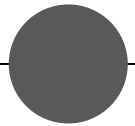
# Trabalho OpenMP - Simulated Annealing

Gabriel Pereira e Arthur Hendges



## Algoritmo - Simulated Annealing

- Heurística probabilística;
- Baseada no processo da metalúrgica;
- Explora soluções vizinhas podendo aceitar soluções piores com certa probabilidade;
- Energia/custo

## Paralelização

- Cálculo da energia
- Múltiplas instâncias do algoritmo com seeds diferentes

## Pontos de paralelização

- Distribuição da soma entre threads;
- Eficaz para muitos pontos;

```
double state::E() {
    double dist = 0;
    int n = points.size();
    #pragma omp parallel for reduction(+:dist)
    for (int i = 0; i < n; i++)
        dist += euclidean(points[i], points[(i+1)%n]);
    return dist;
}
```

## Pontos de paralelização

- Execução independente por seed;
- Exploração de múltiplos mínimos locais;

```cpp
#pragma omp parallel for
for (int i = 0; i < (int)seeds.size(); i++) {
    int seed = seeds[i];

    auto start = chrono::high_resolution_clock::now();

    state s(params.points, seed);
    auto result = simAnneal(params.start_temp, params.decay_rate, seed, s);

    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double, milli> elapsed = end - start;

    #pragma omp critical
    {
        out << "Seed=" << seed << "\n";
        out << "Input_size=" << params.points.size() << "\n";
        out << "Best_energy=" << result.first << "\n";
        auto last = result.second.points.back();
        auto fst = result.second.points.front();
        out << "First_point=(" << fst.first << ", " << fst.second << ")\n";
        out << "Last_point=(" << last.first << ", " << last.second << ")\n";
        out << "Execution_time_(ms)=" << elapsed.count() << "\n";
        out << "-------------------------\n";
    }
}
```

# Resultados

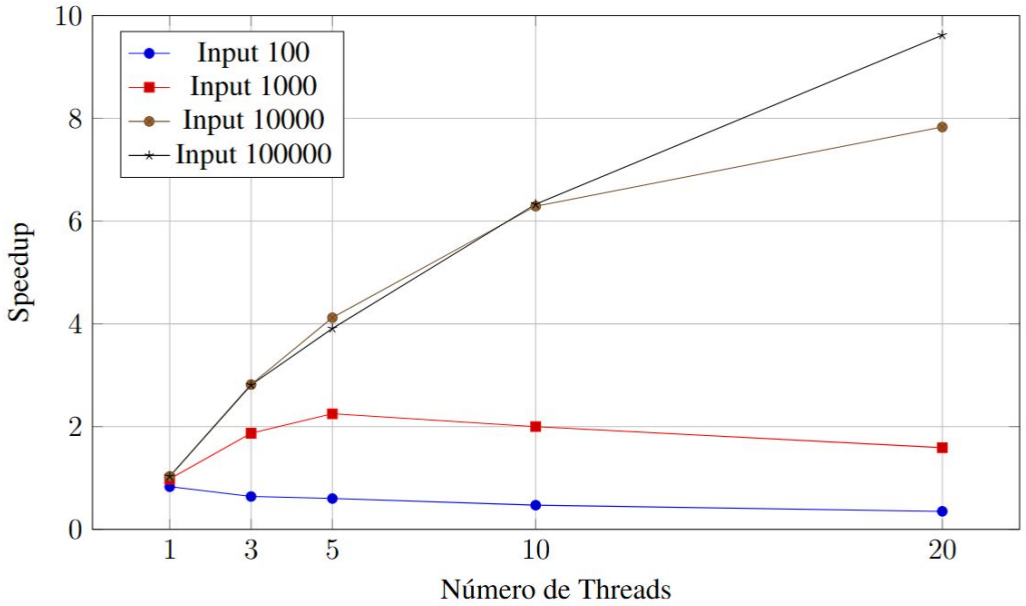



**Figura 1. Speedup obtido em função do número de threads para diferentes tamanhos de entrada.**

# Resultados

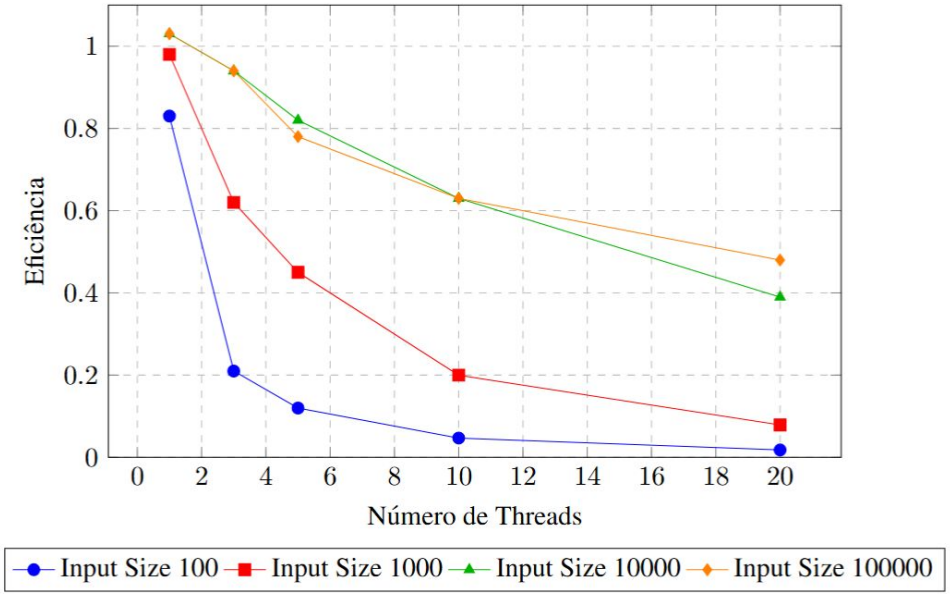



**Figura 2. Eficiência do algoritmo Simulated Annealing paralelo em relação à versão serial.**

# Resultados

**Tabela 3. Performance do algoritmo Simulated Annealing com 1 thread por seed.**

| Input Size | Threads | Average Execution Time (ms) | Speedup |
|---|---|---|---|
| 100 | 1 | 67.50 | 1.00 |
| 100 | 3 | 67.68 | 0.997 |
| 100 | 5 | 70.95 | 0.951 |
| 100 | 10 | 77.05 | 0.876 |
| 100 | 20 | 77.54 | 0.871 |
| 1000 | 1 | 405.15 | 1.00 |
| 1000 | 3 | 409.28 | 0.990 |
| 1000 | 5 | 419.18 | 0.966 |
| 1000 | 10 | 470.28 | 0.862 |
| 1000 | 20 | 469.55 | 0.864 |
| 10000 | 1 | 4766.44 | 1.00 |
| 10000 | 3 | 4794.95 | 0.995 |
| 10000 | 5 | 4983.43 | 0.957 |
| 10000 | 10 | 5497.83 | 0.867 |
| 10000 | 20 | 5495.28 | 0.867 |
| 100000 | 1 | 48028.2 | 1.00 |
| 100000 | 3 | 48343.7 | 0.994 |
| 100000 | 5 | 49836.4 | 0.964 |
| 100000 | 10 | 55510.7 | 0.865 |
| 100000 | 20 | 55371.9 | 0.868 |

## Considerações

- Quanto maior o N mais cache bound o algoritmo ficou;
- Quanto maior o número de threads mais memory bound;

# Obrigado!