k-means algorithm for clustering

Comparing the parallelization using MPI and OMP

Marcio Sampaio Francisco Siles

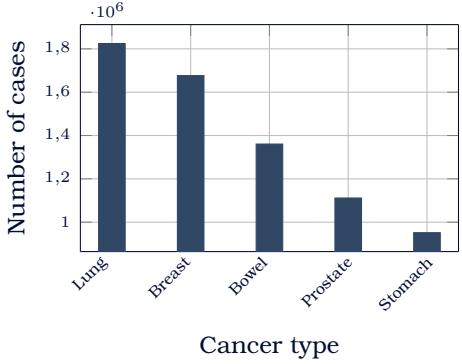
Brasil & Costa Rica

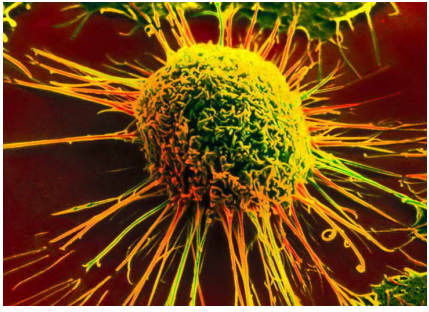

PRI | Cancer's Global Footprint
Produced by Pulitzer Center / PRI's The World
Data Notes
« View series
Costa Rica
Cancer cases annually per 100,000 people
176.3
HDI* rank
48 of 179
GDP per capita (USD)
$6,597
Health expenditure per capita (USD)
$618
Life expectancy
79 years
Sources: GLOBOCAN 2008 v2.0, World Bank, UNDP (2008)
*Human Development Index (HDI) is a combined indicator of life expectancy, education, and income.
SHARE +
Motivation
Incidence
Mortality
All
Breast
Cervical
Liver
Lung
Prostate
Stomach
GLOBAL CANCER INCIDENCE
« Previous | Next »
New cancer cases annually per 100,000 people (age-adjusted)
> 0
> 60
> 120
> 180
> 240
> 300
Source: GLOBOCAN 2008 v2.0
ICTP

Cancer Incidence and Mortality

Complex Problem

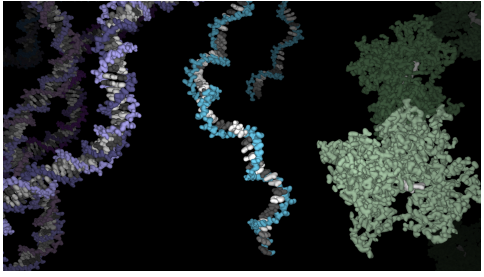
Worldwide:

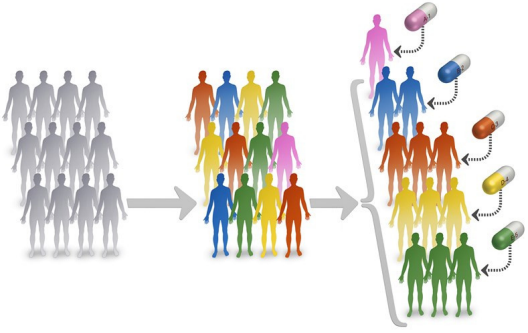




- 2012: 14.1M, 8.2M
- 2030: 23.6M, 13.7M

Personalized Therapy

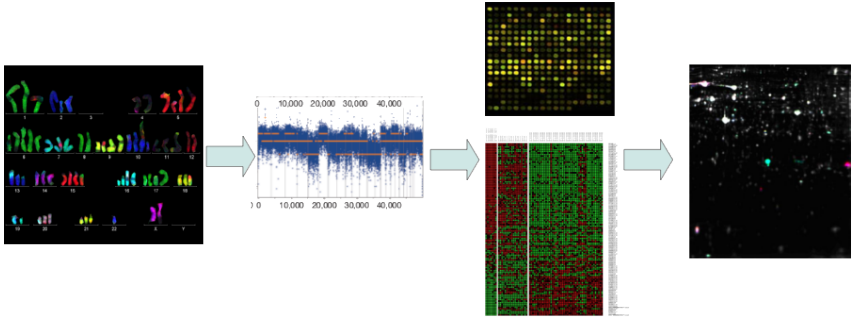




in-silico → in-vitro → in-vivo

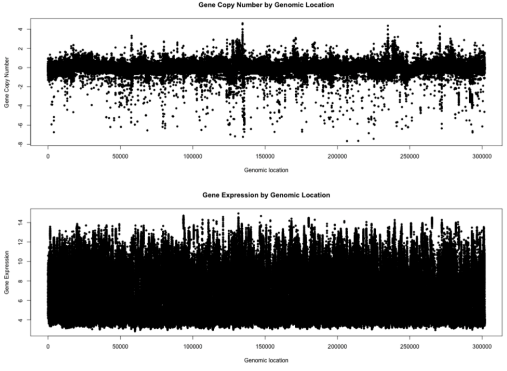

Biocomputational
Platform

Biocomputational Platform

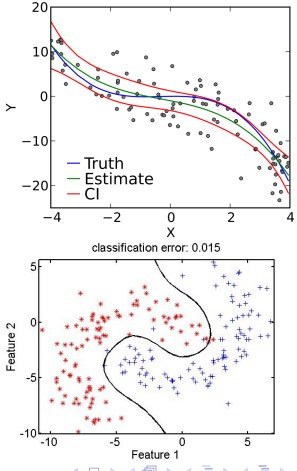


Bio-Plataform → Chemosensitivity Prediction → New Targets

CCLE, NCBI

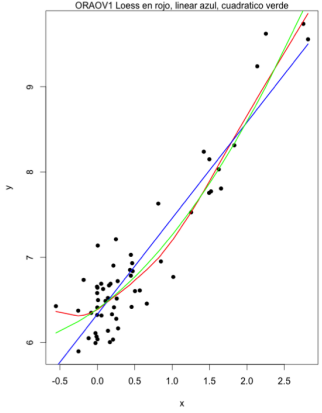


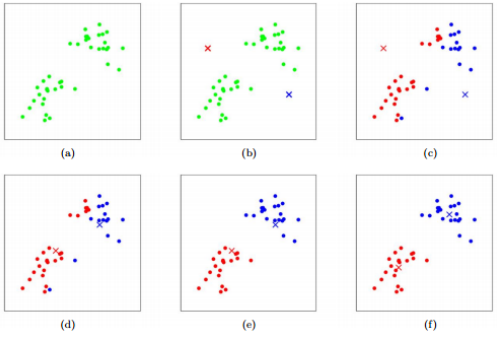
Pattern Recognition



Models

k-Means





Models

k-Means

```
for k=0, ..., N
 for s=0, ..., M
  for t=0,..., T
   for n=0,..., N
    for p=0, ..., P
```

```
for(t=0; t<Tmax; ++t){
  /*************************************************
  * Calculation step
  *************************************************
  /*Calculate the distances*/
  /*initially only squared Euclidean distance is c
  for(i=0; i<N; ++i){
    for(j=0; j<K; ++j){
      /*Distance between i-th Sample and j-th Centr
      distance = 0.0;
      d = 0.0;
      for(q=0; q<P; ++q){
        d = Samples[i*P+q] - Centroids[j*P+q];
        distance += d*d;
      }
      Distances[i*K+j] = distance;
    }
  }
```

Models

k-Means

```
for k=0, ..., N
 for s=0, ..., M
  for t=0,..., T
   for n=0,..., N
    for p=0, ..., P
```

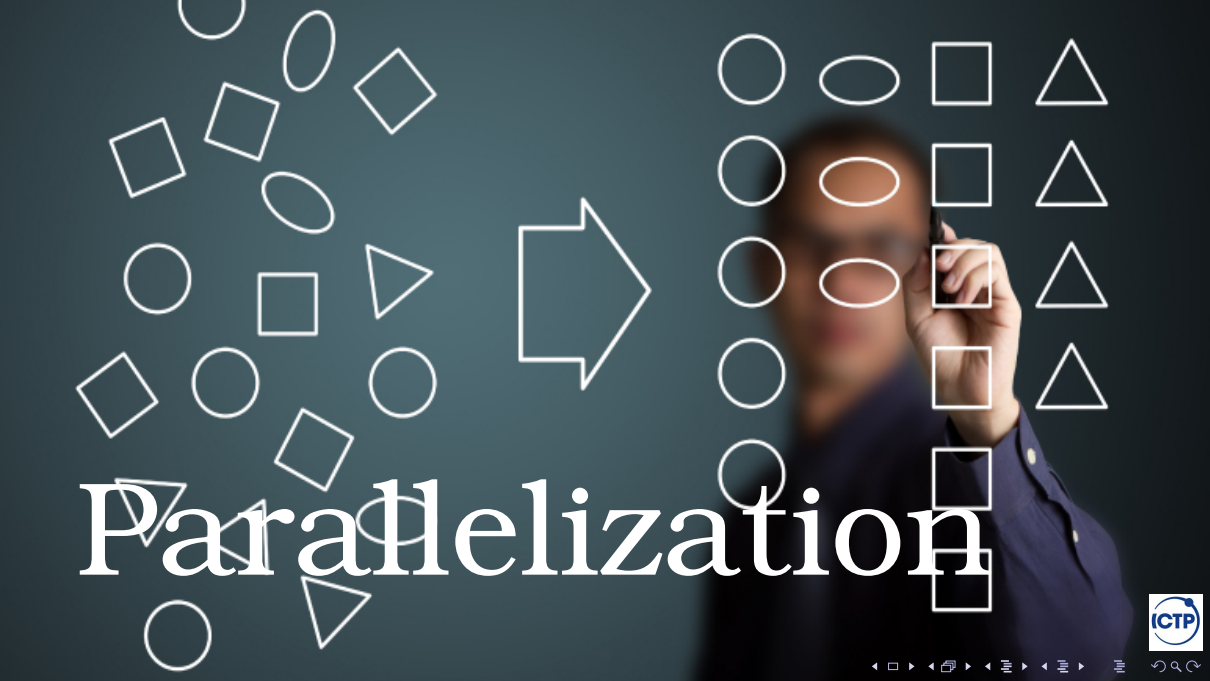
```
/*Do the cluster assignment*/
for(i=0; i<N; ++i){ //assign each sample to the n
  r = 0;
  for(j=1; j<K; ++j){
    if( Distances[i*K+j] < Distances[i*K+r] ){
      r = j;
    }
  }
  new_indexes[i] = r;
}
```

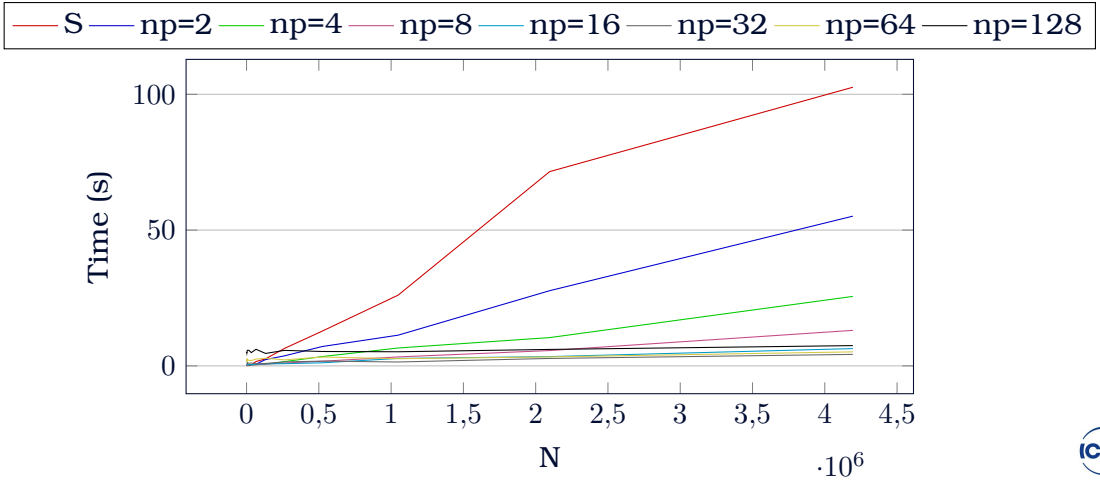
Models

k-Means

```
for k=0, ..., N
 for s=0, ..., M
  for t=0,..., T
   for n=0,..., N
    for p=0, ..., P
```

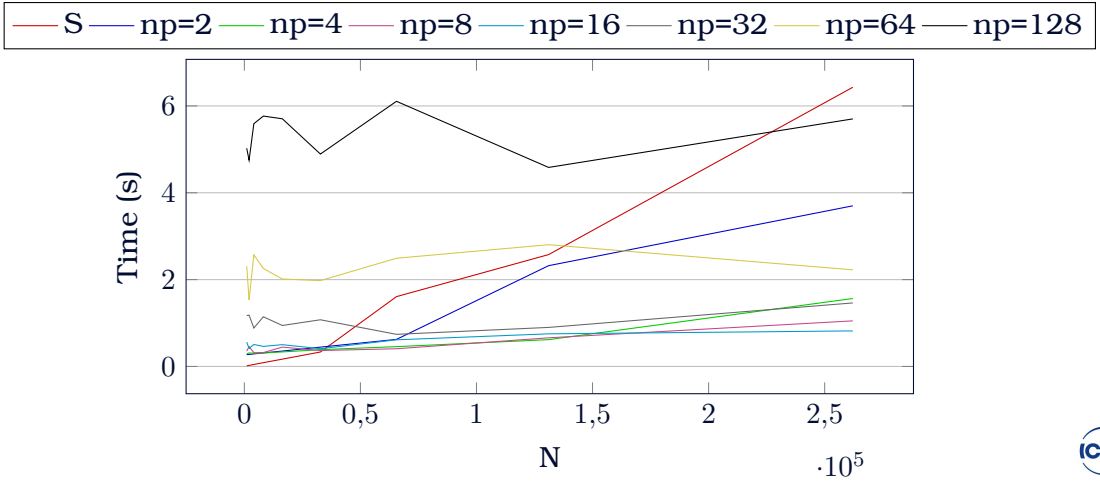
```
/*Recalculate the centroids*/
memset(Centroids,0,K*P*sizeof(double));
memset(cardinality,0,K*sizeof(int));

for(i=0;i<N;++i){
  ++cardinality[new_indexes[i]];
  for(j=0;j<P;++j){
    Centroids[new_indexes[i]*P+j] += Samples[i*P+
  }
}
for(q=0;q<K;++q){
  for(j=0;j<P;++j){
    Centroids[q*P+j] /= cardinality[q];
  }
}
```


Parallelization


S
np=2
np=4
np=8
np=16
np=32
np=64
np=128
Time (s)
100
50
0
0
0,5
1
1,5
2
2,5
3
3,5
4
4,5
N
$\cdot 10^6$

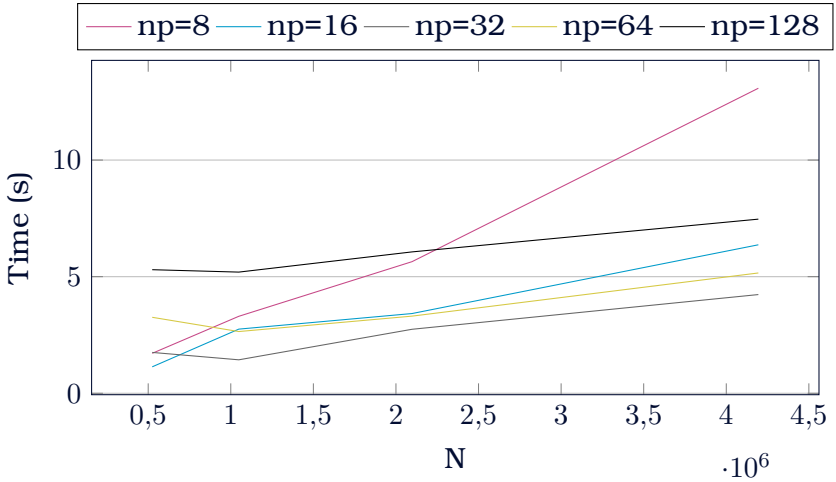
Parallelization

MPI: $N <<$



Parallelization

MPI: only Parallel, $>> N$



Parallelization

MPI code

```
/*communicate the centroids to P0 to form the Centroids matrix and communic
double* sending_buffer = (double*) malloc( k * P * sizeof(double) ); //temp
memcpy(sending_buffer,Centroids+rank*k*P,sizeof(double)*k*P);
MPI_Allgather((void*)(sending_buffer),k*P, \
    MPI_DOUBLE, (void*)(Centroids),k*P,MPI_DOUBLE, MPI_COMM_WORLD);
free(sending_buffer);

/*get the global cardinality calculation*/
int* sending_buffer2 = (int*) malloc( K * sizeof(int) ); //temporary sending
memcpy(sending_buffer2,cardinality,sizeof(int)*K);
MPI_Allreduce((void*)(sending_buffer2),(void*)(cardinality),\
        K,MPI_INT,MPI_SUM,MPI_COMM_WORLD); //try to use temporary buffers t
free(sending_buffer2);

sending_buffer = (double*) malloc( K*P * sizeof(double) ); //temporary send
memcpy(sending_buffer,Centroids,sizeof(double)*K*P);
MPI_Allreduce((void*)(sending_buffer),(void*)(Centroids),K*P,\
        MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
free(sending_buffer);
```

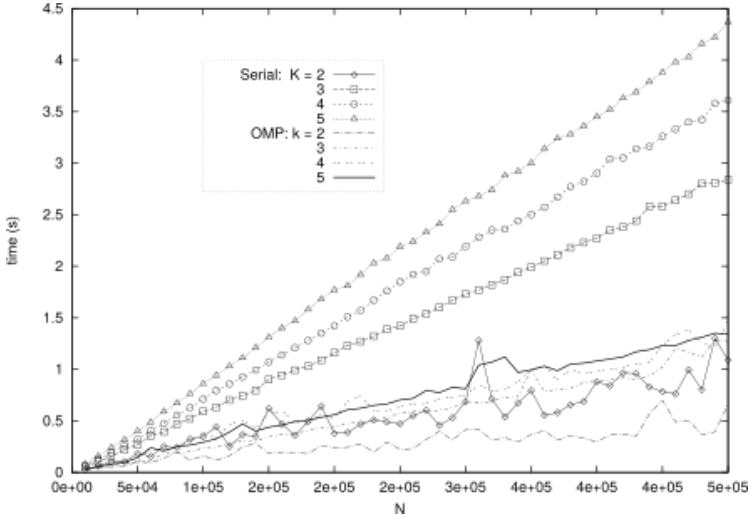
Parallelization

OMP code

```
#pragma omp parallel for reduction(+:distance) private (i, j, q)\
    shared(Distances, Samples, Centroids)
for(i=0; i<N; ++i){
    for(j=0; j<K; ++j){
        /*Distance between i-th Sample and j-th Centroid*/
        distance = 0.0;
        d = 0.0;
        for(q=0; q<P; ++q){
            d = Samples[i*P+q] - Centroids[j*P+q];
            distance += d*d;
        }
        Distances[K*i+j] = distance;
    }
}
```

Parallelization

OMP results




Muchas gracias!

Muito obrigado!