Lecture 04: Productivity in Parallel Programming

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Today's Lecture

- Issues with explicit multithreading
- Tasks based parallel programming model
- Quiz-1

Tasks-based parallel programming model and its underlying runtime system would be referred throughout in this course

Fibonacci using Pthread

```c
#include <inttypes.h>
#include <pthread.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);
  }
}

typedef struct {
  uint64_t input;
  uint64_t output;
} thread_args;

void *thread_func(void *ptr) {
  uint64_t i =
    ((thread_args *) ptr)->input;
  ((thread_args *) ptr)->output = fib(i);
  return NULL;
}
```

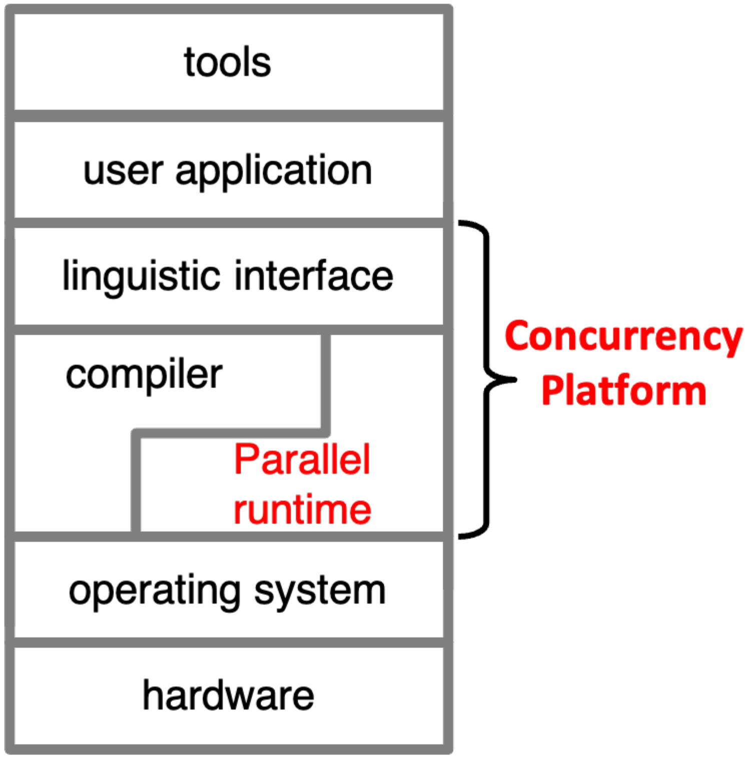
```c
int main(int argc, char argv[]) {
  pthread_t thread;
  thread_args args;
  int status;
  uint64_t result;

  if (argc < 2) { return 1; }
  uint64_t n = strtoul(argv[1], NULL, 0);
  if (n < 30) {
    result = fib(n);
  } else {
    args.input = n-1;
    status = pthread_create(&thread,
                            NULL,
                            thread_func,
                            (void*) &args);
    // main can continue executing
    if (status != NULL) { return 1; }
    result = fib(n-2);
    // Wait for the thread to terminate.
    status = pthread_join(thread, NULL);
    if (status != NULL) { return 1; }
    result += args.output;
  }
  printf("Fibonacci of %" PRIu64 " is %" PRIu64 ".\n",
         n, result);
  return 0;
}
```

What are the issues in this program?

Issues with Pthreads

Overhead	The cost of creating a thread >10^4 cycles ⇒ coarse-grained concurrency. (Thread pools can help.)
Scalability	Fibonacci code gets at most about 1.5 speedup for 2 cores. Need a rewrite for more cores. Although, array sum is scalable with increasing number of codes.
Modularity	The program logic is no longer neatly encapsulated in the original sequential function.
Code Simplicity	Programmers must marshal arguments (shades of 1958!) and engage in error-prone protocols in order to load-balance.

Parallel Runtimes



- A parallel runtime should provide:
  - an interface for specifying the ***logical parallelism*** of the computation;
  - a runtime layer to automate scheduling and synchronization; and
  - guarantees of performance and resource utilization competitive with hand-tuned code

Fibonacci using C++11 std::thread

```
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 result = fib(40);
  printf(”Result is %" PRIu64 ".\n”, result);
  return 0;
}
```

std::thread

```
uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
   uint64_t x, y;
   std::thread t1([&]() {x = fib(n-1);});
   std::thread t2([&]() {y = fib(n-2);});
   t1.join(); t2.join();
   return (x + y);
  }
}
```

- The modularity and code simplicity issues are resolved using C++11 threads
  - Will the above program run for fib(40)?

Today's Lecture

- Issues with explicit multithreading
- Tasks based parallel programming model
- Parallel runtime system for task-scheduling

Types of Tasks

- Synchronous (finish)
  - Blocks until the task execution is complete
- Asynchronous (async)
  - Doesn't blocks for the task to complete its execution

Your Sunday Tasks

Post on Facebook that you are done with all your tasks!

Wash your clothes in washing machine

Watch movies on laptop

Talk to father

Buy fruits online using your smartphone

Talk to mother

Make your bed

Complete your PRMP project deadline

Your Sunday Tasks (Stmt. Reordering)

Complete your PRMP project deadline

Wash your clothes in washing machine

Watch movies on laptop

Talk to father

Talk to mother

Buy fruits online using your smartphone

Make your bed

Post on Facebook that you are done with all your tasks!

Async-Finish Sunday Tasks

```
finish {
        async { Complete your PRMP project deadline }
        async { Wash your clothes in washing machine
}
finish {
        async { Watch movies on laptop }
        async { Talk to father
                Talk to mother }
        async { Buy fruits online using your smartphone }
        async { Make your bed }
}
```

Post on Facebook that you are done with all your tasks!

Async-Finish Sunday Tasks

finish {

async { Complete your PRMP project deadline }

~~async~~ { Wash your clothes in washing machine

}

finish { async { Watch movies on laptop }

async { Talk to father

Talk to mother }

async { Buy fruits online using your smartphone }

~~async~~ { Make your bed }

}

Post on Facebook that you are done with all your tasks!

- Statement "async S1; S2;" implies S1 and S2 could run asynchronously
- Hence, there is no need to specify "async" on S2

Async-Finish Sunday Tasks

finish {

async { Wash your clothes in washing machine }

~~}~~

~~finish {~~

~~async~~ { Complete your PRMP project deadline

async { Watch movies on laptop }

async { Talk to father

Talk to mother }

async { Buy fruits online using your smartphone }

~~async~~ { Make your bed }

}

Post on Facebook that you are done with all your tasks!

- We applied statement reordering here
  - Alas.. PRMP project deadline will take more time than washing machine…
- Hence, the second finish could be removed as we cannot launch async tasks until we are done with PRMP project implementation

Async and Finish Statements for Task Creation and Termination (Pseudocode)

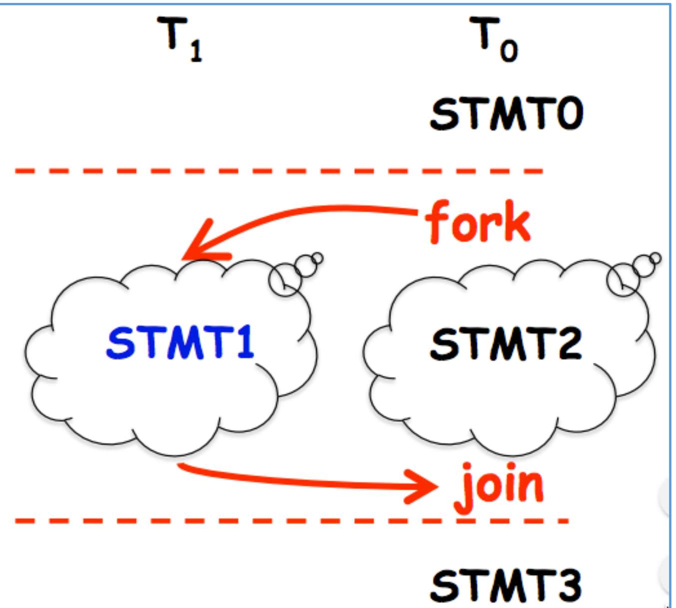
async S

- Creates a new child task that executes statement S

finish S

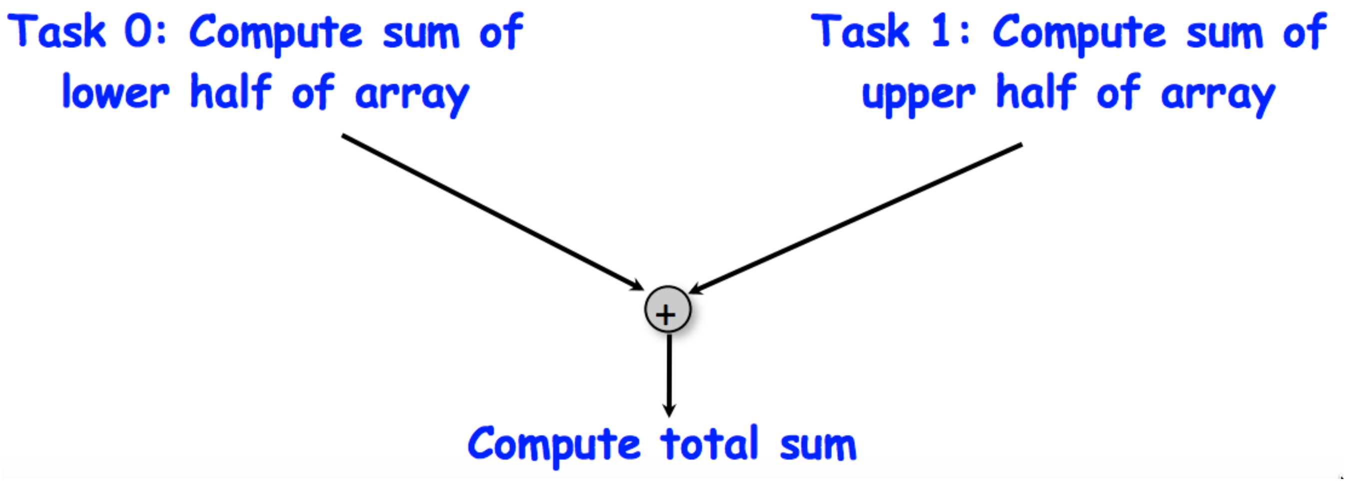
- Execute S but wait until all async in S's scope have terminated

```
// T0(Parent task)
STMT0;
finish {     //Begin finish
  async {
    STMT1; //T1(Child task)
  }
  STMT2;     //Continue in T0
             //Wait for T1
}            //End finish
STMT3;       //Continue in T0
```



Source: https://wiki.rice.edu/confluence/download/attachments/4435861/comp322-s16-lec1-slides.pdf?version=1&modificationDate=1452732285045&api=v2

2-Way Parallel Array Sum using async-finish



- Basic idea
  - Decompose the problem into tasks for partial sums
  - Combine results to obtain final answer
  - Parallel divide-n-conquer pattern

2-Way Parallel Array Sum using async-finish

Algorithm 2: Two-way Parallel ArraySum

Input: Array of numbers, X.
Output: sum = sum of elements in array X.

```
// Start of Task T1 (main program)
sum1 ← 0; sum2 ← 0;
// Compute sum1 (lower half) and sum2 (upper half) in parallel.
finish{
    async{
        // Task T2
        for i ← 0 to X.length/2 − 1 do
            sum1 ← sum1 + X[i];
    };
    async{
        // Task T3
        for i ← X.length/2 to X.length − 1 do
            sum2 ← sum2 + X[i];
    };
};
// Task T1 waits for Tasks T2 and T3 to complete
// Continuation of Task T1
sum ← sum1 + sum2;
return sum;
```

Race condition if you miss the **finish**

N-Way Parallel Array Sum

```
int A[SIZE]; // Initialization code elided
int array_sum(int low, int high) {
  int sum = 0;
  for (int i=low; i<high; i++) {
    sum += A[i];
  }
  return sum;
}
```

N-Way Parallel Array Sum using async-finish

```
int A[SIZE]; // Initialization code elided
int array_sum(int low, int high) {
  int sum = 0;
  for (int i=low; i<high; i++) {
    sum += A[i];
  }
  return sum;
}
```

```
int main(int argc, char *argv[]) {
  int result;
  if (SIZE < 1024) {
    result = array_sum(0, SIZE);
  } else {
    int chunk = SIZE/NTHREADS;
    finish([&result,=]() {
        for (int i=0; i<NTHREADS; i++) {
            async([&result,=]() {
                int low = i*chunk, high = (i+1)*chunk;
                result += array_sum(low, high);
            });
        }
    });
  }
  printf("Total Sum is %d\n", result);
  return 0;
}
```

N-Way Parallel Array Sum using async-finish

```
int A[SIZE]; // Initialization code elided
int array_sum(int low, int high) {
  int sum = 0;
  for (int i=low; i<high; i++) {
    sum += A[i];
  }
  return sum;
}
```

```
int main(int argc, char *argv[]) {
  int result;
  if (SIZE < 1024) {
    result = array_sum(0, SIZE);
  } else {
    int chunk = SIZE/NTHREADS;
    finish([&result,=]() {
        for (int i=0; i<NTHREADS; i++) {
            async([&result,=]() {
                int low = i*chunk, high = (i+1)*chunk;
                result += array_sum(low, high);
            });
        }
    });
  }
  printf("Total Sum is %d\n", result);
  return 0;
}
```

C++11 lambda functions

Race condition !!!

Do you see any issues in this version of the parallel array sum?

N-Way Parallel Array Sum using async-finish

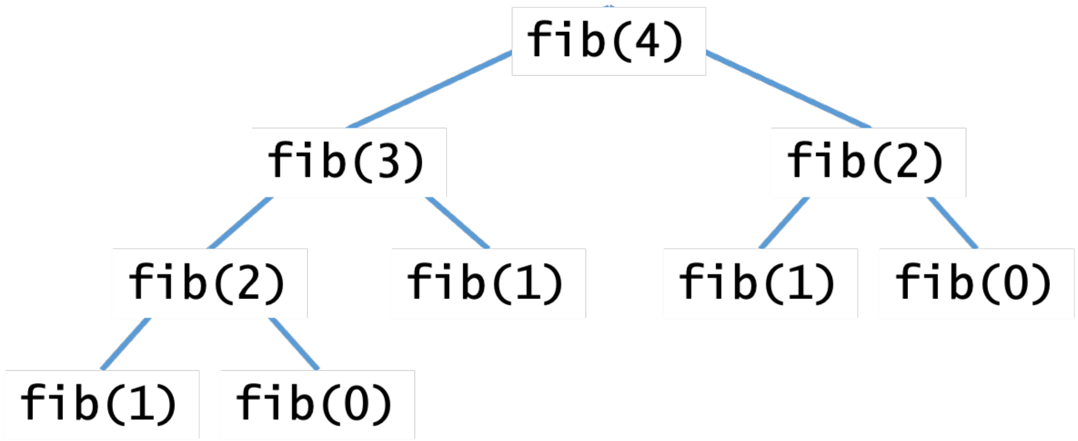
```
int A[SIZE]; // Initialization code elided
int array_sum(int low, int high) {
  int sum = 0;
  for (int i=low; i<high; i++) {
    sum += A[i];
  }
  return sum;
}
```

Race condition is fixed using an array to store result from each thread

```
int main(int argc, char *argv[]) {
  int result[NTHREADS], sum;
  if (SIZE < 1024) {
    sum = array_sum(0, SIZE);
  } else {
    int chunk = SIZE/NTHREADS;
    finish([&result,=]() {
        for (int i=0; i<NTHREADS; i++) {
            async([&result,=]() {
                int low = i*chunk, high = (i+1)*chunk;
                result[i] = array_sum(low, high);
            });
        }
    });
    for (int i=0; i<NTHREADS; i++) {
        sum += result[i];
    }
  }
  printf(”Total Sum is %d\n", sum);
  return 0;
}
```

Recursive Fibonacci using async-finish

```
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);
  }
}
```



Reading Materials

- https://doi.org/10.1007/s11227-018-2238-4

Next Lecture (#05)

- Performance in parallel programming