Porte para OpenMP do Algoritmo Streamcluster

Angelo Leite
Angelo Marcelino
Jhonat Herberson
Maurício Thiago

Metodologia para reescrever o código com OpenMP

- Progressão da análise e entendimento do código:
  - Identificação das principais funções do código (Primeira Apresentação)
  - Análise de problemas de otimização (Segunda Apresentação)
  - Identificação linhas opcionais do código
- Implementação limpa:
  - Serialização do código
  - Remoção do uso de bibliotecas de paralelização (TBB, Pthreads)
- *Hands-on implementation*:
  - Escolha da função a ser abordada
  - Comparação entre Pthreads e OpenMP
  - Paralelização de partes dessa função
  - Comparação de tempos de execução com código serial

Identificação linhas opcionais do código

Objetivos:

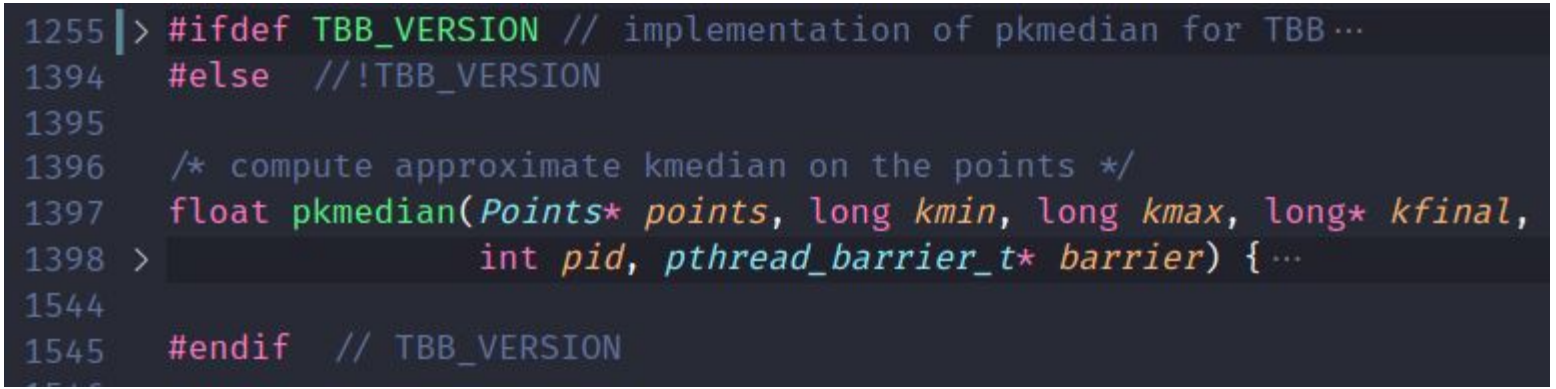
- Redução do código (1950 linhas)
- Melhor análise
- Serialização

```
#include <fstream>
#include <iostream>

#ifdef ENABLE_THREADS
#include <pthread.h>

#include "parsec_barrier.hpp"
#endif

#ifdef TBB_VERSION
#define TBB_STEALER (tbb::task_scheduler_init::occ_stealer)
#define NUM_DIVISIONS (nproc)
#include "tbb/blocked_range.h"
#include "tbb/cache_aligned_allocator.h"
#include "tbb/parallel_for.h"
#include "tbb/parallel_reduce.h"
#include "tbb/task_scheduler_init.h"
using namespace tbb;
#endif

#ifdef ENABLE_PARSEC_HOOKS
#include <hooks.h>
#endif
```

Identificação linhas opcionais do código

139 linhas -->

146 linhas -->



Serialização do código

Objetivos:

- Comparação limpa
- Clareza de algoritmo
- Redução de linhas (815 ou 42%)

Contrapartida:

- Perda dos pontos de paralelização

```
void localSearch(Points* points, long kmin, long kmax, long* kfinal) {
    pthread_barrier_t barrier;
    pthread_t* threads = new pthread_t[nproc];
    pkmedian_arg_t* arg = new pkmedian_arg_t[nproc];

#ifdef ENABLE_THREADS
    pthread_barrier_init(&barrier, NULL, nproc);
#endif
    for (int i = 0; i < nproc; i++) {
        arg[i].points = points;
        arg[i].kmin = kmin;
        arg[i].kmax = kmax;
        arg[i].pid = i;
        arg[i].kfinal = kfinal;

        arg[i].barrier = &barrier;
#ifdef ENABLE_THREADS
        pthread_create(threads + i, NULL, localSearchSub, (void*)&arg[i]);
#else
        localSearchSub(&arg[0]);
#endif
    }
```

```
void localSearch( Points* points, long kmin, long kmax, long* kfinal) {
    pkmedian_arg_t arg;

    arg.points = points;
    arg.kmin = kmin;
    arg.kmax = kmax;
    arg.kfinal = kfinal;

    localSearchSub(&arg);

}
```

Escolha da função a ser abordada

Dados anteriores:

- `pgain()`

Testes (loops):

- `omp_get_wtime()`
- 70% tempo em um loop

Teoria (de um loop):

- Speedup ~= p ⇒ -35% tempo de execução para nproc = 2

```
./streamcluster 50 100 250 10000 1 10000 myin myout 2
Each sample counts as 0.01 seconds.
 %   cumulative   self              self     total
time   seconds   seconds    calls   s/call   s/call  name
82.98     16.75    16.75    24508     0.00     0.00  pgain(lo
16.76     20.13     3.38   251412     0.00     0.00  parsec_b
 0.30     20.19     0.06       14     0.00     0.00  pspeedy(
```

```
for ( i = 0; i < points->num; i++ ) {
  float x_cost = dist(points->p[i], points->p[x], points->dim) * points->p[i].weight;
  float current_cost = points->p[i].cost;

  if ( x_cost < current_cost ) {
    switch_membership[i] = 1;
    cost_of_opening_x += x_cost - current_cost;

  } else {
    int assign = points->p[i].assign;
    lower[center_table[assign]] += current_cost - x_cost;
  }
}
```

Comparação entre Pthreads e OpenMP

Threads “Macro”:

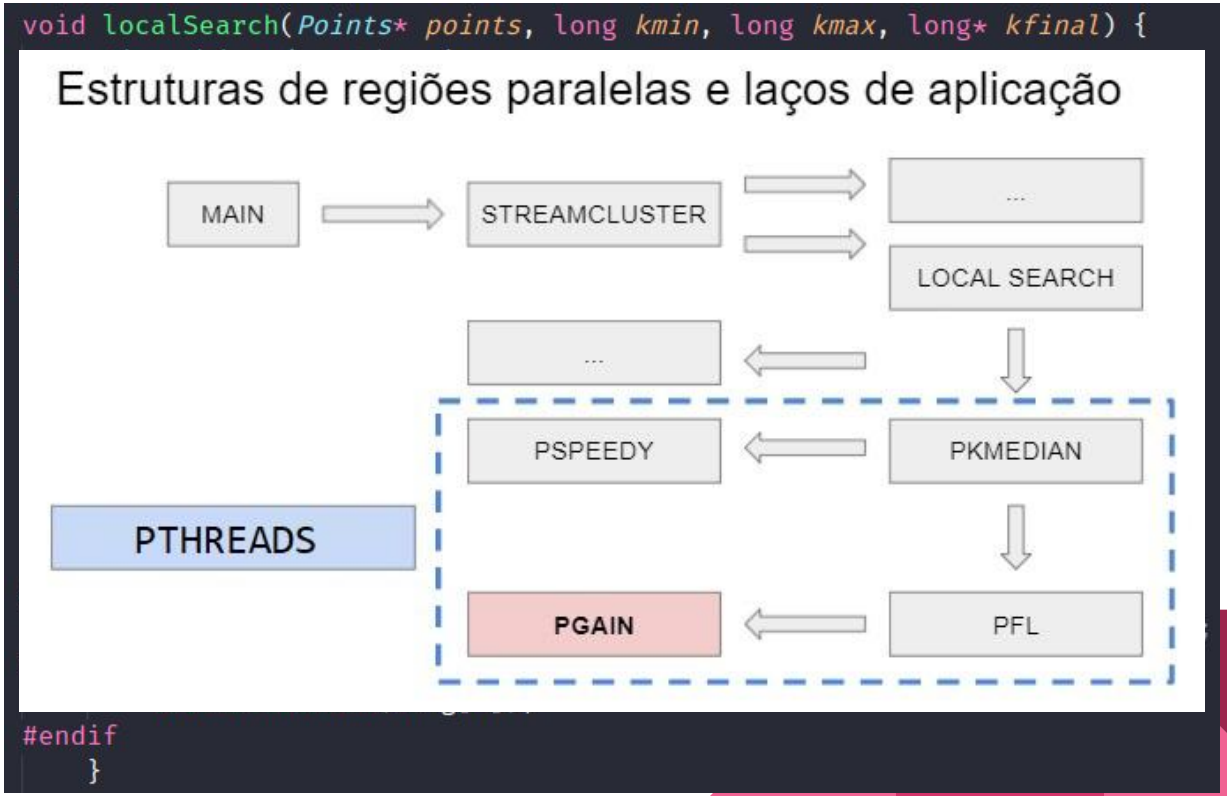
- `localSearchSub()`
- Serialização das threads
- Gerenciamento de barreiras

```
void localSearch(Points* points, long kmin, long kmax, long* kfinal) {
    pthread_barrier_t barrier;
    pthread_t* threads = new pthread_t[nproc];
    pkmedian_arg_t* arg = new pkmedian_arg_t[nproc];

#ifdef ENABLE_THREADS
    pthread_barrier_init(&barrier, NULL, nproc);
#endif
    for (int i = 0; i < nproc; i++) {
        arg[i].points = points;
        arg[i].kmin = kmin;
        arg[i].kmax = kmax;
        arg[i].pid = i;
        arg[i].kfinal = kfinal;

        arg[i].barrier = &barrier;
#ifdef ENABLE_THREADS
        pthread_create(threads + i, NULL, localSearchSub, (void*)&arg[i]);
#else
        localSearchSub(&arg[0]);
#endif
    }
```

Comparação entre Pthreads e OpenMP

Threads “Macro”:

- `localSearchSub()`
- Serialização das threads
- Gerenciamento de barreiras



Comparação entre Pthreads e OpenMP

Threads “Micro”:

- Remoção de barreiras
- Criação de Threads somente quando necessário

Exemplo ao lado:

- Fluxo de controle
- Função `dist()`
- Diferença no `for`
- `schedule(static, total_iteration/ thread_count)`

```
for ( i = 0; i < points->num; i++ ) {
  float x_cost = dist(points->p[i], points->p[x], points->dim) * points->p[i].weight;
  float current_cost = points->p[i].cost;

  if ( x_cost < current_cost ) {
    switch_membership[i] = 1;
    cost_of_opening_x += x_cost - current_cost;

  } else {
    int assign = points->p[i].assign;
    lower[center_table[assign]] += current_cost - x_cost;
  }
}
```

```
double initime = omp_get_wtime();

float* x_cost_arr = (float*)malloc(points→num*sizeof(float));
#pragma omp parallel num_threads(1)
  #pragma omp for
    for ( i = 0; i < points→num; i++ ) {
      x_cost_arr[i] =  dist(points→p[i], points→p[x], points→dim);
      x_cost_arr[i] *= points→p[i].weight;
    }

double finishtime = omp_get_wtime();
```

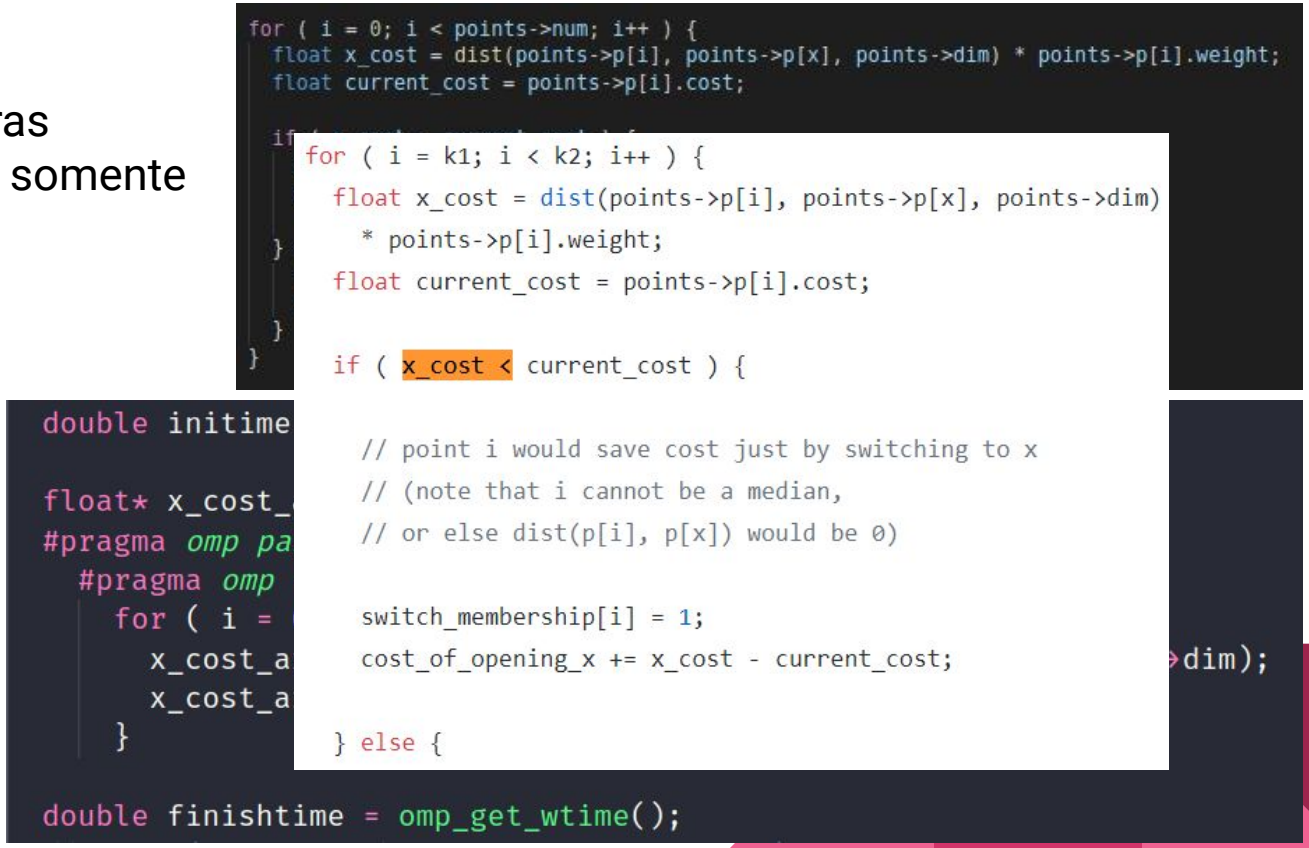
Comparação entre Pthreads e OpenMP

Threads “Micro”:

- Remoção de barreiras
- Criação de Threads somente quando necessário

Exemplo ao lado:

- Fluxo de controle
- Função `dist()`
- Diferença no `for`
- `schedule(static, total_iteration/ thread_count)`



Comparação de tempos

./streamcluster 10 20 200 1000 1 1000 myin myout 1

		N		
		1000	10000	100000
P	sequencial	0,404	7,167	81,676
	1	0,363	7,08	80,442
	2	0,237	5,075	57,416
	3	0,624	6,379	65,64
	4	0,809	5,773	64,484

Testes:

- Comparação com sequencial
- 2-thread processor

Resultados:

- ~30% de redução para nproc = 2

Mudanças futuras:

- Testes em máquinas com mais processadores disponíveis.
- Análise de filosofia do código.