Mergesor u ordenación por mezcla

El mergesort es un algoritmo de ordenación divide y vecerás que consiste en dividir un vector recursivamente hasta llegar a cierto tamaño, ordenar esos subvectores y posteriormente fusionarlos.

1) Descripción del algoritmo

Código secuencial

```
1  template <class U>
2  void insercion(vector<U> &a) {
3        int i, j;
4        for (i = 1; i < a.size(); i++) {
5              j = i;
6              while ((a[j] < a[j - 1]) && (j > 0)) {
7                    swap(a[j], a[j - 1]);
8                    j--;
9              }
10       }
11 }
12
13 template <class U>
14 vector<U> fusion(const vector<U> &a, const vector<U> &b) {
15       vector<U> ret;
16       int i = 0, j = 0;
17       while (i < a.size() && j < b.size()) {
18             if (a[i] < b[j]) {
19                   ret.push_back(a[i]);
20                   i++;
21             } else {
22                   ret.push_back(b[j]);
23                   j++;
24             }
25       }
26       for (int k = j; k < b.size(); k++)
27             ret.push_back(b[k]);
28       for (int k = i; k < a.size(); k++)
29             ret.push_back(a[k]);
30       return ret;
31 }
32
33 template <class U>
34 vector<U> mergesort_omp(const vector<U> &a) {
35       int vsize = a.size() / UMBRAL_MS + 1;
36       vector<U> vec[vsize];
37       for (int i = 0; i < vsize - 1; i++) {
38             vec[i].resize(UMBRAL_MS);
```

```
39        }
40        vec[vsize - 1].resize(a.size() % UMBRAL_MS);
41
42        for (int i = 0; i < a.size(); i++)
43              vec[i / UMBRAL_MS][i % UMBRAL_MS] = a[i];
44        for (int i = 0; i < vsize; i++)
45              insercion(vec[i]);
46
47        while (vsize > 1) {
48              int j=0, v2size = ceil(vsize / 2.0);
49              vector<U> v2[v2size];
50
51              for (int i = 0; i < vsize; i += 2) {
52                    auto tmp = (i != vsize - 1) ? fusion(vec[i], vec[i + 1])
   : vec[i];
53                    v2[j] = tmp;
54                    j++;
55              }
56              for (int i = 0; i < v2size; i++)
57                    vec[i] = v2[i];
58              vsize = v2size;
59        }
60        return vec[0];
61  }
```

Ejemplo de ejecución:

vector = {4,3,5,8,1,0} y UMBRALS_MS = 3
Entre las lineas 35 y 43 el vector es dividido en un array de dos vectores de tamaño 3:
{{4,3,5}, {8,1,0}}

En las lineas 44 y 45 se llama a la función de ordenación por insercion para ordenar los dos vectores:
{{3,4,5}, {0,1,8}}

Entre las lineas 47 y 60 se llama a la función fusion() que fusiona los dos vectores internos del array de vectores y se devuelve el resultado:
{0,1,3,4,5,8}

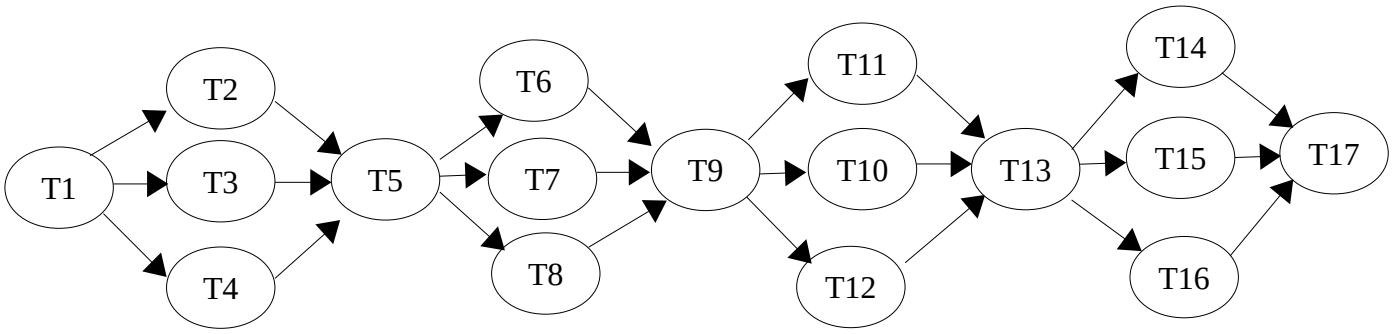
2) Paralelización

Código paralelo

```
1  template <class U>
2  void insercion(vector<U> &a) {
3        int i, j;
4        for (i = 1; i < a.size(); i++) {
5              j = i;
6              while ((a[j] < a[j - 1]) && (j > 0)) {
```

```
                    swap(a[j], a[j - 1]);
                    j--;
              }
        }
}

template <class U>
vector<U> fusion(const vector<U> &a, const vector<U> &b) {
        vector<U> ret;
        int i = 0, j = 0;
        while (i < a.size() && j < b.size()) {
              if (a[i] < b[j]) {
                    ret.push_back(a[i]);
                    i++;
              } else {
                    ret.push_back(b[j]);
                    j++;
              }
        }

        for (int k = j; k < b.size(); k++)
              ret.push_back(b[k]);
        for (int k = i; k < a.size(); k++)
              ret.push_back(a[k]);

        return ret;
}

template <class U>
vector<U> mergesort_omp(const vector<U> &a) {
        int vsize = a.size() / UMBRAL_MS + 1;
        vector<U> vec[vsize];

        #pragma omp parallel for
        for (int i = 0; i < vsize - 1; i++) {
              vec[i].resize(UMBRAL_MS);
        }
        vec[vsize - 1].resize(a.size() % UMBRAL_MS);

        #pragma omp parallel for
        for (int i = 0; i < a.size(); i++)
              vec[i / UMBRAL_MS][i % UMBRAL_MS] = a[i];

        #pragma omp parallel for
        for (int i = 0; i < vsize; i++)
              insercion(vec[i]);

```

```
54      while (vsize > 1) {
55            int j = 0, v2size = ceil(vsize / 2.0);
56            vector<U> v2[v2size];
57            #pragma omp parallel for
58            for (int i = 0; i < vsize; i += 2) {
59                  auto tmp = (i != vsize - 1) ? fusion(vec[i], vec[i + 1])
   : vec[i];
60                  #pragma omp critical
61                  {
62                        v2[j] = tmp;
63                        j++;
64                  }
65            }
66
67            #pragma omp parallel for
68            for (int i = 0; i < v2size; i++)
69                  vec[i] = v2[i];
70
71            vsize = v2size;
72      }
73      return vec[0];
74 }
```

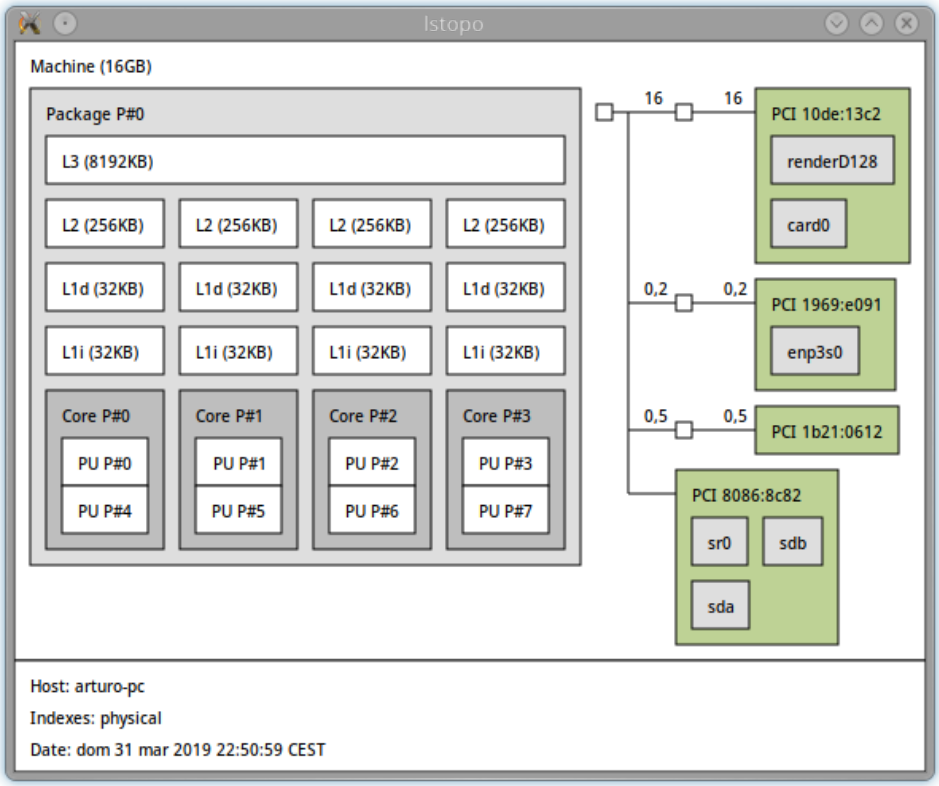


T1: Calculo del numero de vectores en el array
T2,T3,T4: Reserva de espacio de los vectores del arrat
T5: Barrera implicita de #pragma omp parallel for
T6,T7,T8: División del vector inicial en el array de vectores;
T9: Barrera implicita de #pragma omp parallel for
T10,T11,T12: Ordenación por insercion de cada vector del array
T13: Barrera implicita de #pragma omp parallel for
T14, T15, T16: Fusion de cada par de vectores en el array
T17: Barrera implicita de #pragma omp parallel for

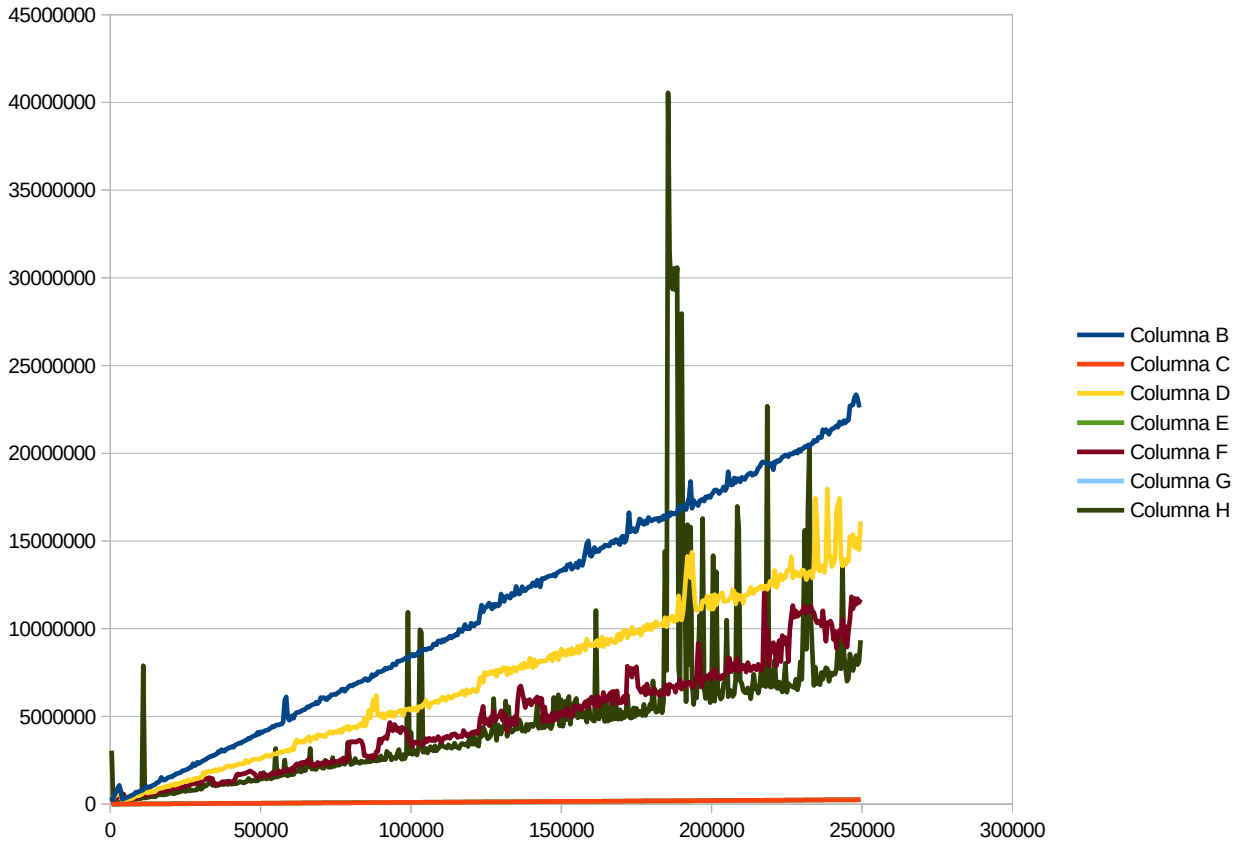
3) Evaluación de prestaciones

Descripción de la arquitectura

```
Arquitectura:                        x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:                  Little Endian
Tamaños de las direcciones:          39 bits physical, 48 bits virtual
CPU(s):                              8
Lista de la(s) CPU(s) en línea:      0-7
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:              4
«Socket(s)»                          1
Modo(s) NUMA:                        1
ID de fabricante:                    GenuineIntel
Familia de CPU:                      6
Modelo:                              60
Nombre del modelo:                   Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz
Revisión:                            3
CPU MHz:                             1738.690
CPU MHz máx.:                        4400,0000
CPU MHz mín.:                        800,0000
BogoMIPS:                            8003.47
Virtualización:                      VT-x
Caché L1d:                           32K
Caché L1i:                           32K
Caché L2:                            256K
Caché L3:                            8192K
CPU(s) del nodo NUMA 0:              0-7
Indicadores:                         fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xt
opology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est
tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt
tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm cpuid_fault invpcid_single
pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase
tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts
flush_l1d
```



Escalabilidad fuerte



Para mas informacion mirar mergesort.ods