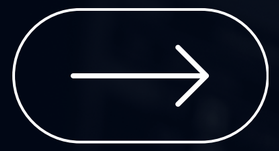
Programação Modelo Híbrido Paralela.

Gustavo Santos



Índice.

Computador:

Processador: Ryzen 7 2700 (8 cores/16 threads) @ 3,8Ghz

Memória: 2 x 8Gb DDR4 @ 2933Mhz + 2x 4Gb DDR4 @2933Mhz

OS: Fedora 38

Kernel: 6.3.12

Modificações MPI
+
OpenMP

```c
void exchangeRows(bool **canva, int apa, int numProcess, int rank) {
    int rowsPerProcess = apa / numProcess;
    int topNeighbor = (rank - 1 + numProcess) % numProcess;
    int bottomNeighbor = (rank + 1) % numProcess;

    MPI_Request request[4];
    MPI_Status status[4];

    bool *topRowRecv = (bool *)malloc(apa * sizeof(bool));
    bool *bottomRowRecv = (bool *)malloc(apa * sizeof(bool));

    int startRowRank = (rank) * rowsPerProcess;
    if(rank == 0){
        startRowRank = 1;
    }
    int endRowRank = (rank + 1) * rowsPerProcess;
    if(bottomNeighbor == 0){
        endRowRank -= 1;
    }

    MPI_Isend(canva[startRowRank], apa, MPI_C_BOOL, topNeighbor, 0, MPI_COMM_WORLD, &request[0]);
    MPI_Irecv(bottomRowRecv, apa, MPI_C_BOOL, topNeighbor, 0, MPI_COMM_WORLD, &request[1]);
    MPI_Isend(canva[endRowRank], apa, MPI_C_BOOL, bottomNeighbor, 0, MPI_COMM_WORLD, &request[2]);
    MPI_Irecv(topRowRecv, apa, MPI_C_BOOL, bottomNeighbor, 0, MPI_COMM_WORLD, &request[3]);

    MPI_Waitall(4, request, status);
    if (rank > 0) {
        memcpy(canva[startRowRank - 1], topRowRecv, apa * sizeof(bool));
    }
    if (rank < numProcess - 1) {
        memcpy(canva[endRowRank + 1], bottomRowRecv, apa * sizeof(bool));
    }

    free(topRowRecv);
    free(bottomRowRecv);
}
```

Modificações MPI + OpenMp

```
void processCanvas(bool ** canvaOld, bool ** canva, int apa, int numProcess, int rank) {

    int rowsPerProcess = apa / numProcess;
    int startRow = rank * rowsPerProcess;
    int endRow = startRow + rowsPerProcess;

    if(rank == 0){
        startRow = 1;
    }

    if(rank == numProcess - 1){
        endRow = apa - 1;
    }

    exchangeRows(canvaOld, apa, numProcess, rank);

    processCanvasThread(canvaOld, canva, apa, startRow, endRow);

}
```

Modificações MPI + OpenMp

```
11
12   #define NumThreadsOpenMP 2
13
```

```
void  initializeRows(bool ** canva, bool ** canvaOld, int startRow, int endRow, int apa){

    int number = 0;
    int seed = rand();

    #pragma omp parallel for num_threads(NumThreadsOpenMP) shared(canvaOld, canva) private(number, seed)
    for(int i = startRow; i < endRow; i++){
        for(int j = 0; j < apa; j ++){
            number = rand_r(&seed);
            canva[i][j] = (number % 2 == 0);
            canvaOld[i][j] = (number % 2 == 0);
        }
    }
}
```

```
void processCanvasThread(bool ** canvaOld, bool ** canva, int apa, int startRow, int endRow) {

    int neighbors = 0;

    #pragma omp parallel for num_threads(NumThreadsOpenMP) shared(canvaOld, canva) private(neighbors)
    for (int i = startRow; i < endRow; i++) {
        for (int j = 0; j < apa; j++) {
            neighbors = countNeighbors(canvaOld, i, j, apa);
            canva[i][j] = ((canvaOld[i][j] == 1) && ((neighbors == 2) || (neighbors == 3))) + ((canvaOld[i][j] == 0) && ((neighbors == 3)));
        }
    }
}
```

PARÂMETROS

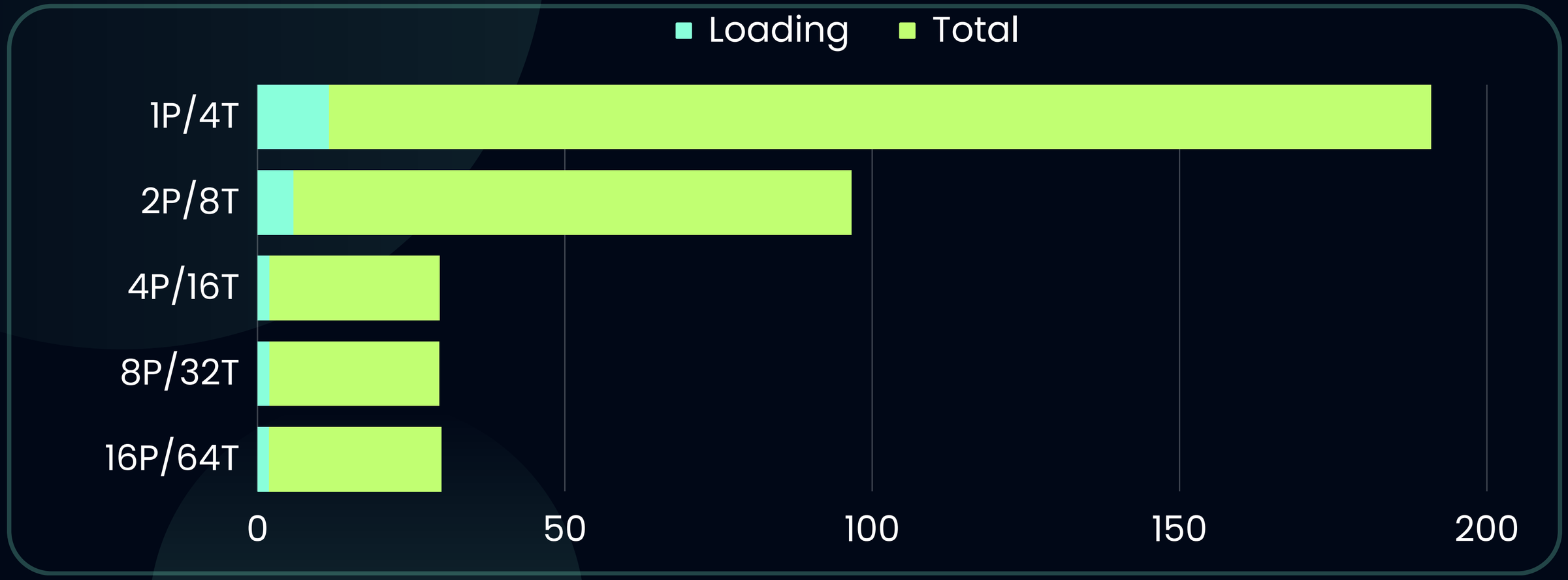
Threads: 4

Iterações: 3

Quantidade:
2.500.000.000 Células
(50.000 x 50.000)

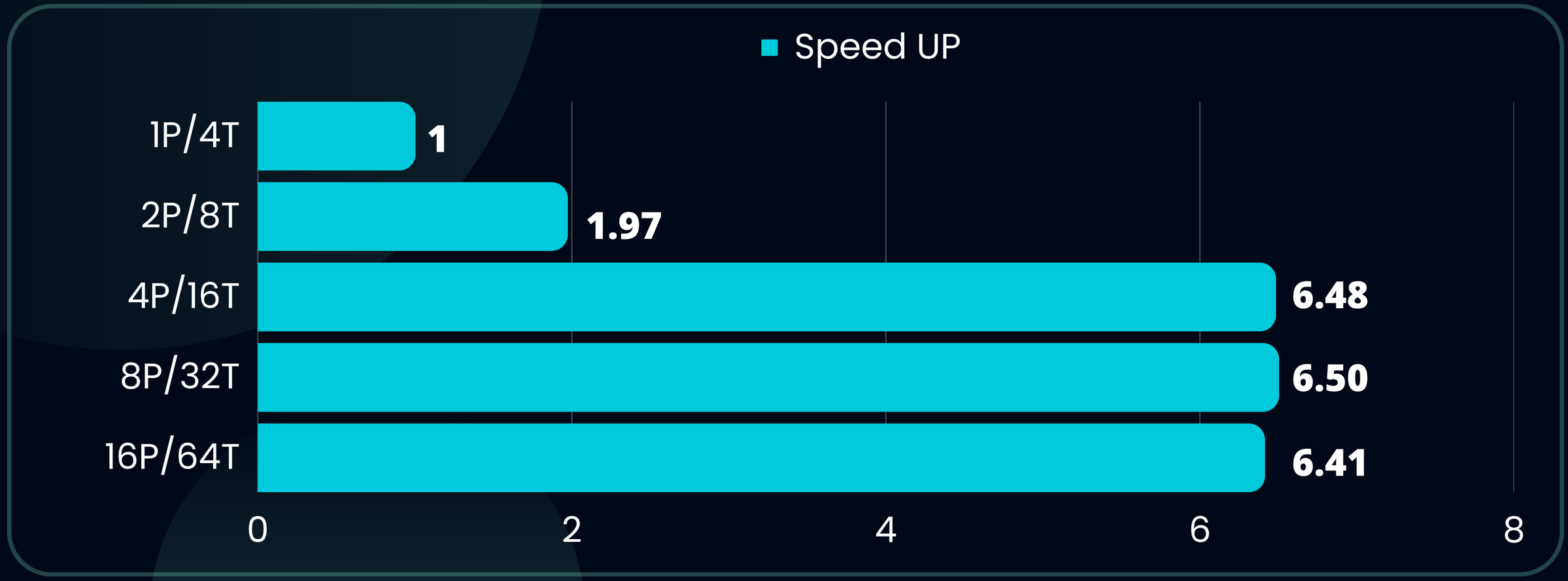
Tempo por quantidade de Processos.

Em segundos.



Speed MPI + OpenMP.

CPU de 8 cores/16 threads.



PARÂMETROS

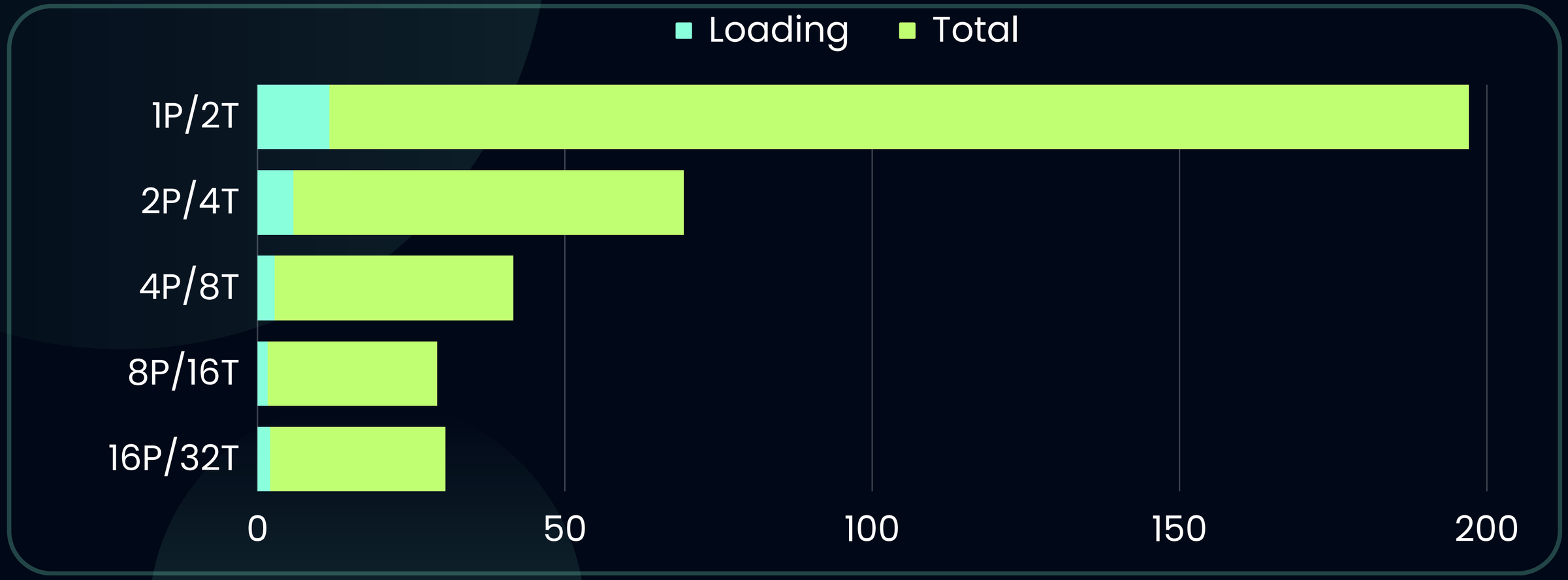
Threads: 2

Iterações: 3

Quantidade:
2.500.000.000 Células
(50.000 x 50.000)

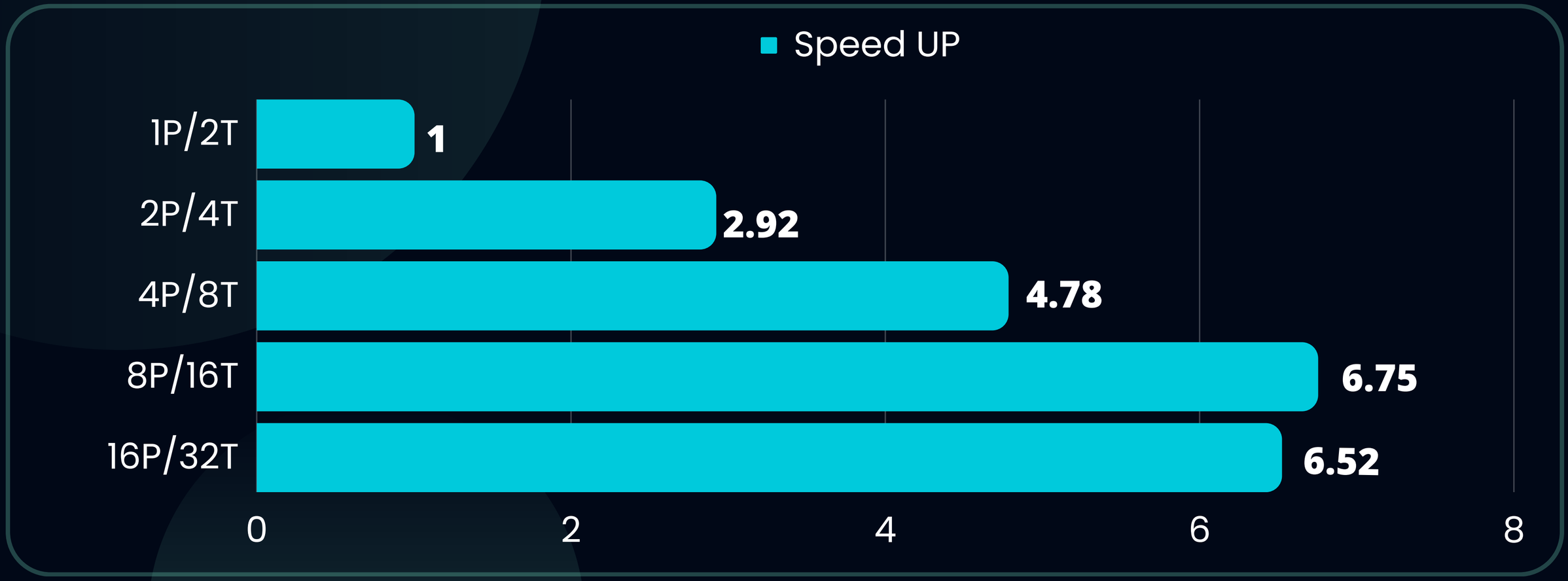
Tempo por quantidade de Processos.

Em segundos.



Speed MPI + OpenMP.

CPU de 8 cores/16 threads.



PARÂMETROS

Como as comunicações só ocorrem a cada iteração, foi aumentada a quantidade de iterações e reduzido o tamanho do problema.

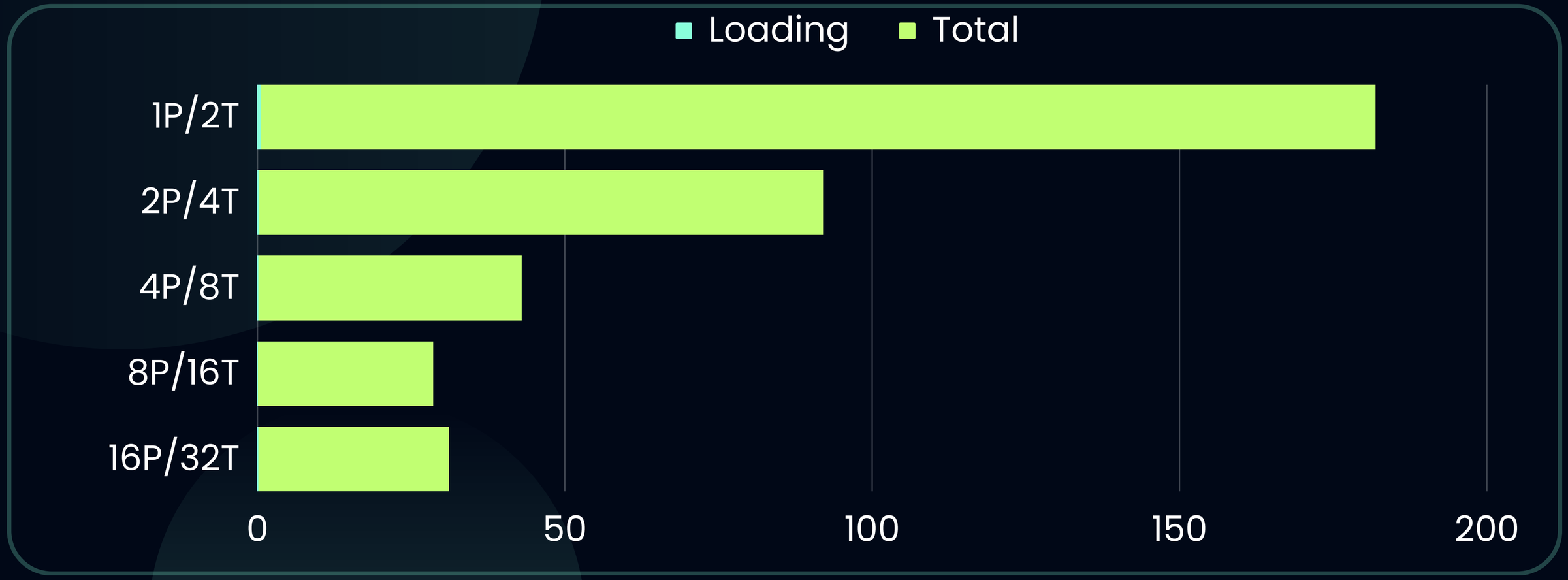
Threads: 2

Iterações: 100

Quantidade:
100.000.000 Células
(10.000 x 10.000)

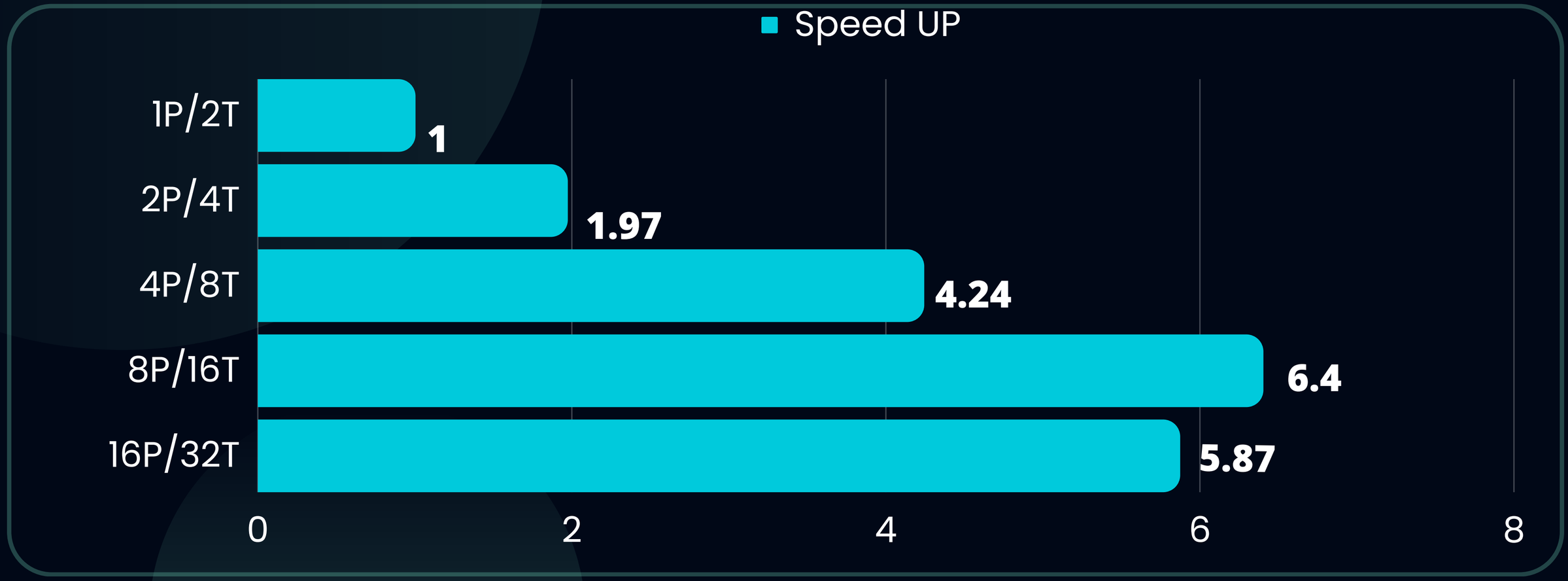
Tempo por quantidade de Processos.

Em segundos.



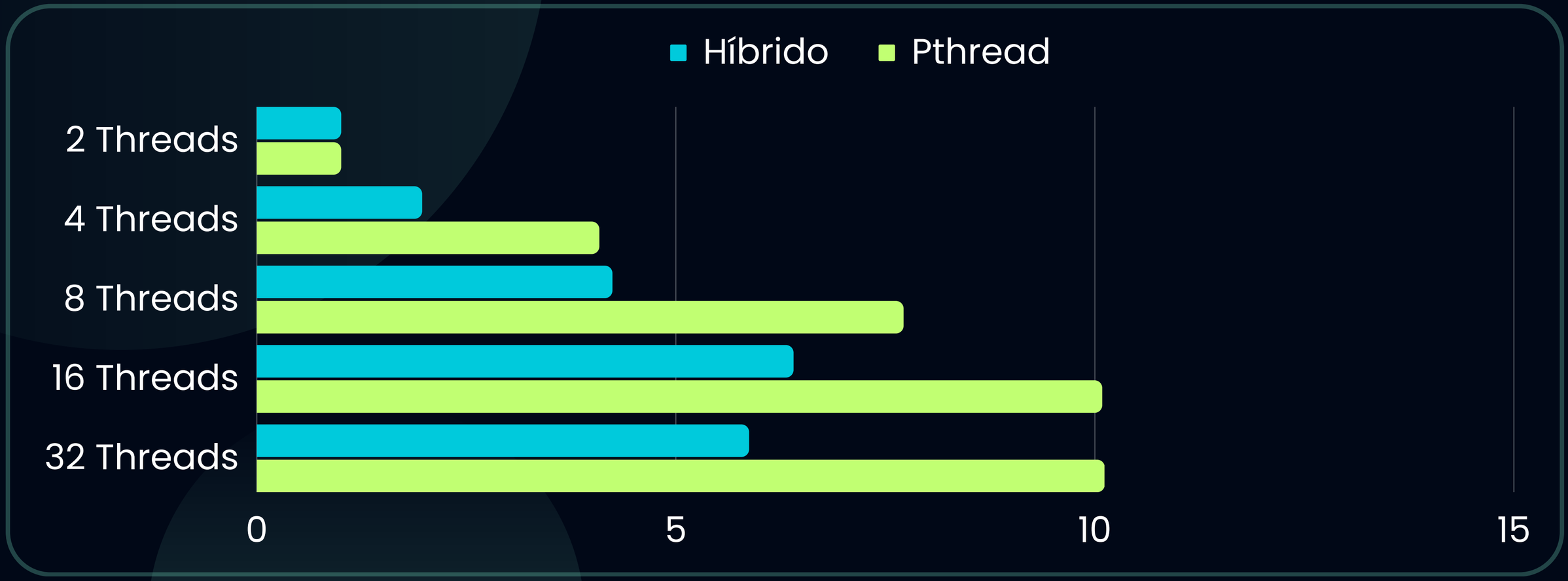
Speed MPI + OpenMP.

CPU de 8 cores/16 threads.



Comparação Híbrido vs Pthread.

CPU de 8 cores/16 threads.



CONCLUSÃO

Comunicar é caro.

Obrigado.