Implementações de algorítmos de contagem de números primos naive e Bag of tasks para diferentes instruções de comunicação ponto-a-ponto de MPI.

Victor Abi-Ramia Antonio Rachide

May 15, 2024

1 Introdução

Este trabalho tem como objetivo estudar as métricas de speedup e eficiência para diferentes implementações de algorítmos de contagem de números primos usando abordagem naive e Bag of tasks para diferentes instruções de comunicação de envio e recebimento ponto-a-ponto de MPI. 14 casos são estudados no presente trabalho. Sendo 8 implementações usando abordagem Naive e 6 para Bag of tasks. Foram utilizados os modos padrão, pronto e síncrono de envio bloqueantes, junto com o modo padrão não-bloqueante. Além de rotinas bloqueantes e não bloqueantes de recebimento ponto a ponto para a maioria dos modos de envio empregados. A descrição dos algoritmos Naive e Bag of tasks para o problema de contagem de números primos é apresentada na seção 2. Na seção 3 e 4, os resultados são demonstrados. Os tempos de execução constam na seção 5.

2 Algoritmos de contagem de números primos

2.1 Abordagem Naive

O algoritmo baseado em método Naive para contagem de números primos envolve na divisão de um intervalo de números a serem verificados entre diferentes processos. Cada processo verifica, em seu intervalo atribuído, a quantidade de números primos. Então, os resultados do números primos de cada intervalo são enviados para um processo, o processo mestre, que vai calcular a quantidade total de números primos somando a quantidade de números primos encontrada por todos os processos.

2.2 Abordagem Bag of tasks

O algoritmo baseado no método Bag of tasks para contagem de números primos consiste na partilha do intervalo total de números a serem verificados em intervalos menores por um processo mestre. Então, cada processo subordinado recebe um intervalo para verificar a quantidade de números primos, enviando o valor calculado para o processo mestre. Em seguida, o processo mestre devolve outro intervalo para ser verificado pelo processo subordinado, caso ainda tenha algum intervalo restante. Além disso, o processo mestre soma o valor recebido na contagem parcial de números primos encontrados por todos os processos subordinados. Este processo termina quando o último intervalo é processado e o processo mestre retorna a quantidade total de números primos.

3 Resultados das implementações da abordagem Naive

3.1 p-Send

p-Send é uma implementação da abordagem Naive do algoritmo de contagem de números primos que utiliza envio padrão e recebimento ponto-a-ponto bloqueantes MPI_Send e MPI_Recv.

```
if(num_procs > 1 && meu_ranque != 0) {
    //MPI_Reduce(&cont, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    //int MPI_Send(void* mensagem, int cont, MPI_Datatype tipo_mpi, int destino, int etiq, MPI_Comm com)
    MPI_Send(&cont, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
} else {
    total = cont;
}

if(meu_ranque == 0 ){
    for (int ii = 0; ii < num_procs - 1; ii++){
        //int MPI_Recv(void* mensagem, int cont, MPI_Datatype tipo_mpi, int origem, int etiq, MPI_Comm com, MPI_Status* estado)
        MPI_Recv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
        total += cont;
    }
};
```

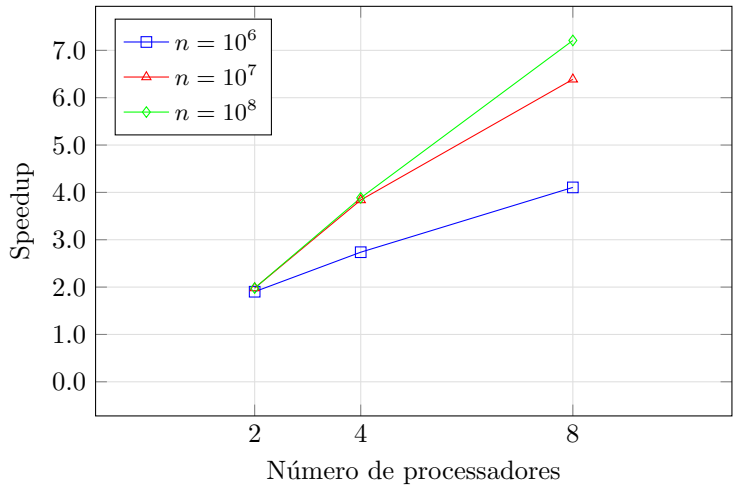

Figure 1: Speedup vs. Número de processadores - Send

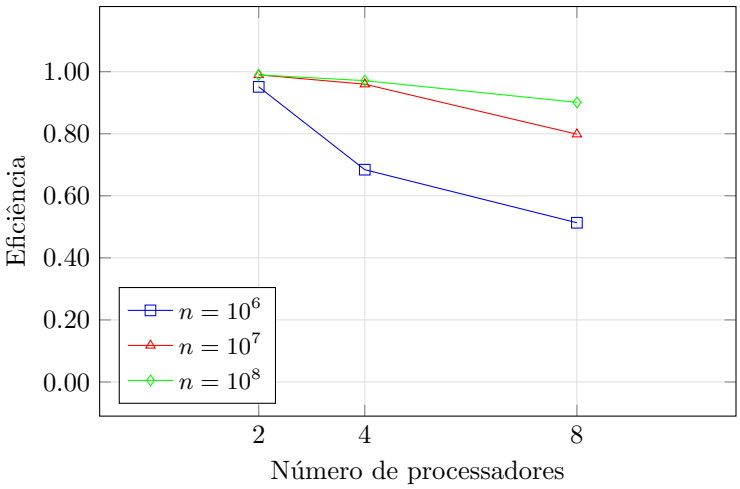

Figure 2: Eficiência vs. Número de processadores - p_Send

3.2 p-Send-Irecv

p-Send-Irecv é uma implementação da abordagem Naive do algoritmo de contagem de números primos que utiliza envio padrão e recebimento ponto-a-ponto bloqueante MPI_Send e não bloqueante MPI_Irecv em conjunto com MPI_Wait.

```
if(num_procs > 1 && meu_ranque != 0) {
    //MPI_Reduce(&cont, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    //int MPI_Send(void* mensagem, int cont, MPI_Datatype tipo_mpi, int destino, int etiq, MPI_Comm com)
    MPI_Send(&cont, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
} else {
    total = cont;
}

if(meu_ranque == 0 ){
    for (int ii = 0; ii < num_procs - 1; ii++){
        //int MPI_Recv(void* mensagem, int cont, MPI_Datatype tipo_mpi, int origem, int etiq, MPI_Comm com, MPI_Status* estado)
        //int MPI_Irecv(void* mensagem, int cont, MPI_Datatype tipo_mpi, int origem, int etiq, MPI_Comm com,MPI_Request *pedido)
        MPI_Irecv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, MPI_STATUS_IGNORE);
        total += cont;
    }
};
```

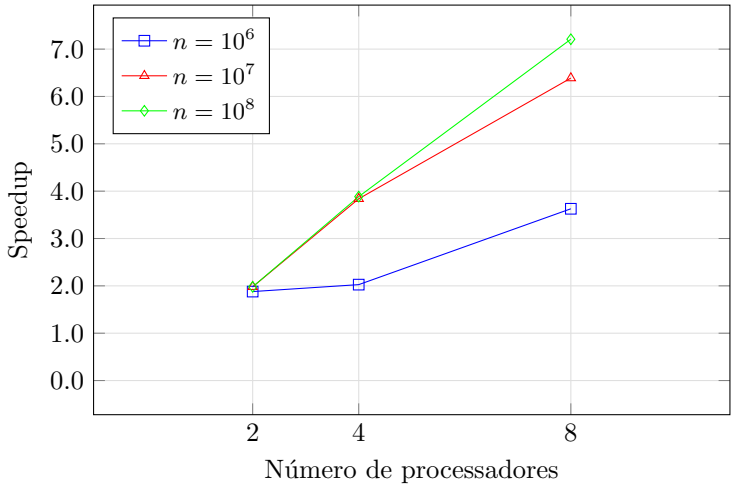

Figure 3: Speedup vs. Número de processadores - p_Send_Irecv

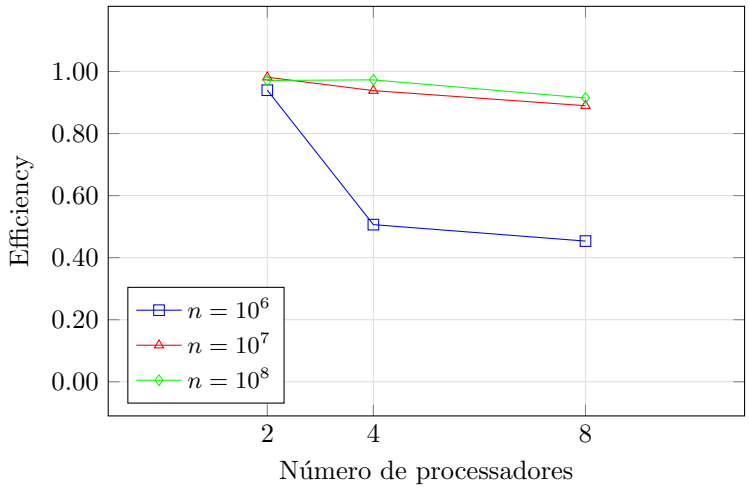

Figure 4: Efficiency vs. Número de processadores - p_Send_Irecv

3.3 p-Isend

p-Send-Irecv é uma implementação da abordagem Naive do algoritmo de contagem de números primos que utiliza envio padrão e recebimento ponto-a-ponto não bloqueante MPI_Isend e bloqueante MPI_Recv. Note que não é necessário nenhum tratamento para o MPI_Isend.

```
if(num_procs > 1 && meu_ranque != 0) {
    //MPI_Reduce(&cont, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    //int MPI_Isend(void* mensagem, int cont, MPI_Datatype tipo_mpi, int destino, int etiq, MPI_Comm com, MPI_Request *pedido)
    MPI_Isend(&cont, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,&request);
} else {
    total = cont;
}

if(meu_ranque == 0 ){
    for (int ii = 0; ii < num_procs - 1; ii++){
        //int MPI_Recv(void* mensagem, int cont, MPI_Datatype tipo_mpi, int origem, int etiq, MPI_Comm com, MPI_Status* estado)
        MPI_Recv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
        total += cont;
    }
};
```

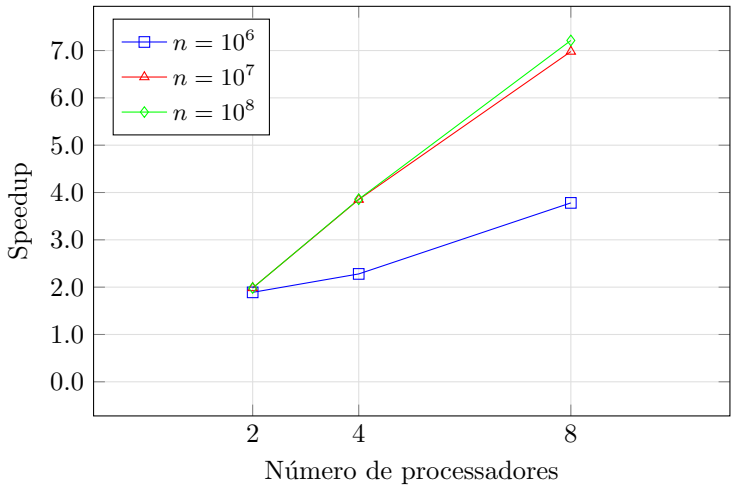

Figure 5: Speedup vs. Número de processadores - p-Isend

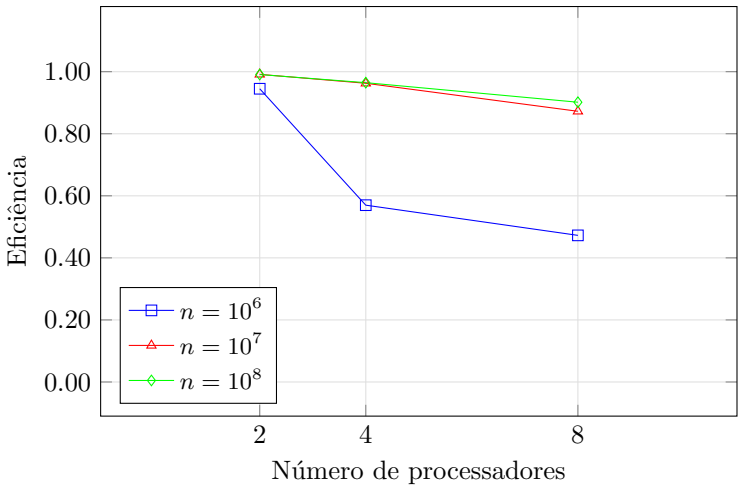

Figure 6: Eficiência vs. Número de processadores - p_Isend

3.4 p-Isend_Irecv

p-Isend-Irecv é uma implementação da abordagem Naive do algoritmo de contagem de números primos que utiliza envio padrão e recebimento ponto-a-ponto não bloqueante MPI_Isend e não bloqueante MPI_Irecv junto com o MPI_Wait.

```
if(num_procs > 1 && meu_ranque != 0) {
    //MPI_Reduce(&cont, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    //int MPI_Isend(void* mensagem, int cont, MPI_Datatype tipo_mpi, int destino, int etiq, MPI_Comm com, MPI_Request *pedido)
    MPI_Isend(&cont, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,&request);
} else {
    total = cont;
}

if(meu_ranque == 0 ){
    for (int ii = 0; ii < num_procs - 1; ii++){
        //int MPI_Recv(void* mensagem, int cont, MPI_Datatype tipo_mpi, int origem, int etiq, MPI_Comm com, MPI_Status* estado)
        //int MPI_Irecv(void* mensagem, int cont, MPI_Datatype tipo_mpi, int origem, int etiq, MPI_Comm com,MPI_Request *pedido)
        MPI_Irecv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, MPI_STATUS_IGNORE);
        total += cont;
    }
};
```

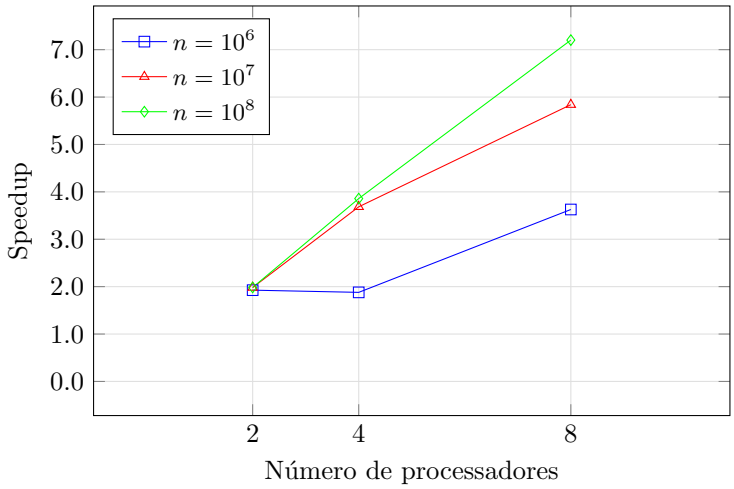

Figure 7: Speedup vs. Número de processadores - p-Isend_Irecv

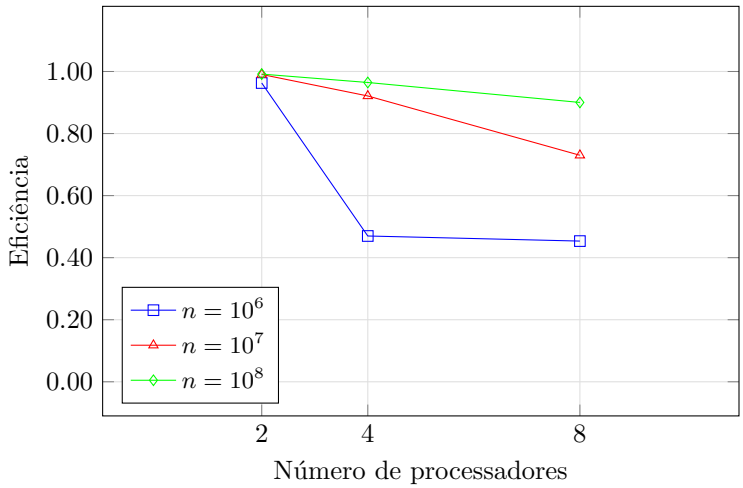

Figure 8: Eficiência vs. Número de processadores - p-Send_Irecv

3.5 p-Ssend

p-Ssend é uma implementação da abordagem Naive do algoritmo de contagem de números primos que utiliza envio síncrono e recebimento ponto-a-ponto bloqueante MPI_Ssend e MPI_Recv. Note que nenhum tratamento foi necessário pelo uso do MPI_Ssend.

```
if(meu_ranque == 0 ){
    for (int ii = 0; ii < num_procs; ii++){
        MPI_Recv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
        total += cont;
    }
};


if(num_procs > 1 && meu_ranque != 0 ) {
    //MPI_Reduce(&cont, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    MPI_Ssend(&cont, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
} else {
    total = cont;
}
```

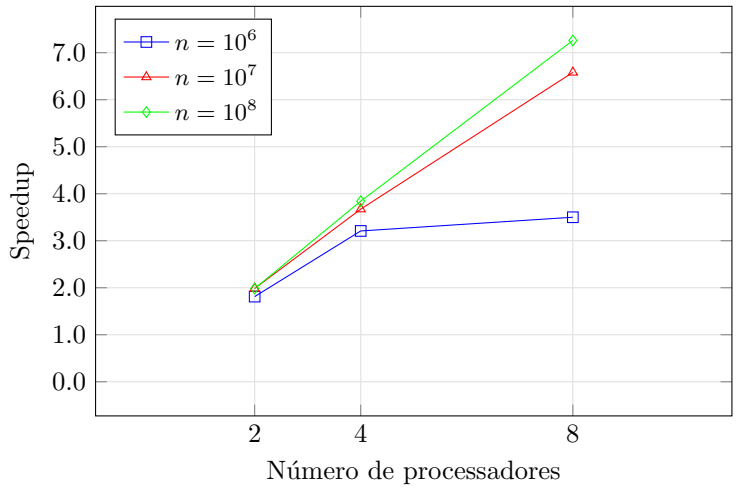

Figure 9: Speedup vs. Número de processadores - p-Ssend

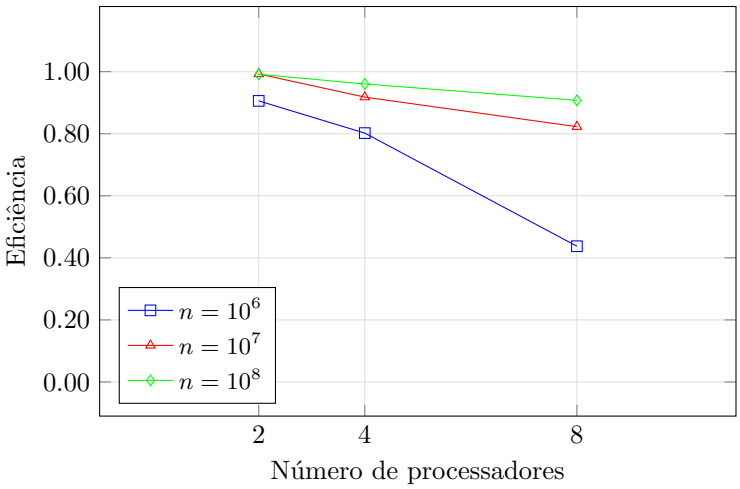

Figure 10: Eficiência vs. Número de processadores - p_Ssend

3.6 p-Ssend_Irecv

p-Ssend_Irecv é uma implementação da abordagem Naive do algoritmo de contagem de números primos que utiliza envio síncrono e recebimento ponto-a-ponto bloqueante MPI_Ssend e não bloqueante MPI_Irecv em conjunto com MPI_Waitall.

```
if(meu_ranque == 0 ){
    for (int ii = 0; ii < num_procs; ii++){
        MPI_Irecv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
    }

    // Wait for all receives to complete
    MPI_Waitall(num_procs - 1, request, status);

    // Update total with received counts
    for (int ii = 0; ii < num_procs - 1; ii++) {
        total += counts[ii];
    }

};


if(num_procs > 1 && meu_ranque != 0 ) {
    //MPI_Reduce(&cont, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    MPI_Ssend(&cont, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
} else {
    total = cont;
}
```

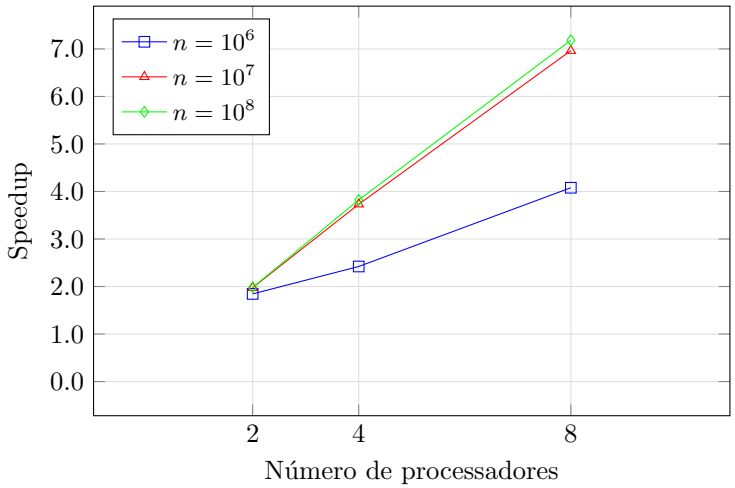

Figure 11: Speedup vs. Número de processadores - p-Ssend_Irecv

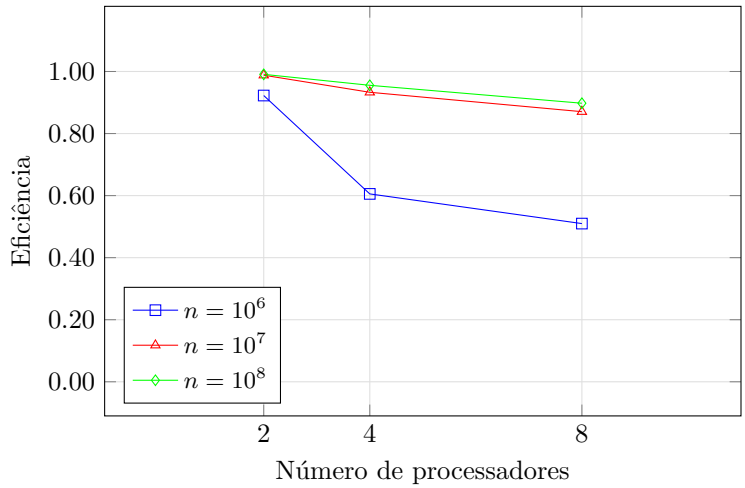

Figure 12: Eficiência vs. Número de processadores - p_Ssend_Irecv

3.7 p-Rsend

p-Rsend é uma implementação da abordagem Naive do algoritmo de contagem de números primos que utiliza envio síncrono e recebimento ponto-a-ponto bloqueante MPI_Rsend e MPI_Recv. O mais próximo que consegui chegar de uma implementação que respeita os requisitos para o uso do p-Rsend foi a seguinte:

```
if(meu_ranque == 0 ){
    total = cont;
    for (int ii = 0; ii < num_procs - 1; ii++){
        //int MPI_Send(void* mensagem, int cont, MPI_Datatype tipo_mpi, int destino, int etiq, MPI_Comm com)
        MPI_Send(&flag, 1, MPI_INT, ii + 1, 0, MPI_COMM_WORLD);
        //int MPI_Recv(void* mensagem, int cont, MPI_Datatype tipo_mpi, int origem, int etiq, MPI_Comm com, MPI_Status* estado)
        MPI_Recv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
        total += cont;
    }
};


if(num_procs > 1 && meu_ranque != 0) {
    //MPI_Reduce(&cont, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    //int MPI_Recv(void* mensagem, int cont, MPI_Datatype tipo_mpi, int origem, int etiq, MPI_Comm com, MPI_Status* estado)
    MPI_Recv(&flag, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
    //int MPI_Rsend(void* mensagem, int cont,MPI_Datatype tipo_mpi, int dest, int etiq, MPI_Comm com)
    MPI_Rsend(&cont, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
}
```

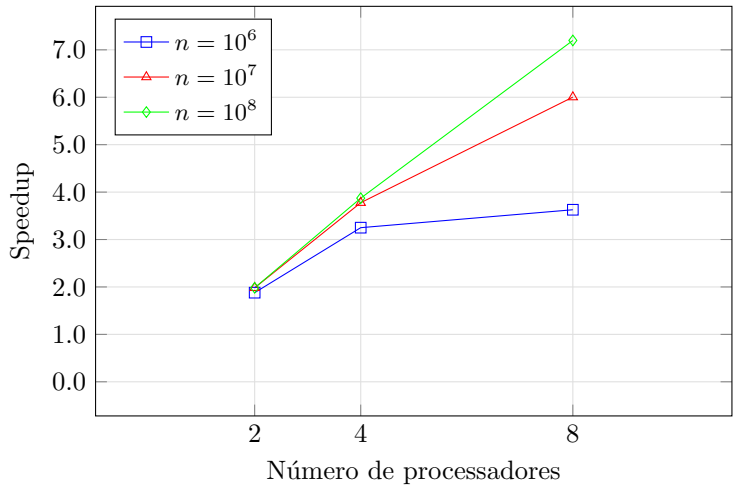

Figure 13: Speedup vs. Número de processadores - p-Rsend

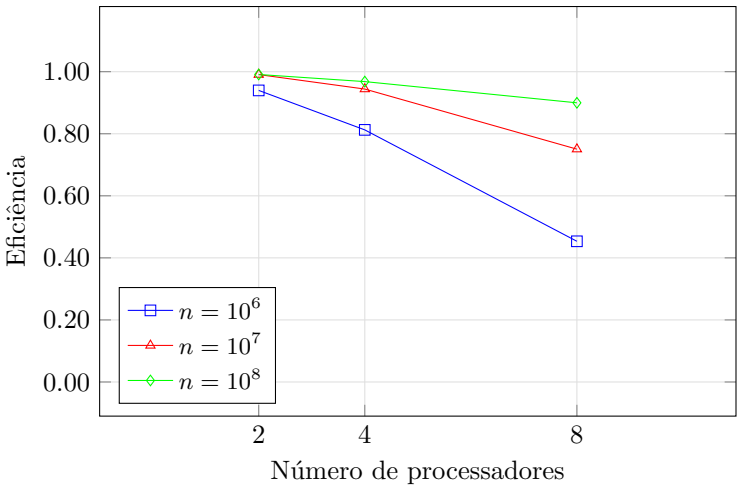

Figure 14: Eficiência vs. Número de processadores - p_Rsend

3.8 p-Rsend_Irecv

p-Rsend_Irecv é uma implementação da abordagem Naive do algoritmo de contagem de números primos que utiliza envio síncrono e recebimento ponto-a-ponto bloqueante MPI_Rsend e não bloqueante MPI_Recv em conjunto com o MPI_Waitall.

```c
if (meu_ranque == 0) {
    total = cont;

    for (int ii = 0; ii < num_procs - 1; ii++) {
        MPI_Irecv(&counts[ii], 1, MPI_INT, ii + 1, MPI_ANY_TAG, MPI_COMM_WORLD, &request[ii]);
    }

    MPI_Waitall(num_procs - 1, request, status);

    for (int ii = 0; ii < num_procs - 1; ii++) {
        total += counts[ii];
    }

    printf("\n total %d \n", total);
} else {
    // Send count to process 0
    MPI_Rsend(&cont, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
}
```

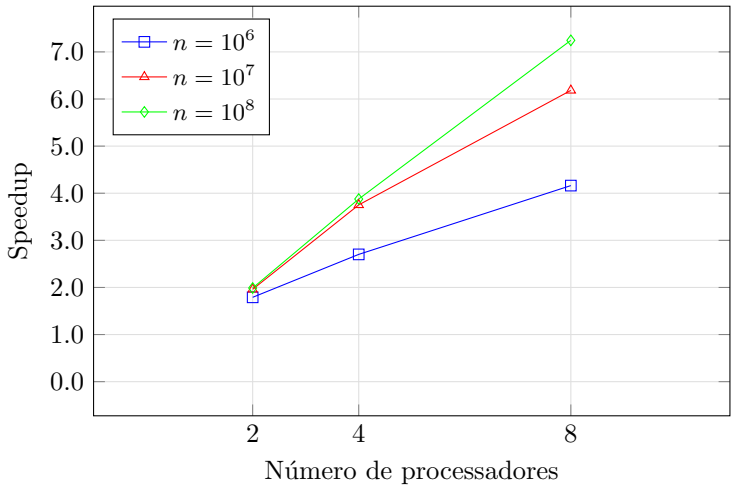

Figure 15: Speedup vs. Número de processadores - p-Rsend_Irecv

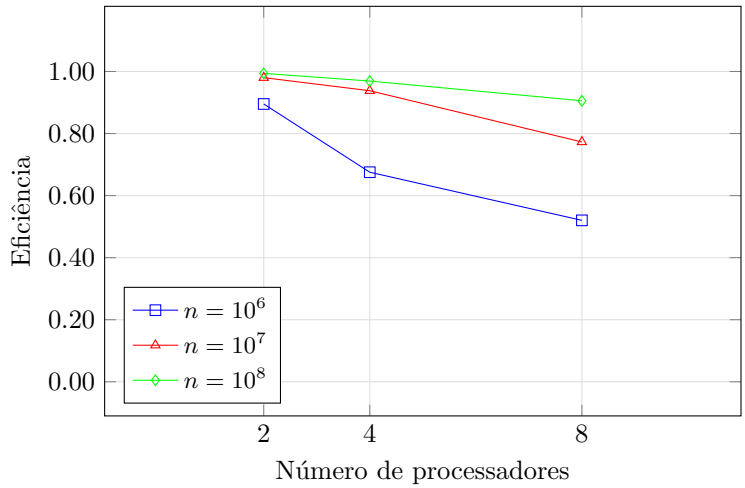

Figure 16: Eficiência vs. Número de processadores - p_Rsend_Irecv

4 Resultados para abordagem Bag of Tasks

4.1 b-Send

b-Send é uma implementação da abordagem Bag of tasks do algoritmo de contagem de números primos que utiliza envio padrão e recebimento ponto-a-ponto bloqueantes MPI_Send e MPI_Recv.

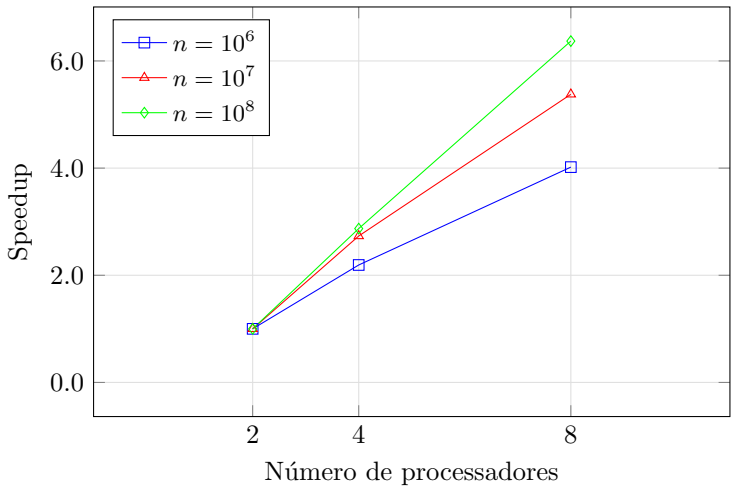

Figure 17: Speedup vs. Número de processadores - b_Send

```
    if (meu_ranque == 0) {
        for (dest=1, inicio=3; dest < num_procs && inicio < n; dest++, inicio += TAMANHO) {
            MPI_Send(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
        }
/* Fica recebendo as contagens parciais de cada processo */
        while (stop < (num_procs-1)) {
            MPI_Recv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
            total += cont;
            dest = estado.MPI_SOURCE;
            if (inicio > n) {
                tag = 99;
                stop++;
            }
/* Envia um nvo pedaço com TAMANHO números para o mesmo processo*/
            MPI_Send(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
            inicio += TAMANHO;
        }
    }
    else {
/* Cada processo escravo recebe o início do espaço de busca */
        while (estado.MPI_TAG != 99) {
            MPI_Recv(&inicio, 1, MPI_INT, raiz, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
            if (estado.MPI_TAG != 99) {
                for (i = inicio, cont=0; i < (inicio + TAMANHO) && i < n; i+=2)
                    if (primo(i) == 1)
                        cont++;
/* Envia a contagem parcial para o processo mestre */
                MPI_Send(&cont, 1, MPI_INT, raiz, tag, MPI_COMM_WORLD);
            }
        }
```

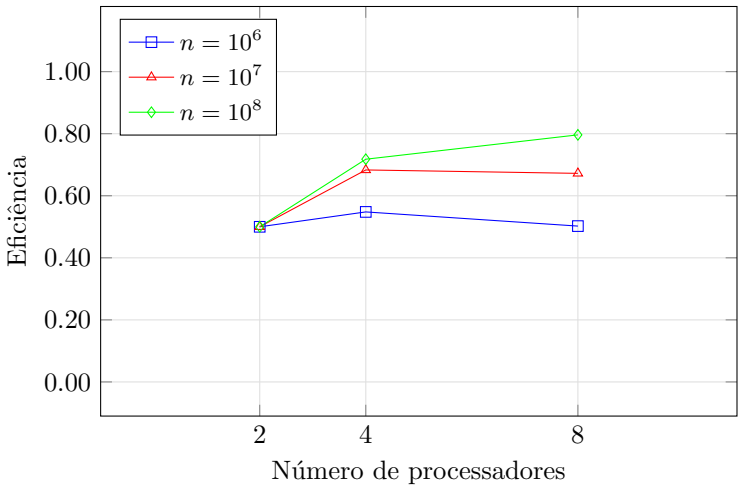

Figure 18: Eficiência vs. Número de processadores - b_Send

4.2 b-Send_Irecv

p-Send-Irecv é uma implementação da abordagem Bag of tasks do algoritmo de contagem de números primos que utiliza envio padrão e recebimento ponto-a-ponto bloqueante MPI_Send e não bloqueante MPI_Irecv em conjunto com MPI_Wait.

```
    if (meu_ranque == 0) {
        for (dest=1, inicio=3; dest < num_procs && inicio < n; dest++, inicio += TAMANHO) {
            MPI_Send(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
        }
/* Fica recebendo as contagens parciais de cada processo */
        while (stop < num_active_procs) {
            MPI_Irecv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
            MPI_Wait(&request, &estado);
            total += cont;
            dest = estado.MPI_SOURCE;
            if (inicio > n) {
                tag = 99;
                stop++;
            }
/* Envia um nvo pedaço com TAMANHO números para o mesmo processo*/
            MPI_Send(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
            inicio += TAMANHO;
        }
    }
    else {
/* Cada processo escravo recebe o início do espaço de busca */
        while (estado.MPI_TAG != 99) {
            MPI_Irecv(&inicio, 1, MPI_INT, raiz, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
            MPI_Wait(&request, &estado);
            if (estado.MPI_TAG != 99) {
                for (i = inicio, cont=0; i < (inicio + TAMANHO) && i < n; i+=2)
                    if (primo(i) == 1)
                        cont++;
/* Envia a contagem parcial para o processo mestre */
                MPI_Send(&cont, 1, MPI_INT, raiz, tag, MPI_COMM_WORLD);
```

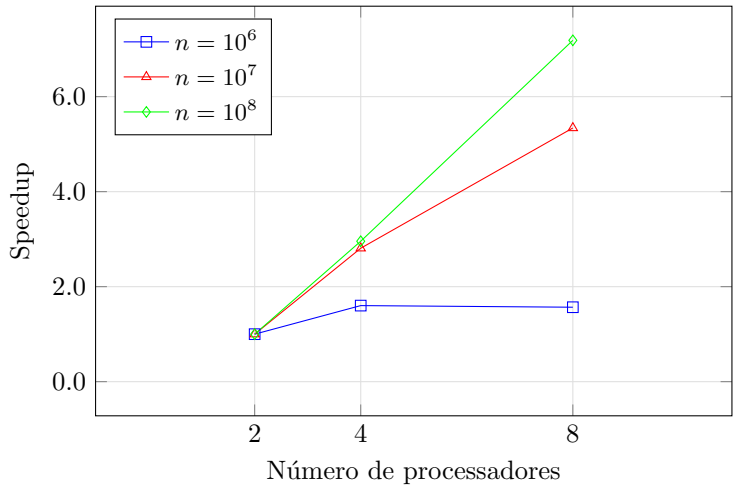

Figure 19: Speedup vs. Número de processadores - b_Send_Irecv

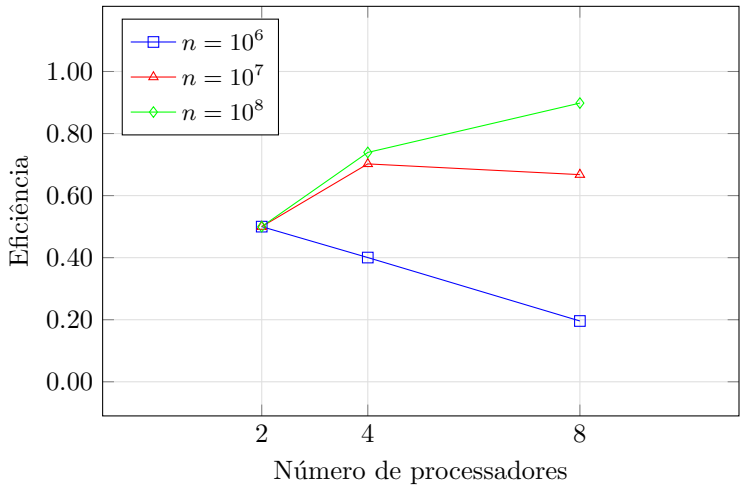

Figure 20: Eficiência vs. Número de processadores - b_Send_Irecv

4.3 b-Isend

b-Isend é uma implementação da abordagem Bag of tasks do algoritmo de contagem de números primos que utiliza envio padrão e recebimento ponto-a-ponto não bloqueante MPI_Isend e bloqueante MPI_Recv. Note que não é necessário nenhum tratamento para o MPI_Isend.

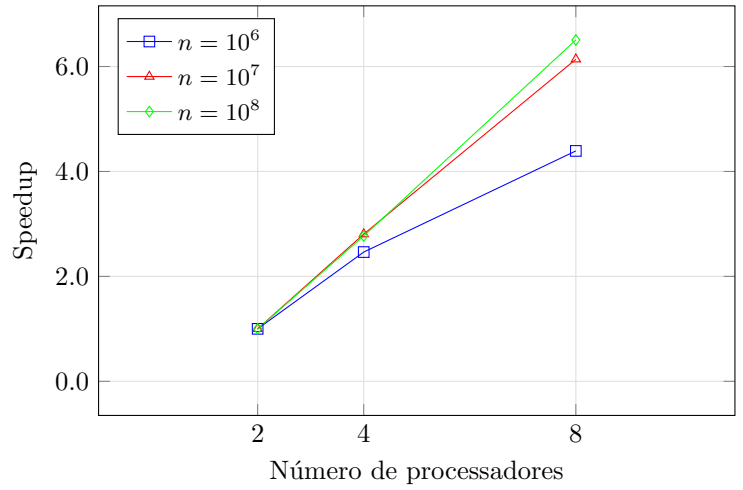

Figure 21: Speedup vs. Número de processadores - b_Isend

```
/* Envia pedaços com TAMANHO números para cada processo */
    if (meu_ranque == 0) {
        for (dest=1, inicio=3; dest < num_procs && inicio < n; dest++, inicio += TAMANHO) {
            MPI_Isend(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD, &request);
        }
/* Fica recebendo as contagens parciais de cada processo */
        while (stop < (num_procs-1)) {
            MPI_Recv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
            total += cont;
            dest = estado.MPI_SOURCE;
            if (inicio > n) {
                tag = 99;
                stop++;
            }
/* Envia um nvo pedaço com TAMANHO números para o mesmo processo*/
            MPI_Isend(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD, &request);
            inicio += TAMANHO;
        }
    }
    else {
/* Cada processo escravo recebe o início do espaço de busca */
        while (estado.MPI_TAG != 99) {
            MPI_Recv(&inicio, 1, MPI_INT, raiz, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
            if (estado.MPI_TAG != 99) {
                for (i = inicio, cont=0; i < (inicio + TAMANHO) && i < n; i+=2)
                    if (primo(i) == 1)
                        cont++;
/* Envia a contagem parcial para o processo mestre */
                MPI_Isend(&cont, 1, MPI_INT, raiz, tag, MPI_COMM_WORLD, &request);
            }
        }
```

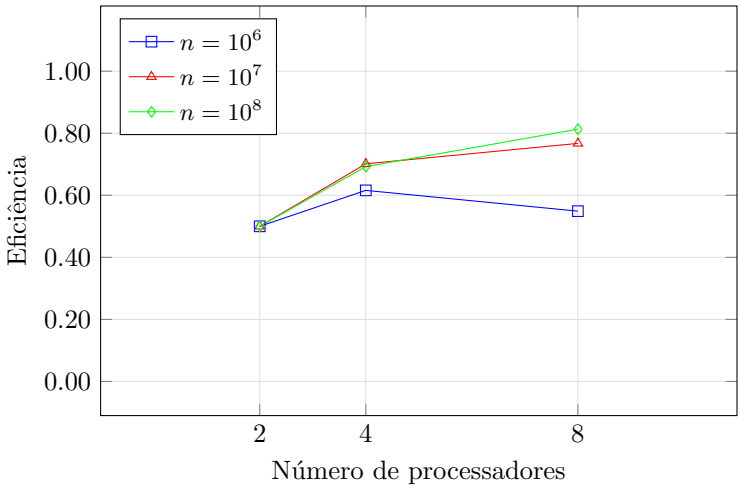

Figure 22: Eficiência vs. Número de processadores - b_Isend

4.4 b-Isend_Irecv

b-Isend-Irecv é uma implementação da abordagem Bag of tasks do algoritmo de contagem de números primos que utiliza envio padrão e recebimento ponto-a-ponto não bloqueante MPI_Isend e não bloqueante MPI_Irecv junto com o MPI_Wait.

```
    if (meu_ranque == 0) {
        for (dest=1, inicio=3; dest < num_procs && inicio < n; dest++, inicio += TAMANHO) {
            MPI_Isend(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD,&request);
        }
/* Fica recebendo as contagens parciais de cada processo */
        int num_active_procs = dest - 1;
        while (stop < num_active_procs) {
            MPI_Irecv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
            MPI_Wait(&request, &estado);
            total += cont;
            dest = estado.MPI_SOURCE;
            if (inicio > n) {
                tag = 99;
                stop++;
            }
/* Envia um nvo pedaço com TAMANHO números para o mesmo processo*/
            MPI_Isend(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD, &request);
            inicio += TAMANHO;
        }
    }
    else {
/* Cada processo escravo recebe o início do espaço de busca */
        while (estado.MPI_TAG != 99) {
            MPI_Irecv(&inicio, 1, MPI_INT, raiz, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
            MPI_Wait(&request, &estado);
            if (estado.MPI_TAG != 99) {
                for (i = inicio, cont=0; i < (inicio + TAMANHO) && i < n; i+=2)
                    if (primo(i) == 1)
                        cont++;
/* Envia a contagem parcial para o processo mestre */
                MPI_Isend(&cont, 1, MPI_INT, raiz, tag, MPI_COMM_WORLD,&request);
```

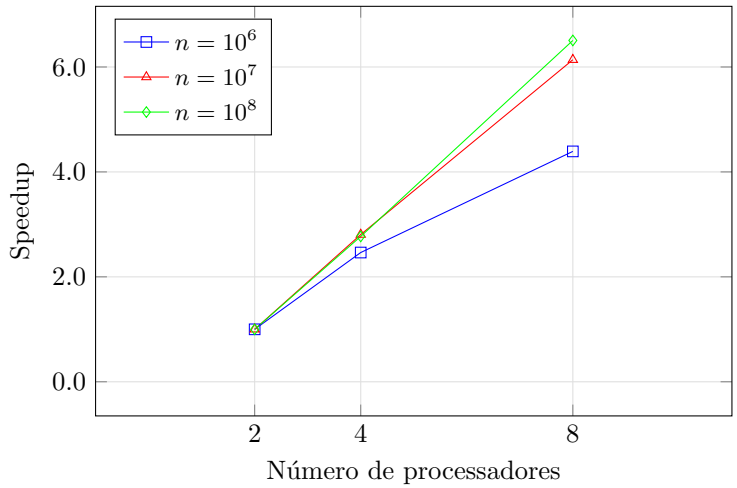

Figure 23: Speedup vs. Número de processadores - b_Isend_Irecv.c

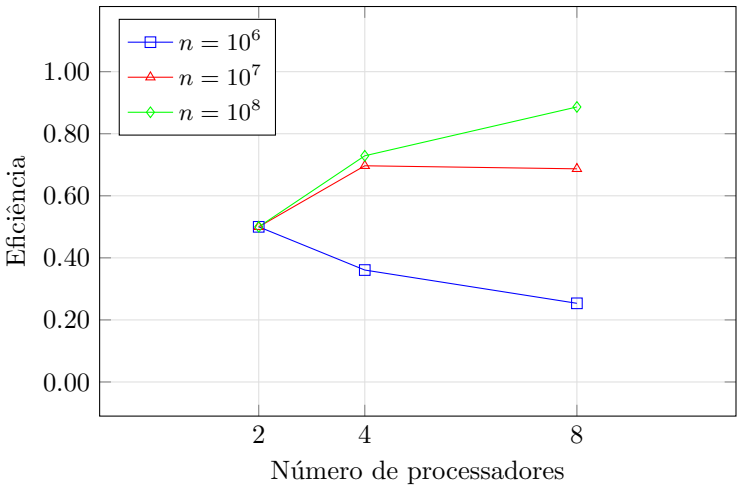

Figure 24: Eficiência vs. Número de processadores - b_Isend_Irecv.c

4.5 b-Ssend

B-Ssend é uma implementação da abordagem Bag pf tasks do algoritmo de contagem de números primos que utiliza envio síncrono e recebimento ponto-a-ponto bloqueante MPI_Ssend e MPI_Recv. Note que nenhum tratamento foi necessário pelo uso do MPI_Ssend.

```c
/* Envia pedaços com TAMANHO números para cada processo */
    if (meu_ranque == 0) {
        for (dest=1, inicio=3; dest < num_procs && inicio < n; dest++, inicio += TAMANHO) {
            MPI_Ssend(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
        }
/* Fica recebendo as contagens parciais de cada processo */
        while (stop < (num_procs-1)) {
            MPI_Recv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
            total += cont;
            dest = estado.MPI_SOURCE;
            if (inicio > n) {
                tag = 99;
                stop++;
            }
/* Envia um nvo pedaço com TAMANHO números para o mesmo processo*/
            MPI_Ssend(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
            inicio += TAMANHO;
        }
    }
    else {
/* Cada processo escravo recebe o início do espaço de busca */
        while (estado.MPI_TAG != 99) {
            MPI_Recv(&inicio, 1, MPI_INT, raiz, MPI_ANY_TAG, MPI_COMM_WORLD, &estado);
            if (estado.MPI_TAG != 99) {
                for (i = inicio, cont=0; i < (inicio + TAMANHO) && i < n; i+=2)
                    if (primo(i) == 1)
                        cont++;
/* Envia a contagem parcial para o processo mestre */
                MPI_Ssend(&cont, 1, MPI_INT, raiz, tag, MPI_COMM_WORLD);
            }
```

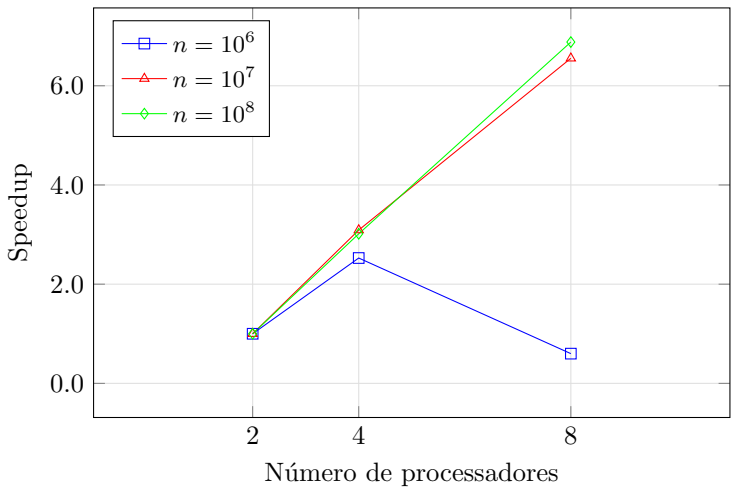

Figure 25: Speedup vs. Número de processadores - b_Ssend

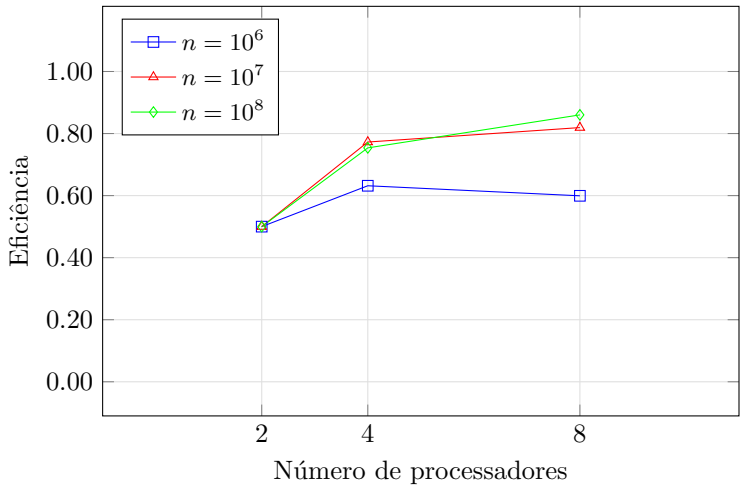

Figure 26: Eficiência vs. Número de processadores - b_Ssend

4.6 b-Ssend_Irecv

b-Ssend_Irecv é uma implementação da abordagem Bag of tasks do algoritmo de contagem de números primos que utiliza envio síncrono e recebimento ponto-a-ponto bloqueante MPI_Ssend e não bloqueante MPI_Irecv em conjunto com MPI_Wait.

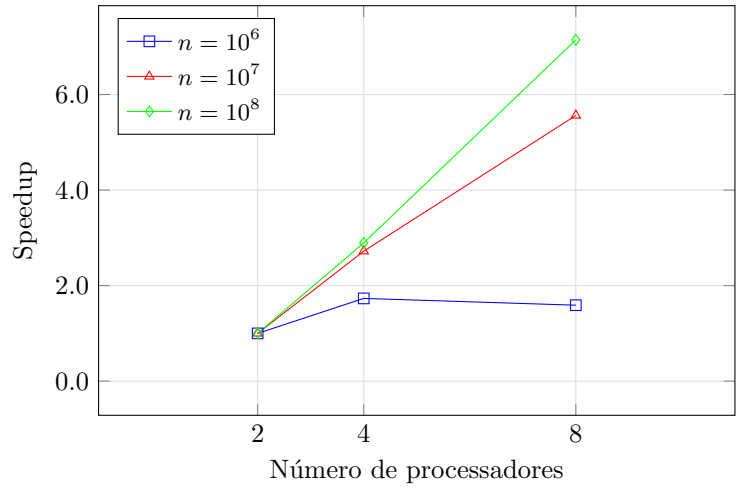

Figure 27: Speedup vs. Número de processadores - b_Ssend_Irecv

```
    if (meu_ranque == 0) {
        for (dest=1, inicio=3; dest < num_procs && inicio < n; dest++, inicio += TAMANHO) {
            MPI_Ssend(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
        }
/* Fica recebendo as contagens parciais de cada processo */
        int num_active_procs = dest - 1;
        while (stop < num_active_procs) {
            MPI_Irecv(&cont, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
            MPI_Wait(&request, &estado);
            total += cont;
            dest = estado.MPI_SOURCE;
            if (inicio > n) {
                tag = 99;
                stop++;
            }
/* Envia um nvo pedaço com TAMANHO números para o mesmo processo*/
            MPI_Ssend(&inicio, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
            inicio += TAMANHO;
        }
    }
    else {
/* Cada processo escravo recebe o início do espaço de busca */
        while (estado.MPI_TAG != 99) {
            MPI_Irecv(&inicio, 1, MPI_INT, raiz, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
            MPI_Wait(&request, &estado);
            if (estado.MPI_TAG != 99) {
                for (i = inicio, cont=0; i < (inicio + TAMANHO) && i < n; i+=2)
                    if (primo(i) == 1)
                        cont++;
/* Envia a contagem parcial para o processo mestre */
                MPI_Ssend(&cont, 1, MPI_INT, raiz, tag, MPI_COMM_WORLD);
```

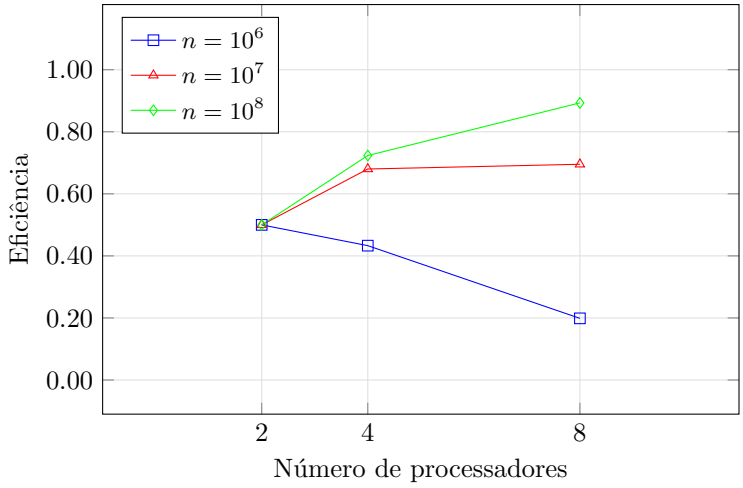

Figure 28: Eficiência vs. Número de processadores - b_Ssend_Irecv

5 Tempos de execução

p-Send				
n	$np = 1$	$np = 2$	$np = 4$	$np = 8$
10^6	0.156	0.082	0.057	0.038
10^7	4.05	2.046	1.055	0.634
10^8	111.844	56.505	28.791	15.515
p-Send_Irecv				
n	$np = 1$	$np = 2$	$np = 4$	$np = 8$
10^6	0.156	0.083	0.077	0.043
10^7	4.050	2.063	1.079	0.569
10^8	112.060	57.714	28.873	15.302
p-Isend				
n	$np = 1$	$np = 2$	$np = 4$	$np = 8$
10^6	0.155	0.082	0.068	0.041
10^7	4.055	2.045	1.053	0.581
10^8	111.863	56.443	28.990	15.508
p-Isend_Irecv				
n	$np = 1$	$np = 2$	$np = 4$	$np = 8$
10^6	0.156	0.081	0.083	0.043
10^7	4.053	2.046	1.100	0.694
10^8	111.913	56.427	29.005	15.539
p-Ssend				
n	$np = 1$	$np = 2$	$np = 4$	$np = 8$
10^6	0.154	0.085	0.048	0.044
10^7	4.054	2.042	1.104	0.616
10^8	111.851	56.393	29.116	15.405
p-Ssend_Irecv				
n	$np = 1$	$np = 2$	$np = 4$	$np = 8$
10^6	0.155	0.084	0.064	0.038
10^7	4.053	2.051	1.086	0.582
10^8	111.865	56.451	29.270	15.576
p-Rsend				
n	$np = 1$	$np = 2$	$np = 4$	$np = 8$
10^6	0.156	0.083	0.048	0.043
10^7	4.052	2.045	1.073	0.675
10^8	111.839	56.423	28.881	15.538
p-Rsend_Irecv				
n	$np = 1$	$np = 2$	$np = 4$	$np = 8$
10^6	0.154	0.086	0.057	0.037
10^7	4.056	2.069	1.081	0.656
10^8	111.924	56.299	28.870	15.448

Table 1: Tempos de execução para implementações do Naive

b_Isend			
n	$np = 2$	$np = 4$	$np = 8$
10^6	0.202	0.082	0.046
10^7	4.972	1.772	0.810
10^8	133.339	48.130	20.496
b_Isend_Irecv			
n	$np = 2$	$np = 4$	$np = 8$
10^6	0.215	0.149	0.106
10^7	5.457	1.958	0.993
10^8	142.372	48.825	20.079
b_Ssend			
n	$np = 2$	$np = 4$	$np = 8$
10^6	0.187	0.074	0.039
10^7	4.568	1.478	0.697
10^8	124.326	41.232	18.067
b_Ssend_Irecv			
n	$np = 2$	$np = 4$	$np = 8$
10^6	0.194	0.112	0.122
10^7	5.212	1.916	0.937
10^8	136.360	47.122	19.084
b_Send			
n	$np = 2$	$np = 4$	$np = 8$
10^6	0.217	0.099	0.054
10^7	5.340	1.954	0.993
10^8	136.754	47.629	21.467
b_Send_Irecv			
n	$np = 2$	$np = 4$	$np = 8$
10^6	142.626	48.240	19.849
10^7	5.362	1.909	1.004
10^8	0.221	0.138	0.141

Table 2: Tempos de execução para imlementação Bag of tasks