    if(n.equals(goal){
18     found=true;
19     pool.shutdownNow();
20    }
21    for(int i=0;i<n.nChild;i++){
22     new Recurse(n.child[i]).fork();
23    }
24   }/*compute*/ }/*Recurse*/
25 }
```

Java Fork/Join HARD



- Work-stealing support from **inside** the JVM
  - **But, how?**

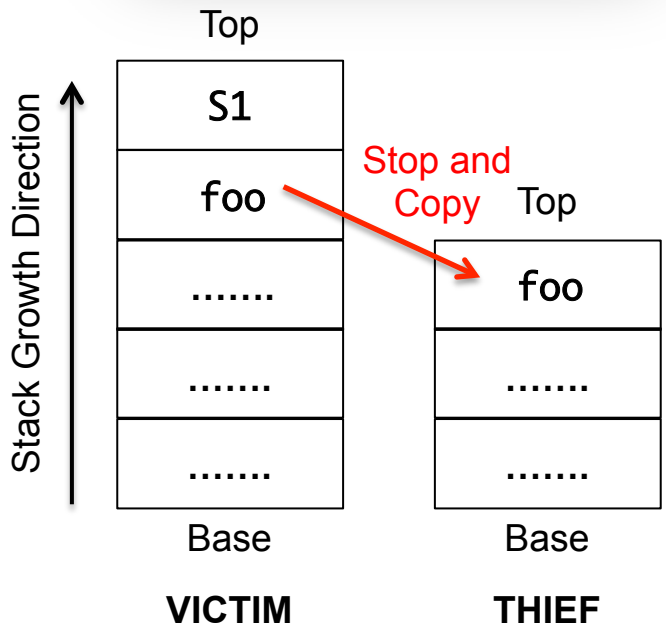


*[1] **V Kumar**, Featherlight Speculative Task Parallelism, EuroPar 2019 (to appear)*

*[2] **V Kumar**, D Frampton, SM Blackburn, D Grove, and O Tardieu, Work-Stealing Without the Baggage, OOPSLA 2012*

JVM Supported Work-Stealing (2/2)

```
foo() {
    finish {
        async X = S1();
        Y = S2();
    }
}
```




```
try {
    try {
        // Declare tasks available for stealing
        X = S1();
        // Check if anything stolen
    } catch (ExceptionEntryThief t) {
        // Entrypoint for Thief
    }
    Y = S2();
    // Try finish based synchronization
}  catch (ExceptionFinish e) {
    // 1. Store partial results
    // 2. Initiate stealing ?
}
```

Java TryCatchWS reuses runtime mechanisms already available within managed runtimes, such as JVM

- Yieldpoint mechanism
- On-stack replacement
- Dynamic code patching
- Exception handling

*[1] **V Kumar**, Featherlight Speculative Task Parallelism, EuroPar 2019 (to appear)*

*[2] **V Kumar**, D Frampton, SM Blackburn, D Grove, and O Tardieu, Work-Stealing Without the Baggage, OOPSLA 2012*

Research Focus

- Energy efficient execution of async-finish program
  - Multicore computing
  - Heterogeneous computing
  - Distributed computing
- Exploring async-finish programming in WebAssembly
- Data race detection and avoidance

- Visit us at: http://hipec.iiitd.edu.in
- Email: vivekk@iiitd.ac.in
- Openings for Research Assistants and PhD students