LAB 3

Introduction

In this laboratory we are going to study the implementation of "divide and conquer" strategy using OpneMP paralization. **(ha de continuar)**

Task decomposition analysis for Mergesort

Divide and conquer

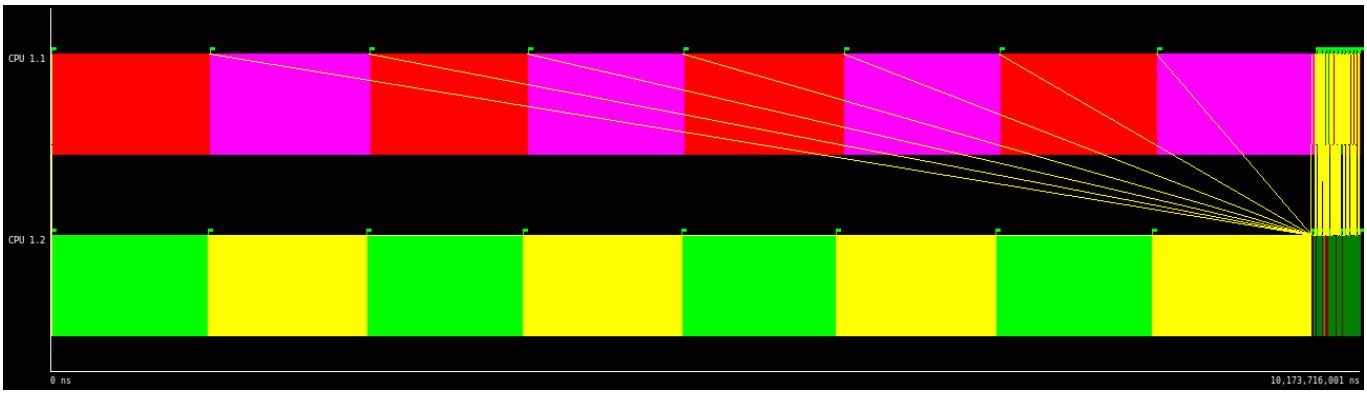
Task decomposition analysis with Tareador

```
ADD CODE
```

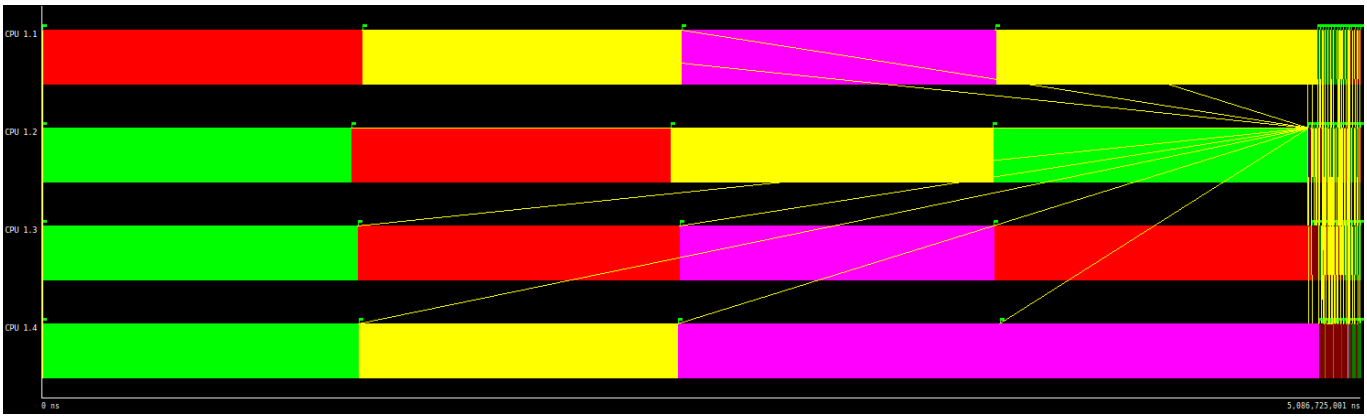
link code

We have added a tareador task in each of the recursive tasks, to fully visualize the possible parallelizations of the code.

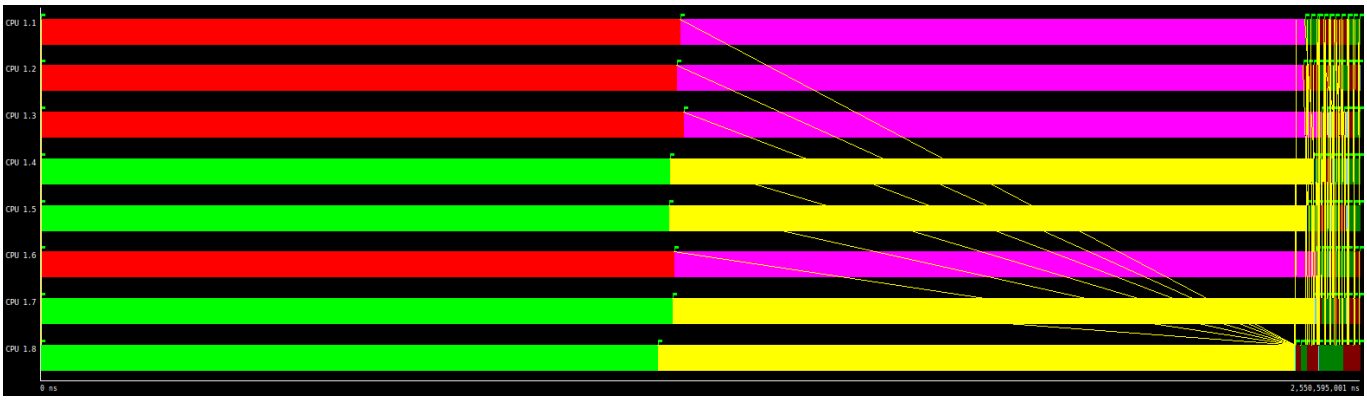
Trace of multisort-tareador using 1 core



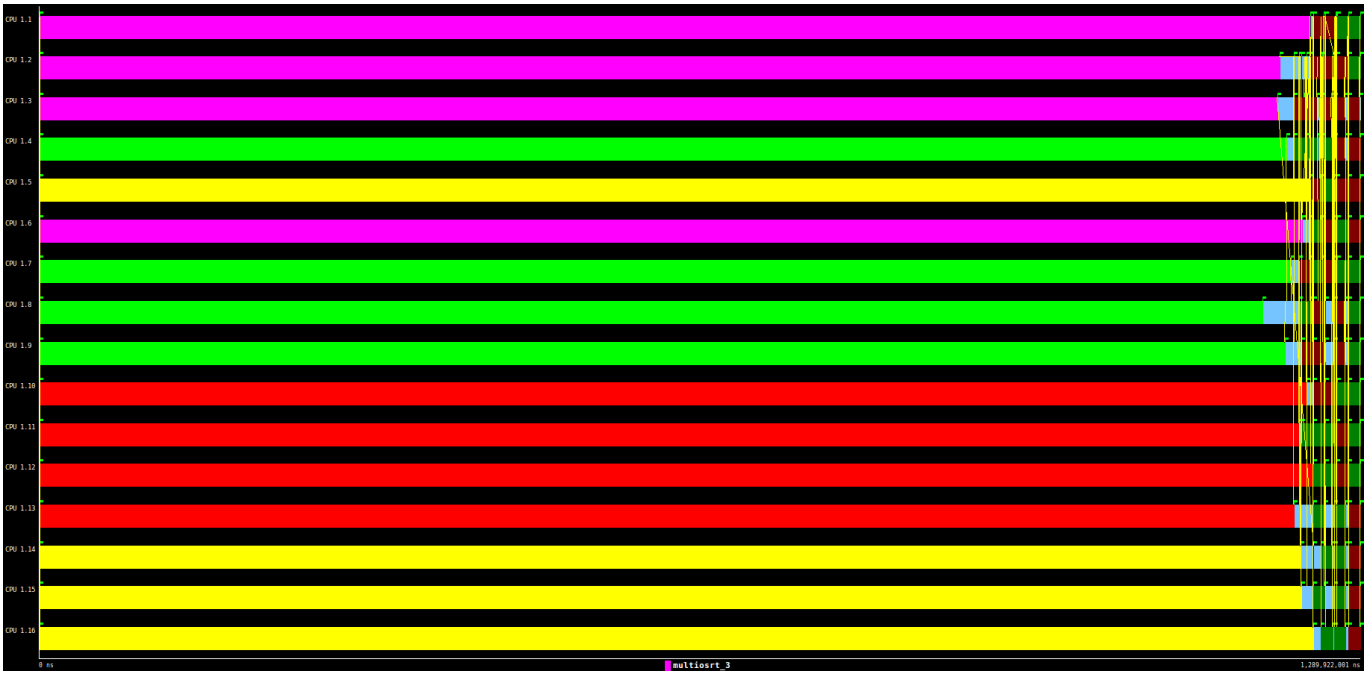
Trace of multisort-tareador using 2 core



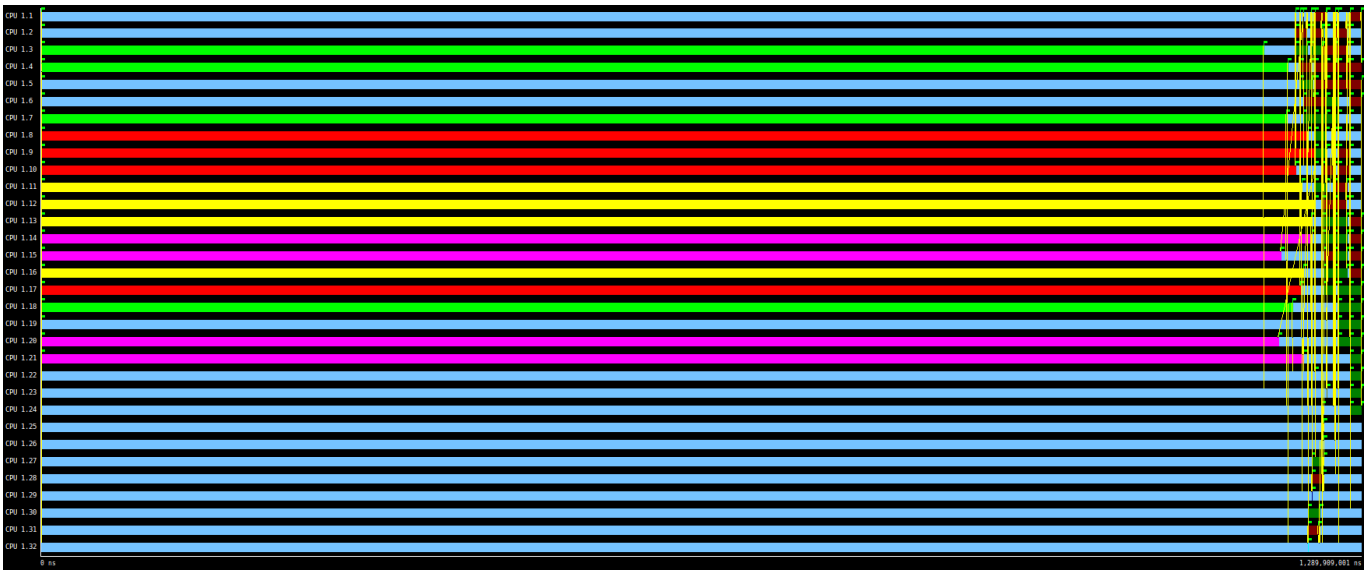
Trace of multisort-tareador using 4 core



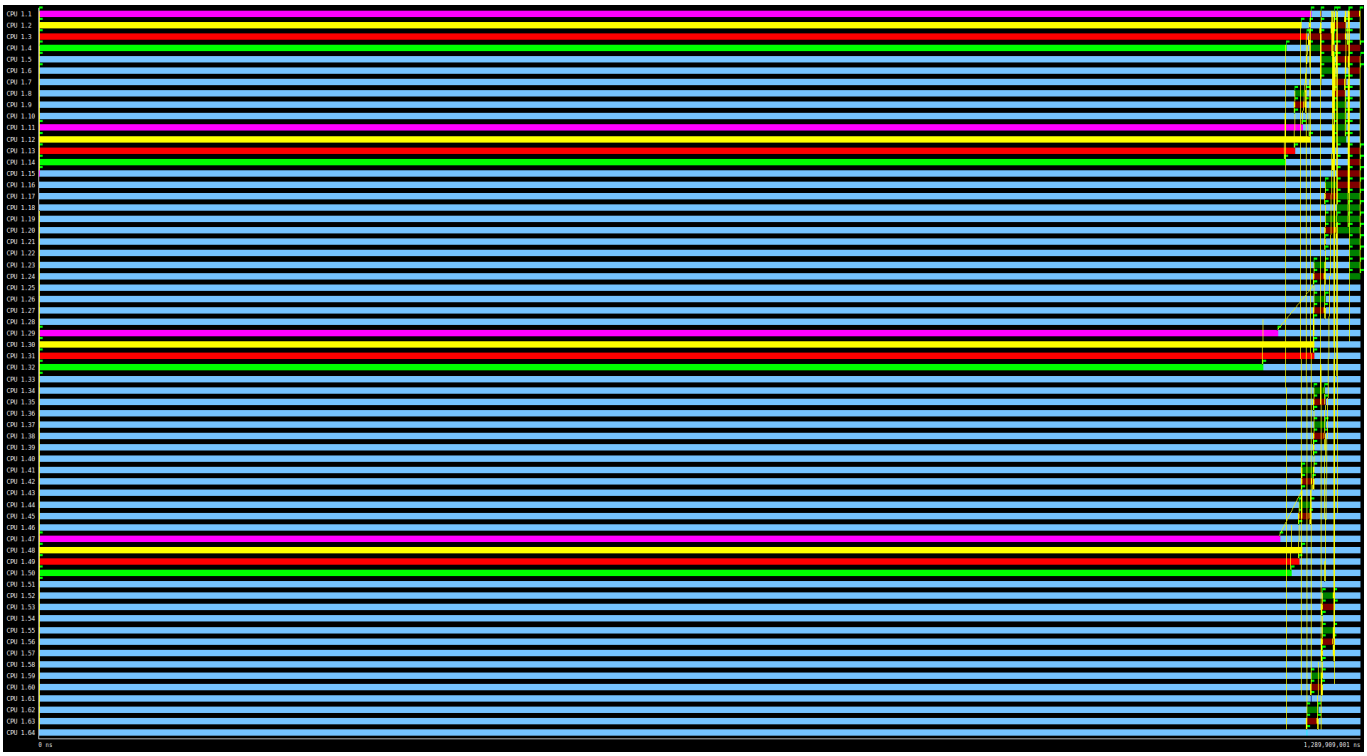
Trace of multisort-tareador using 8 core



Trace of multisort-tareador using 16 core



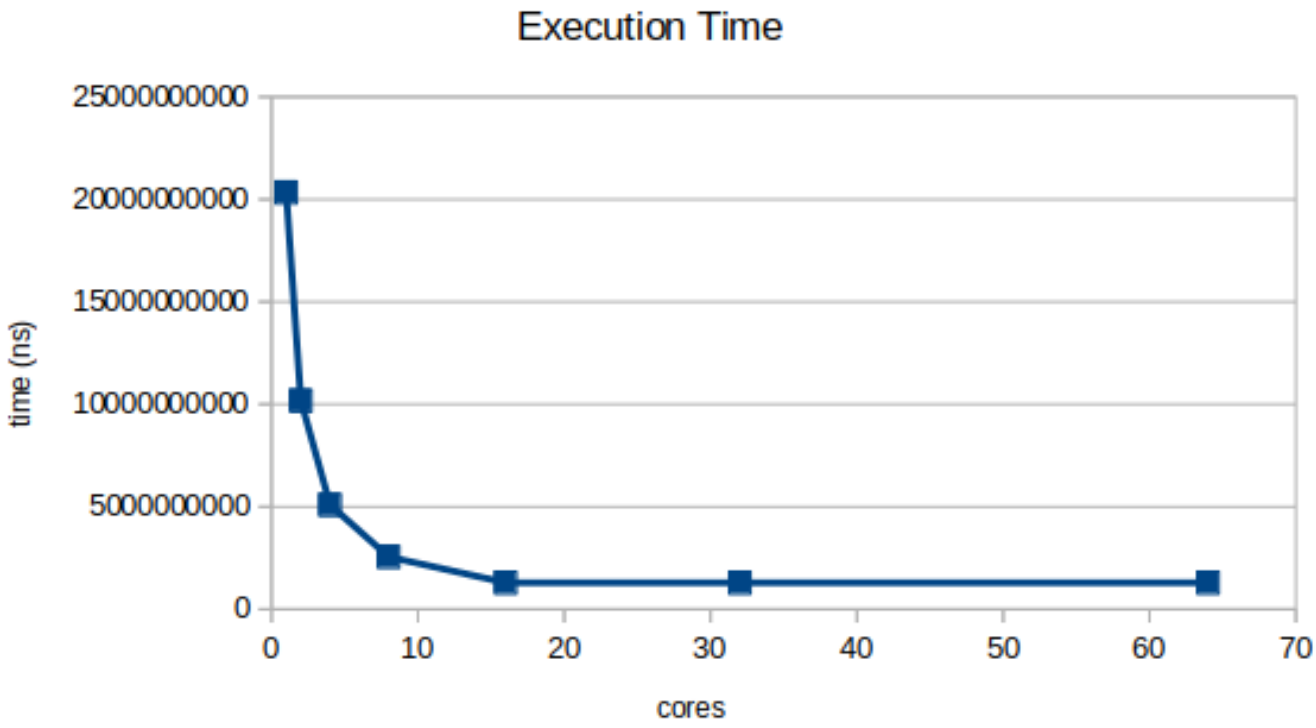
Trace of multisort-tareador using 32 core



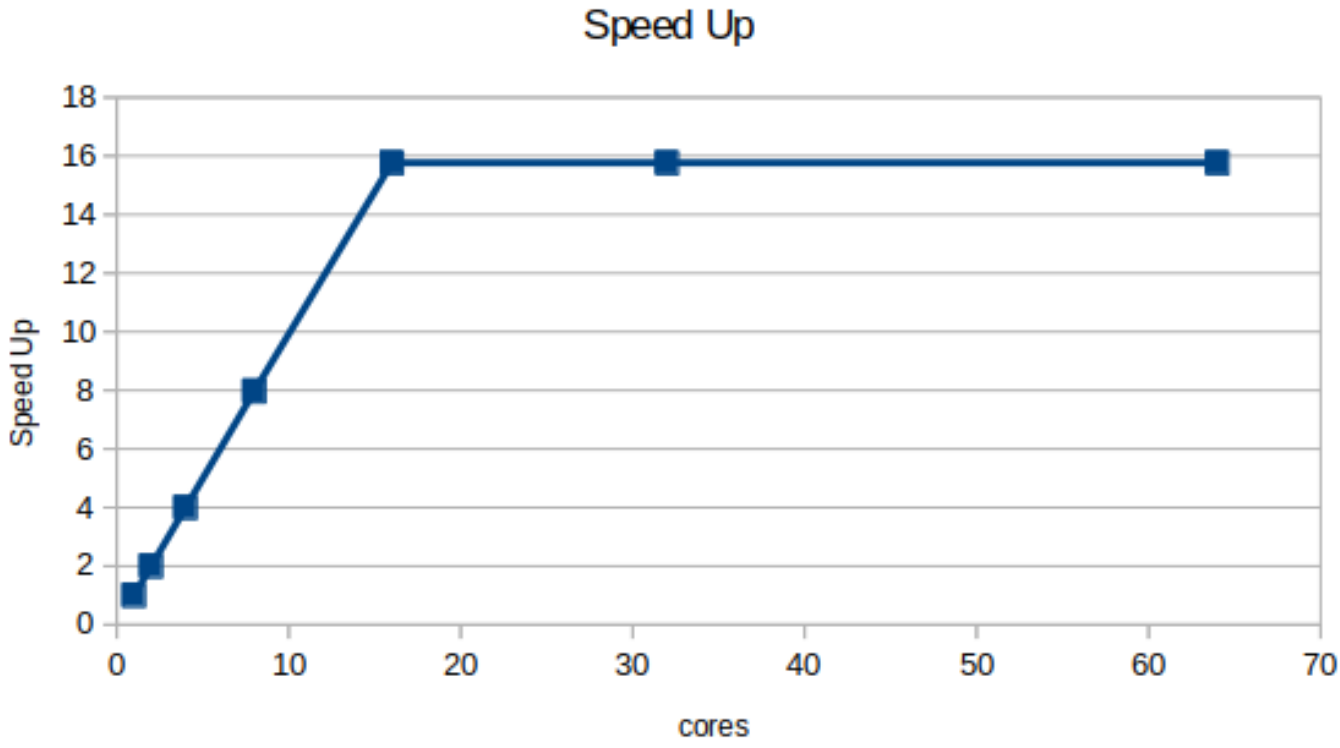
Trace of multisort-tareador using 64 core

NUM CPUs	time (ns)	SPEED UP	EFFICIENCY
1	20334411001	1	0.999360066616823
2	10173716001	1.99872013323365	0.99938619627572
4	5086725001	3.99754478510288	0.99938619627572
8	2550595001	7.97241858979085	0.996552323723856
16	1289922001	15.7640624667507	0.985253904171916
32	1289909001	15.7642213406029	0.492631916893841
64	1289909001	15.7642213406029	0.246315958446921

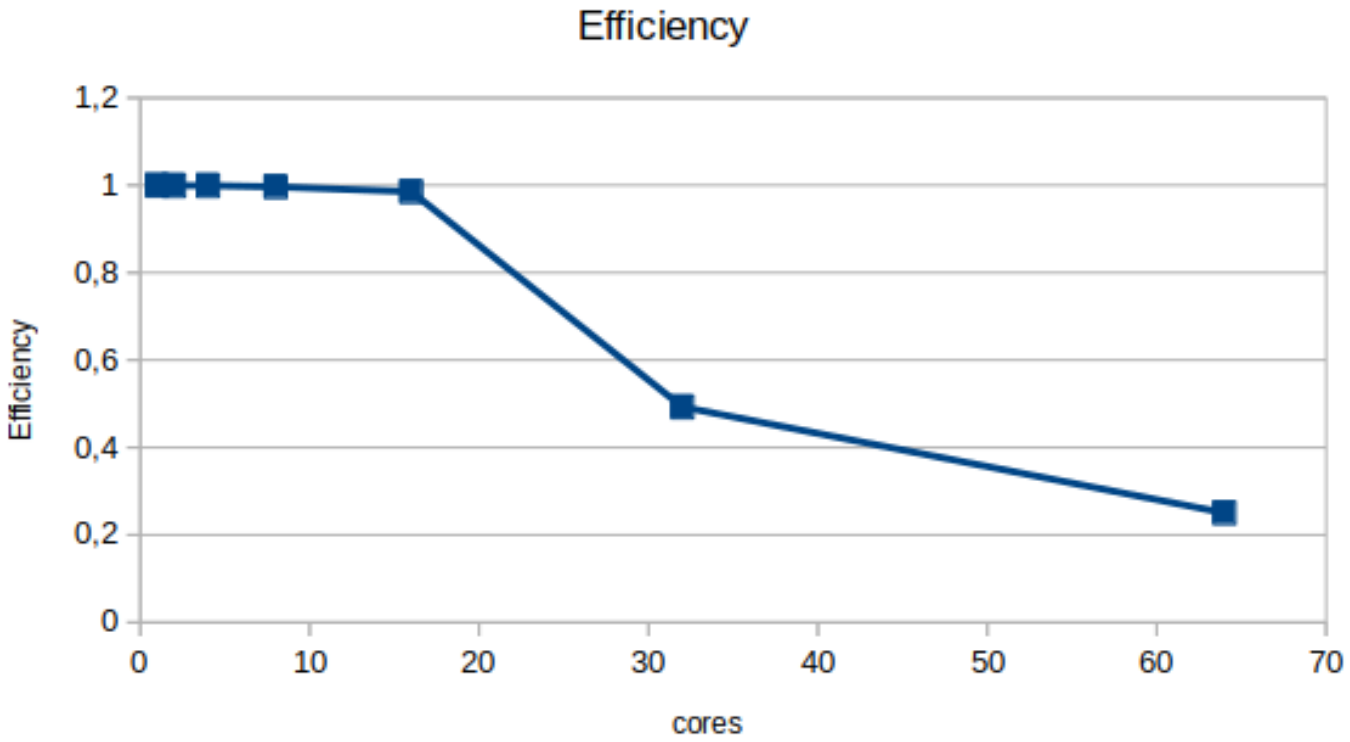
The following pots represents number of cores vs. plot of time, speed up and efficiency, respectively.



Number of cores vs. execution time plot



Number of cores vs. speed up plot



Number of cores vs. efficiency plot

The multisort program parallizes ideally untill 16 cores, when , the dependences between the multisort marge calls, as seen in the tareador capture of the tasks, doesn't allow for more. The efficiency drops going further than 16 threads (i.e fig: efficiency plot and 32,64 captures) Proving that adding more CPUs is pointless.

Shared-memory parallelization with OpenMP tasks

Task cut–off mechanism

Tree stragey code:

```
void merge(long n, T left[n], T right[n], T result[n*2], long start, long
length) {
    if (length < MIN_MERGE_SIZE*2L) {
        // Base case
        basicmerge(n, left, right, result, start, length);
    } else {
        // Recursive decomposition
        #pragma omp taskgroup
        {
            #pragma omp task
            merge(n, left, right, result, start, length/2);
            #pragma omp task
            merge(n, left, right, result, start + length/2, length/2);
        }
    }
}

void multisort(long n, T data[n], T tmp[n]) {
    if (n >= MIN_SORT_SIZE*4L)
    {
        #pragma omp taskgroup
        {
            #pragma omp task
            multisort(n/4L, &data[0], &tmp[0]);
            #pragma omp task
            multisort(n/4L, &data[n/4L], &tmp[n/4L]);
            #pragma omp task
            multisort(n/4L, &data[n/2L], &tmp[n/2L]);
            #pragma omp task
            multisort(n/4L, &data[3L*n/4L], &tmp[3L*n/4L]);
        }
        #pragma omp taskgroup
        {
            #pragma omp task
            merge(n/4L, &data[0], &data[n/4L], &tmp[0], 0, n/2L);
            #pragma omp task
            merge(n/4L, &data[n/2L], &data[3L*n/4L], &tmp[n/2L], 0, n/2L);
            //#pragma omp taskwait
        }
        #pragma omp task
        merge(n/2L, &tmp[0], &tmp[n/2L], &data[0], 0, n);
        }
    else
    {
                basicsort(n, data);
```

```
        }
}
```

multisort-omp-cutoff.c

	multisort	merge
THREAD 1.1.1	266	3,098
THREAD 1.1.2	259	2,956
THREAD 1.1.3	329	1,990
THREAD 1.1.4	280	2,782
THREAD 1.1.5	329	2,028
THREAD 1.1.6	294	1,812
THREAD 1.1.7	301	1,872
THREAD 1.1.8	329	1,896
Total	2,387	18,434
Average	298.38	2,304.25
Maximum	329	3,098
Minimum	259	1,812
StDev	26.87	506.70
Avg/Max	0.91	0.74

Lear stragegy code:

```
void multisort(long n, T data[n], T tmp[n]) {
        if (n >= MIN_SORT_SIZE*4L) {
                // Recursive decomposition
                multisort(n/4L, &data[0], &tmp[0]);
                multisort(n/4L, &data[n/4L], &tmp[n/4L]);
                multisort(n/4L, &data[n/2L], &tmp[n/2L]);
                multisort(n/4L, &data[3L*n/4L], &tmp[3L*n/4L]);

                merge(n/4L, &data[0], &data[n/4L], &tmp[0], 0, n/2L);
                merge(n/4L, &data[n/2L], &data[3L*n/4L], &tmp[n/2L], 0,
n/2L);

                merge(n/2L, &tmp[0], &tmp[n/2L], &data[0], 0, n);
        }
    else {
                // Base case
                #pragma omp taskgroup
                {
                        #pragma omp task
                        basicsort(n, data);
                }
                //#pragma omp taskwait
        }
}
```

multisort-omp-cutoff.c

	multisort
THREAD 1.1.1	136
THREAD 1.1.2	102
THREAD 1.1.3	100
THREAD 1.1.4	167
THREAD 1.1.5	147
THREAD 1.1.6	104
THREAD 1.1.7	158
THREAD 1.1.8	110
Total	1,024
Average	128
Maximum	167
Minimum	100
StDev	25.51
Avg/Max	0.77

Cut-off strategy code:

```
void merge(long n, T left[n], T right[n], T result[n*2], long start, long
length) {
        if (length < MIN_MERGE_SIZE*2L) {
                // Base case
                basicmerge(n, left, right, result, start, length);
        } else {
                if(NUM_TASKS < CUTOFF){
                                        #pragma omp taskgroup
                                        {
                                                NUM_TASKS++;
                                                #pragma omp task
                                                merge(n, left, right,
result, start, length/2);
                                                NUM_TASKS++;
                                                #pragma omp task
                                                merge(n, left, right,
result, start + length/2, length/2);
                                        }
                                        NUM_TASKS -= 2;
                                }
                                else{
                                        merge(n, left, right, result,
start, length/2);
                                        merge(n, left, right, result, start
+ length/2, length/2);
                                }
        }
}

void multisort(long n, T data[n], T tmp[n]) {
```

```
        if (n >= MIN_SORT_SIZE*4L) {
                // Recursive decomposition
                if(NUM_TASKS < CUTOFF){
                                        #pragma omp taskgroup
                                        {
                                                NUM_TASKS++;
                                                #pragma omp task
                                                multisort(n/4L, &data[0],
&tmp[0]);
                                                NUM_TASKS++;
                                                #pragma omp task
                                                multisort(n/4L,
&data[n/4L], &tmp[n/4L]);
                                                NUM_TASKS++;
                                                #pragma omp task
                                                multisort(n/4L,
&data[n/2L], &tmp[n/2L]);
                                                NUM_TASKS++;
                                                #pragma omp task
                                                multisort(n/4L,
&data[3L*n/4L], &tmp[3L*n/4L]);
                                        }
                                        NUM_TASKS -= 4;
                                        #pragma omp taskgroup
                                        {
                                                NUM_TASKS++;
                                                #pragma omp task
                                                merge(n/4L, &data[0],
&data[n/4L], &tmp[0], 0, n/2L);
                                                NUM_TASKS++;
                                                #pragma omp task
                                                merge(n/4L, &data[n/2L],
&data[3L*n/4L], &tmp[n/2L], 0, n/2L);
                                                //#pragma omp taskwait
                                        }
                                        NUM_TASKS -= 2;

                                        NUM_TASKS++;
                                        #pragma omp task
                                        merge(n/2L, &tmp[0], &tmp[n/2L],
&data[0], 0, n);
                                }
                                else{
                                        // Recursive decomposition
                                        multisort(n/4L, &data[0], &tmp[0]);
                                        multisort(n/4L, &data[n/4L],
&tmp[n/4L]);
                                        multisort(n/4L, &data[n/2L],
&tmp[n/2L]);
                                        multisort(n/4L, &data[3L*n/4L],
&tmp[3L*n/4L]);

                                        merge(n/4L, &data[0], &data[n/4L],
&tmp[0], 0, n/2L);
```

```
                                        merge(n/4L, &data[n/2L],
&data[3L*n/4L], &tmp[n/2L], 0, n/2L);

                                        merge(n/2L, &tmp[0], &tmp[n/2L],
&data[0], 0, n);
                                }
        } else {
                if(NUM_TASKS >= CUTOFF){
                        #pragma omp task
                        basicsort(n, data);
                        #pragma omp taskwait
                }
                else{
                        basicsort(n, data);
                }
        }
}
```

multisort-omp-cutoff.c