Parallel Task Programming

Techniques & Best Practices

BY RICCARDO TERRELL - @RIKACE

Task and Data Parallelism

Divide and Conquer

```
void QuickSort_Parallel<T>(T[] items, int left, int right)
{
    int pivot = left;


    SwapElements(items, left, pivot);

    Parallel.Invoke(
            () => QuickSort_Parallel(items, left, pivot - 1),
            () => QuickSort_Parallel(items, pivot + 1, right)
        );
}
```

Quick Sort

```
static void QuickSort<T>(T[] items, int left, int right)
{
    if (right - left < 2)
    {
        if (left+1 == right &&
            items[left].CompareTo(items[right]) > 0)
            Swap(ref items[left], ref items[right]);
        return;
    }

    int pivot = Partition(items, left, right);

    QuickSort (items, left, pivot);
    QuickSort (items, pivot + 1, right);
}
```

Quick Sort

```
static void QuickSort_Parallel<T>(T[] items, int left, int right)
{
    if (right - left < 2)
    {
        if (left+1 == right &&
            items[left].CompareTo(items[right]) > 0)
            Swap(ref items[left], ref items[right]);
        return;
    }

    int pivot = Partition(items, left, right);

    Task leftTask = Task.Run(() => QuickSort_Parallel(items, left, pivot));
    Task rightTask = Task.Run(() => QuickSort_Parallel(items, pivot + 1, right));

    Task.WaitAll(leftTask, rightTask);
}
```

Quick Sort – Oversubscription

Preemption

Quick Sort – with depth

```
void QuickSort_Parallel_Threshold<T>(T[] items, int left, int right, int depth)
{
    if (left >= right)  return;
    SwapElements(items, left, (left + right) / 2);
    var pivot = Partition(items);
    if (depth >= 0)     {
        QuickSort_Parallel_Threshold(items, left, pivot - 1, depth);
        QuickSort_Parallel_Threshold(items, pivot + 1, right, depth);
    }
    else
    {
        --depth;
        Parallel.Invoke(
                () => QuickSort_Parallel_Threshold(items, left, pivot - 1, depth),
                () => QuickSort_Parallel_Threshold(items, pivot + 1, right, depth)
            );
    }
```

Quick Sort benchmark

Measure twice and cut once

Assume nothing – Profile everything

- Be scientific
  - Do test multiple implementations
  - Don’t set out to confirm your bias
  - Instrument and profile your code

Task parallel programming in action

The Fork/Join pattern

Fork/Join

```
R InvokeParChildRelationship<T, R>(Func<R, T, R> reduce, Func<R> seedInit,
                                                        params Func<T>[] operations)
{
    var results = new T[operations.Length];
    var task = Task.Run(() =>
    {
        for (int i = 0; i < operations.Length; i++)
        {
            int index = i;
            Task.Factory.StartNew(() => results[index] = operations[index](),
                                                    TaskCreationOptions.AttachedToParent);
        }
    });
    Task.WhenAny(task);
    return results.Aggregate(seedInit(), reduce);
}
```

Sequential Fuzzy Match

```
List<string> matches = new List<string>();
foreach (var word in WordsToSearch)
{
    var localMathes = JaroWinklerModule.bestMatch(Words, word);
    matches.AddRange(localMathes.Select(m => m.Word));
}
```

Fuzzy match 7 words against 13.4 Mb of text

Time execution in 4 Logical cores – 6 Gb Ram : **23,167** ms

Two Threads Fuzzy Match

```
List<string> matches = new List<string>();
        var t1 = new Thread(() =>{
            var take = WordsToSearch.Count / 2;

            foreach (var word in WordsToSearch.Take(take))  {
                var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, word);
                matches.AddRange(localMathes.Select(m => m.Word));
            }
        });
        var t2 = new Thread(() =>{
            var start = WordsToSearch.Count / 2;
            var take = WordsToSearch.Count - start;

            foreach (var word in WordsToSearch.Skip(start).Take(take)) {
                var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, word);
                matches.AddRange(localMathes.Select(m => m.Word));
            }
        });
```

Time execution : **15,436** ms

Multi Thread Fuzzy Match

```
List<string> matches = new List<string>();
var threads = new Thread[Environm      Processor

for (int i = 0; i < threads.Lengt
                    var index = i

      thre

      var take = Words                                          hreads.Length));
      var start = index                                          e : index * take;


      foreach                                                   e)) {
                                                                bestMatch(Words, word);
                  matches                                       rd));

                      }
                  }

for (int i = 0;  < threads.Leng  ; i++)
    threads[i].Start();
for (int i = 0; i < threads.Length; i++)
   threads[i].Join();
```

WRONG!!!

Time execution : **6,857** ms

Multi Thread Fuzzy Match

```
List<string> matches = new List<string>();
var threads = new Thread[Environment.ProcessorCount];

for (int i = 0; i < threads.Length; i++) {
      var index = i;
      threads[index] = new Thread(() =>   {

      var take = WordsToSearch.Count / (Math.Min(WordsToSearch.Count, threads.Length));
      var start = index == threads.Length - 1 ? WordsToSearch.Count - take : index * take;


      foreach (var word in WordsToSearch.Skip(start).Take(take)) {
              var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, word);
              lock (matches)
                      matches.AddRange(localMathes.Select(m => m.Word));
                      }
                  });
                }

 for (int i = 0; i < threads.Length; i++)
    threads[i].Start();
for (int i = 0; i < threads.Length; i++)
   threads[i].Join();
```

Time execution : **7,731** ms

Task Parallel Library (TPL) - Task Programming

- Efficient and scalable use of system resources..
  - When used correctly
- Programmatic fine grain control
- Model largely applicable
- Easy to integrate into existing programs

- Hard to solve complex problems
- Sometimes requires the introduction of locking primitives

Parallel Loop Fuzzy Match

```
 List<string> matches = new List<string>();
object sync = new object();

Parallel.ForEach(WordsToSearch,
                 // thread local initializer
                 () => { return new List<string>(); },
                 (word, loopState, localMatches) =>  {
                      var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, word);
                        localMatches.AddRange(localMathes.Select(m => m.Word));// same code
                        return localMatches;
                 },
    (finalResult) =>
    {
        // thread local aggregator
        lock (sync) matches.AddRange(finalResult);
    }
);
```

Time execution : **7,429** ms

Parallel Loop … no for I/O

```
Task ForEachAsync<T>(this IEnumerable<T> source, int dop,
                                                Func<T, Task> body)
{
    return Task.WhenAll(
        from partition in Partitioner.Create(source).GetPartitions(dop)
        select Task.Run(async delegate {
            using (partition)
                while (partition.MoveNext())
                    await body(partition.Current);
        }));
}
```

Parallel LINQ (PLINQ)

```
var query =
    from i in Enumerable.Range(1, 10).AsParallel()
```

LINQ to Objects
LINQ to XML
not LINQ to SQL,EF

Extension method in System.LINQ

Extends **IEnumerable<T>**

"Usually" would be
Enumerable

Results in a **ParallelQuery<T>**

Select() | Where() | Etc.

PLINQ Fuzzy Match

```
ParallelQuery<string> matches =
        (from word in WordsToSearch.AsParallel()
         from match in JaroWinklerModule.bestMatch(Words, word)
         select match.Word);
```

Time execution : **6,347** ms

PLINQ under the hood

PLINQ options

```
var parallelQuery = from t in source.AsParallel()
    select t;

var cts = new CancellationTokenSource();
cts.CancelAfter(TimeSpan.FromSeconds(3));

    parallelQuery
        .WithDegreeOfParallelism(Environment.ProcessorCount)
        .WithExecutionMode(ParallelExecutionMode.ForceParallelism)
        .WithMergeOptions(ParallelMergeOptions.Default)
        .WithCancellation(cts.Token)
        .ForAll(Console.WriteLine);
```

Multi Task Fuzzy Match

```
var tasks = new List<Task<List<string>>>();
var matches = new ThreadLocal<List<string>>(() => new List<string>());

foreach (var word in WordsToSearch) {

        tasks.Add(Task.Factory.StartNew<List<string>>((w) => {
             List<string> localMatches = matches.Value;
             var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, w);
             localMatches.AddRange(localMathes.Select(m => m.Word));
             return localMatches;
    }, word));
}

Task.WhenAll(tasks.ToArray(), (ts) =>
      return new List<string>(tasks.SelectMany(t => t.Result).Distinct()));
```

Time execution : **4,492** ms

Partitioning

Partitioning in PLINQ

Range Partitioning

Chunk Partitioning

Striped Partitioning

Hash Partitioning

PLINQ Fuzzy Match

```
Create a load-balancing partitioner, specify false for static partitioning.

Partitioner<string> partitioner = Partitioner.Create(WordsToSearch, false);

ParallelQuery<string> matches =

       (from word in partitioner.AsParallel()

        from match in JaroWinklerModule.bestMatch(Words, word)

        select match.Word);
```

Time execution : **4,217** ms

PLINQ Fuzzy Match

```
Create a load-balancing partitioner, specify false for static partitioning.

Partitioner<string> partitioner = Partitioner.Create(WordsToSearch, false);

ParallelQuery<string> matches =

        (from word in partitioner.AsParallel()

         from match in JaroWinklerModule.bestMatch(Words, word)

         select match.Word);
```

Deforesting your query

Deforesting your query

```
var data = new int[100000];
for(int i = 0; i < data.Length; i++)
    data[i]=i;


long total =
    data.AsParallel()
        .Where(n => n % 2 == 0)
        .Select(n => n + n)
        .Select(n=> (long)n)
        .Sum();
```

New Project
.NET Framework 4.5
Sort by:
Default
Search Installed Te
Installed
Templates
Visual C#
Visual C++
SQL Server
Visual F#
F# Application
Visual F#
F# Library
F# Tutorial
Visual F#
Type: Visual F#
A tutorial project providing a walkthrough of the F# language
Name:
Tutorial1
Location:
C:\source\fsharpintro
Create new solution
Solution name:
Add to source control
OK
DEMO

Task continuation

Think Continuations, not Callbacks

Continuation Passing Style

```
void GetMaxCPS(int x, int y, Action<int> f)
{
    if (x > y)
        f(x);
    else
        f(y);
}
```

Task continuation

```
void RunContinuation<T>(Func<T> input, Action<T> rest)
        => ThreadPool.QueueUserWorkItem(new WaitCallback(o => rest(input())));


Action<string> printMsg = msg =>
    Console.WriteLine($"ThreadID = {Thread.CurrentThread.ManagedThreadId}, Url = {url}, {msg}");

RunContinuation(() => {
    printMsg("Creating webclient...");
    return new System.Net.WebClient();
}, (webclient) => RunContinuation(() => {
        printMsg("Downloading url...");
        return webclient.DownloadString(url);
    }, (html) => RunContinuation(() => {
            printMsg("Extracting urls...");
            return Regex.Matches(html, @"http://\S+");
        }, (matches) =>
                printMsg("Found " + matches.Count.ToString() + " links")
            )));
```

Task continuation

```
ThreadID = 19, Url = http://www.google.com/, Creating webclient...
ThreadID = 21, Url = http://microsoft.com/, Creating webclient...
ThreadID = 16, Url = http://www.wordpress.com/, Creating webclient...
ThreadID = 18, Url = http://www.wordpress.com/, Downloading url...
ThreadID = 11, Url = http://www.google.com/, Downloading url...
ThreadID = 19, Url = http://microsoft.com/, Downloading url...
ThreadID = 15, Url = http://www.wordpress.com/, Extracting urls...
ThreadID = 16, Url = http://www.wordpress.com/, Found 15 links
ThreadID = 18, Url = http://www.peta.org, Creating webclient...
ThreadID = 05, Url = http://www.peta.org, Downloading url...
ThreadID = 19, Url = http://www.google.com/, Extracting urls...
ThreadID = 16, Url = http://www.google.com/, Found 14 links
ThreadID = 20, Url = http://www.peta.org, Extracting urls...
ThreadID = 11, Url = http://www.peta.org, Found 60 links
ThreadID = 24, Url = http://microsoft.com/, Extracting urls...
ThreadID = 21, Url = http://microsoft.com/, Found 1 links
```

Task continuation

```
Task.Run(() =>
{
    printMsg("Creating webclient...");
    return new System.Net.WebClient();
}).ContinueWith(taskWebClient =>
{
    var webclient = taskWebClient.Result;
    printMsg("Downloading url...");
    return webclient.DownloadString(url);
}).ContinueWith(htmlTask =>
{
    var html = htmlTask.Result;
    printMsg("Extracting urls...");
    return Regex.Matches(html, @"http://\S+");
}).ContinueWith(rgxTask =>
{
    var matches = rgxTask.Result;
    printMsg("Found " + matches.Count.ToString() + " links");
});
```

Task continuation

```
var matches = from webclient in Task.Run(() => new System.Net.WebClient())
              from html in Task.Run(() => webclient.DownloadString(url))
              from rgx in Task.Run(() => Regex.Matches(html, @"http://\S+"))
              select rgx;
```

Mathematical patterns for better composition

Composing Tasks

Function Composition

Function Composition - Building with composition. Composition is the 'glue' that allows us build larger systems from smaller ones. This is the very heart of the functional style. Almost every line of code is a composable expression. Composition is used to build basic functions, and then functions that use those functions, and so on.

Composition — in the form of passed parameters plus first-class functions

How can we compose Tasks?

```
Task<int> RunOne() => // ...
Task<decimal> RunTwo(int input) => // ...

var result = RunTwo(RunOne()); // Error!!
```

C#

```
Task<string> RunOne(int input) =>  // ...
Task<bool> RunTwo(string input) => // ...

Func<T1, T3> Compose<T1, T2, T3>(Func<T1, T2> f1,
                                 Func<T2, T3> f2) => a => f2(f1(a));

var result = RunOne(42).Compose(RunTwo); // Error!!
```

C#

Composing Tasks

```
Task<R> Bind<T, R>(this Task<T> m, Func<T, Task<R>> k) …

Task<T> Return(T value) …
```

Functional Design Patterns – Monad

Generic type

With function (>>=): bind : M<‘a> -> (‘a -> M<‘b>) -> M<‘b>

With function: return : ‘a -> M<‘a>

Composing Task

```
static member Then(input : Task<'T>, binder : 'T -> Task<'U>) =
     let tcs = new TaskCompletionSource<'U>()
     input.ContinueWith(fun (task:Task<'T>) ->
        if (task.IsFaulted) then
          tcs.SetException(task.Exception.InnerExceptions)
        elif (task.IsCanceled) then tcs.SetCanceled()
        else
             try
                (binder(task.Result)).ContinueWith(fun
(nextTask:Task<'U>) -> tcs.SetResult(nextTask.Result)) |> ignore
             with
             | ex -> tcs.SetException(ex)) |> ignore
     tcs.Task
```

Task Bind and Task Return

```
// 'T -> M<'T>
static member Return (value : 'T) : Task<'T>

 // M<'T> * ('T -> M<'U>) -> M<'U>
static member Bind (input : Task<'T>, binder : 'T -> Task<'U>) : Task<'U>
```

```
// M<'T> * ('T -> M<'U>) -> M<'U>
static member SelectMany (input : Task<'T>, binder : 'T -> Task<'U>) : Task<'U>
```

The hidden Functor – Map

Map : (T -> R) -> [T] -> [R]

```
Select(this IEnumerable<T> task, Func<T , R> map) …

Select(this Task<T> task, Func<T , R> map) …
```

IEnumerable SelectMany

```
// M<T> * (T -> M<U>) -> M<U>
IEnumerable<R> SelectMany<T, R>( this IEnumerable<T> source,
                                 Func<T, IEnumerable<R>> selector )


// M<T> * (T -> M<U>) -> (T -> M -> U) -> M<U>
IEnumerable<R> SelectMany<T, M, R>( this IEnumerable<T> source,
                Func<T, IEnumerable<M>> collectionSelector, Func<T, M, R> resultSelector )


// M<T> * (T -> M<U>) -> M<U>
IEnumerable<R> Select<T, R>( this IEnumerable<T> source, Func<T, R> selector )
```

Task SelectMany

```
// M<T> * (T -> M<U>) -> M<U>
Task<R> Task<T, R>( this Task<T> source,
                                        Func<T,  Task<R>> selector )


// M<T> * (T -> M<U>) -> (T -> M -> U) -> M<U>
Task<R> SelectMany<T, M, R>( this Task<T> source,
                Func<T,  Task<M>> collectionSelector, Func<T, M, R> resultSelector )

// M<T> * (T -> M<U>) -> M<U>
Task<R> Select<T, R>( this Task<T> source, Func<T, R> selector )
```

Composing Tasks

```
from image in Task.Run<Emgu.CV.Image<Bgr, byte>()
from imageFrame in Task.Run<Emgu.CV.Image<Gray, byte>>()
from faces in Task.Run<System.Drawing.Rectangle[]>()
select faces;
```

Composing Task async/await

```
static Task<T> Return<T>(T task)=> Task.FromResult(task);


static async Task<R> Bind<T, R>(this Task<T> task,
    Func<T, Task<R>> cont)
     => await cont(await task.ConfigureAwait(false))
                                        .ConfigureAwait(false);

static async Task<R> Map<T, R>(this Task<T> task, Func<T, R> map)
=> map(await task.ConfigureAwait(false));
```

Composing Task async/await

```
static async Task<R> SelectMany<T, R>(this Task<T> task,
        Func<T, Task<R>> then) => await Bind(await task);

static async Task<R> SelectMany<T1, T2, R>(this Task<T1> task,
         Func<T1, Task<T2>> bind, Func<T1, T2, R> project)
        {
            T taskResult = await task;
            return project(taskResult, await bind(taskResult));
        }

static async Task<R> Select<T, R>(this Task<T> task, Func<T, R> project)
            => await Map(task, project);

static async Task<R> Return<R>(R value) => Task.FromResult(value);
```

Task Inline - ExecuteSynchronously

```
public static Task<TOut> SelectMany<TIn, TOut>(this Task<TIn> first, Func<TIn,
Task<TOut>> next)
{
    var tcs = new TaskCompletionSource<TOut>();
    first.ContinueWith(delegate
    {
        if (first.IsFaulted) tcs.TrySetException(first.Exception.InnerExceptions);
        else if (first.IsCanceled) tcs.TrySetCanceled();
        else
        {
                next(first.Result).ContinueWith(t =>
                {
                    if (t.IsFaulted) tcs.TrySetException(t.Exception.InnerExceptions);
                    else if (t.IsCanceled) tcs.TrySetCanceled();
                    else tcs.TrySetResult(t.Result);
                }, TaskContinuationOptions.ExecuteSynchronously);
    }, TaskContinuationOptions.ExecuteSynchronously);
    return tcs.Task;
}
```

Task continuation

```
task1.ContinueWith(antecedent =>
        Console.WriteLine("Task #1 completion continuation."),
        TaskContinuationOptions.OnlyOnRanToCompletion);

task1.ContinueWith(antecedent =>
        Console.WriteLine("Task #1 cancellation continuation."),
        TaskContinuationOptions.OnlyOnCanceled);

task1.ContinueWith(antecedent =>
        Console.WriteLine("Task #1 on error continuation."),
        TaskContinuationOptions.OnlyOnFaulted);

task1.ContinueWith(antecedent =>
        Console.WriteLine("Task #1 continuation lomg running."),
        TaskContinuationOptions.LongRunning);
```

Process Tasks as they complete

Handle Tasks as they complete

```
string[] stocks = new[] { "MSFT", "FB", "AAPL","GOOG", "ORCL" };

List<Task<Tuple<string, StockData[]>>> stockHistoryTasks =
    stocks.Select(ProcessStockHistory).ToList();

while (stockHistoryTasks.Count > 0)
{
    Task<Tuple<string, StockData[]>> stockHistoryTask =
        await Task.WhenAny(stockHistoryTasks);

     stockHistoryTasks.Remove(stockHistoryTask);

    Tuple<string, StockData[]> stockHistory = await stockHistoryTask;

    await ProcessStock(stockHistory);
}
```