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

Maio 13, 2024

1 Abordagem Naive e Bag of tasks.

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

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

2 Resultados das implementações da abordagem Naive

2.1 p-Send

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;
    }
};
```

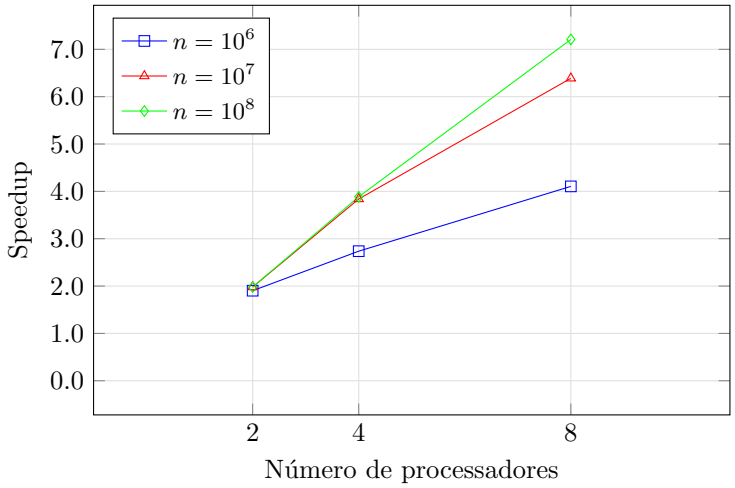

Figure 1: Speedup vs. Número de processadores - Send

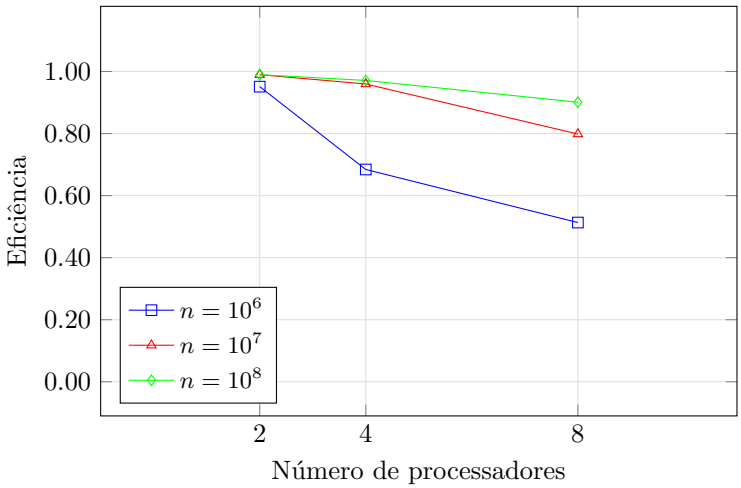

Figure 2: Eficiência vs. Número de processadores - p_Send

2.2 p-Send-Irecv

MPI_Send e MPI_Irecv, 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;
    }
};
```

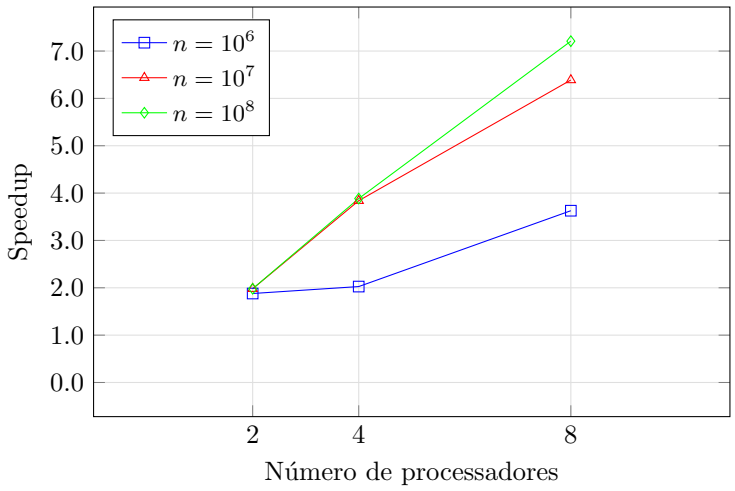

Figure 3: Speedup vs. Número de processadores - p_Send_Irecv

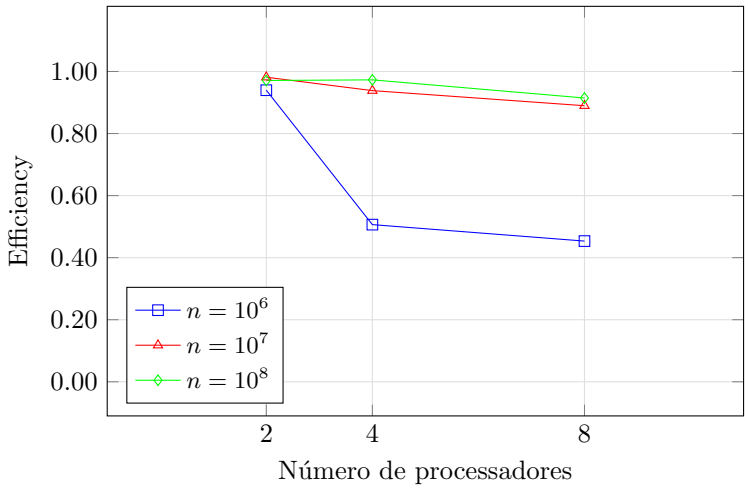

Figure 4: Efficiency vs. Número de processadores - p_Send_Irecv

2.3 p-Isend

MPI_Isend 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_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;
    }
};
```

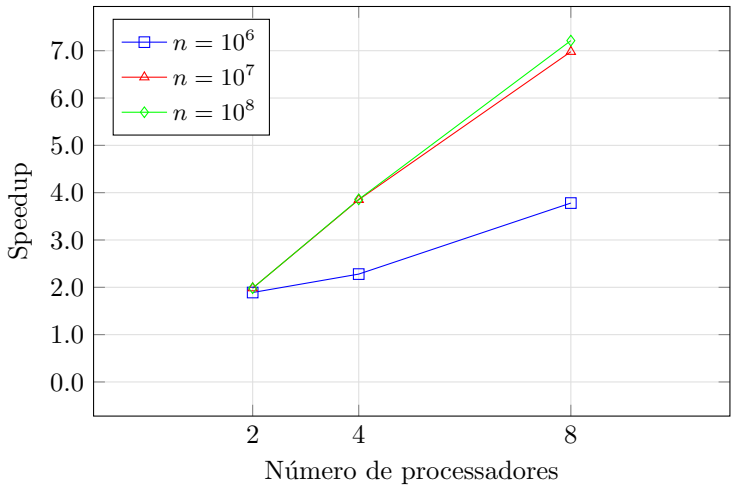

Figure 5: Speedup vs. Número de processadores - p-Isend

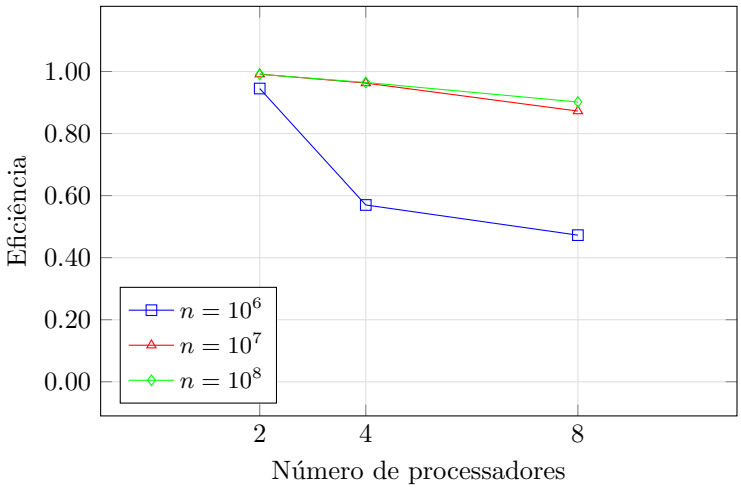

Figure 6: Eficiência vs. Número de processadores - p_Isend

2.4 p-Isend_Irecv

MPI_Isend e MPI_Irecv, 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;
    }
};
```

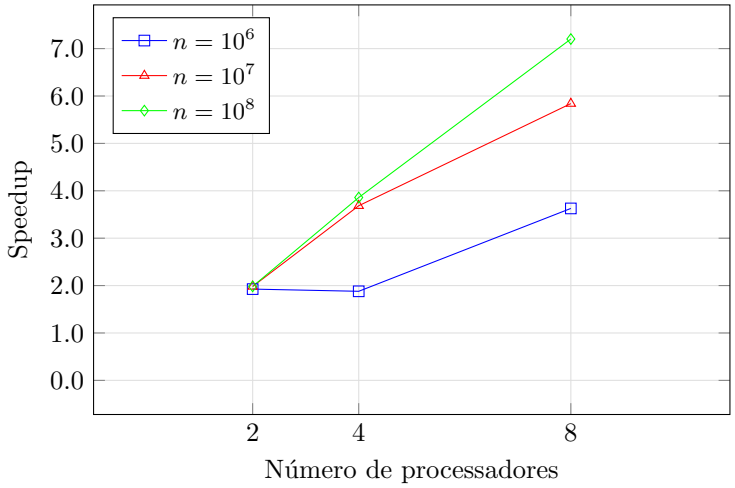

Figure 7: Speedup vs. Número de processadores - p-Isend_Irecv

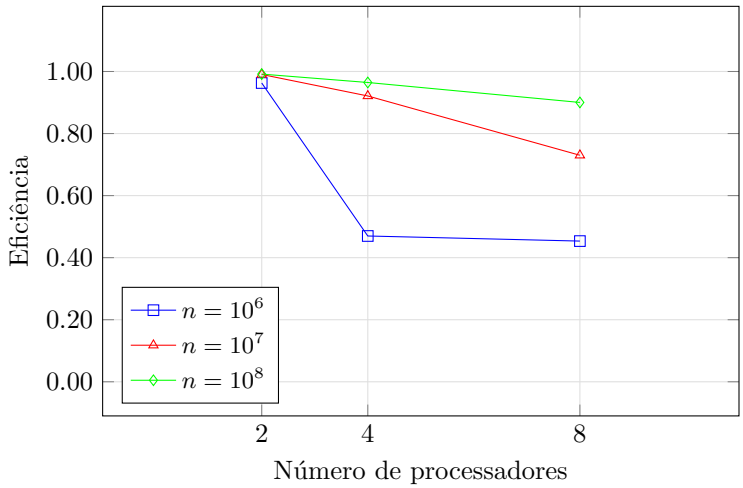

Figure 8: Eficiência vs. Número de processadores - p-Send_Irecv

2.5 p-Ssend

MPI_Ssend e MPI_Recv.

```
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;
}
```

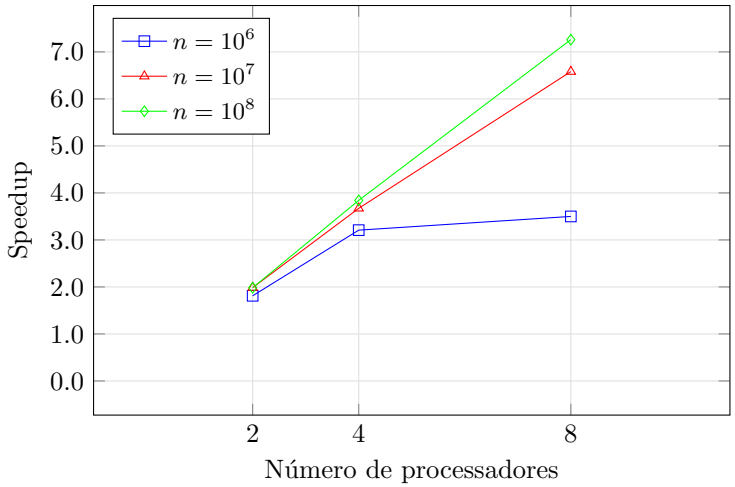

Figure 9: Speedup vs. Número de processadores - p-Ssend

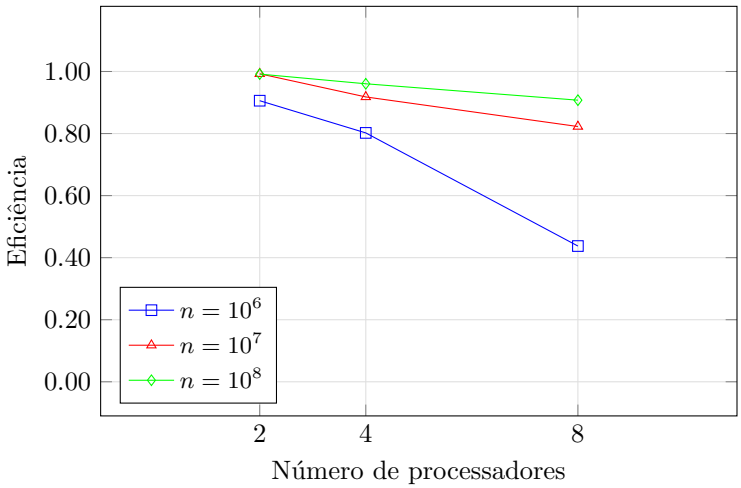

Figure 10: Eficiência vs. Número de processadores - p_Ssend

2.6 p-Ssend_Irecv

MPI_Ssend e MPI_Irecv, 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;
}
```

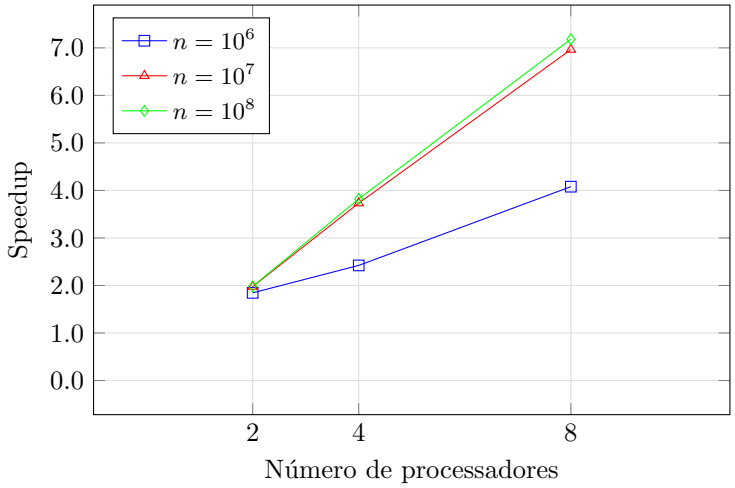

Figure 11: Speedup vs. Número de processadores - p-Ssend_Irecv

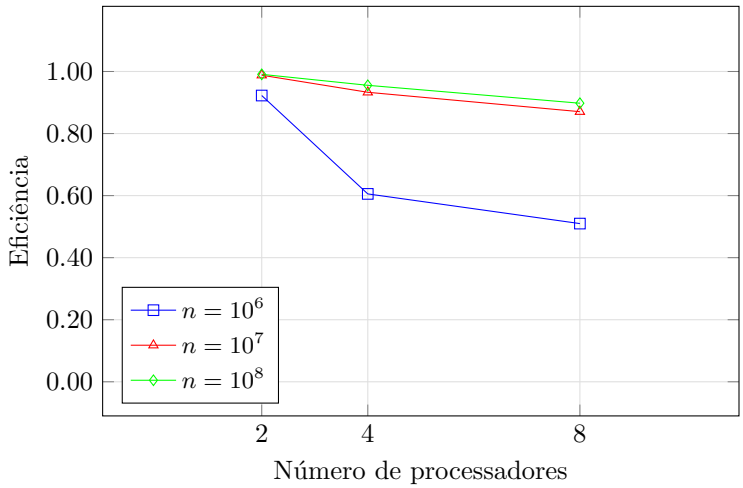

Figure 12: Eficiência vs. Número de processadores - p_Ssend_Irecv

2.7 p-Rsend

MPI_Rsend e MPI_Recv, MPI_Send, MPI$_{Recv}$.

```
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);
}
```

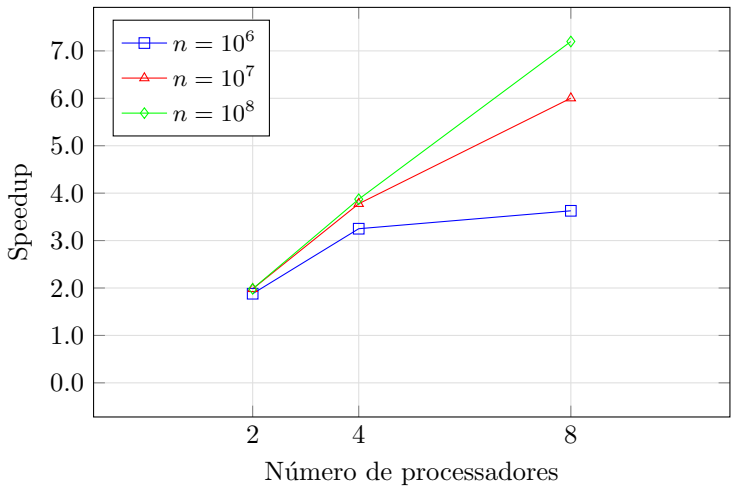

Figure 13: Speedup vs. Número de processadores - p-Rsend

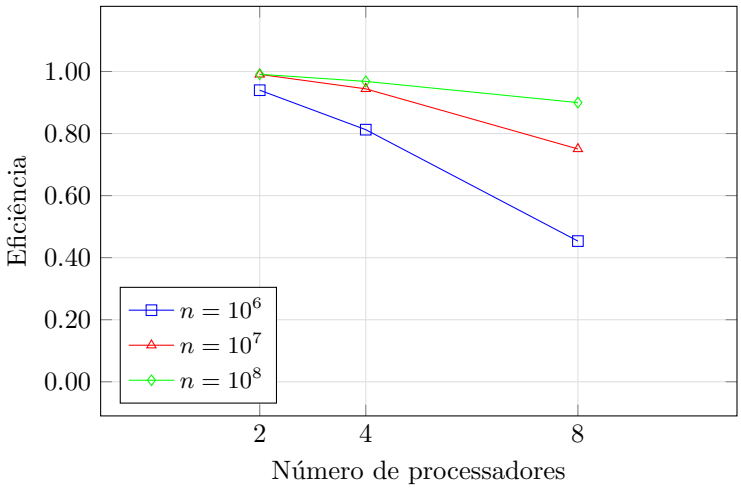

Figure 14: Eficiência vs. Número de processadores - p_Rsend

2.8 p-Rsend_Irecv

MPI_Rsend e MPI_Recv ,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);
}
```

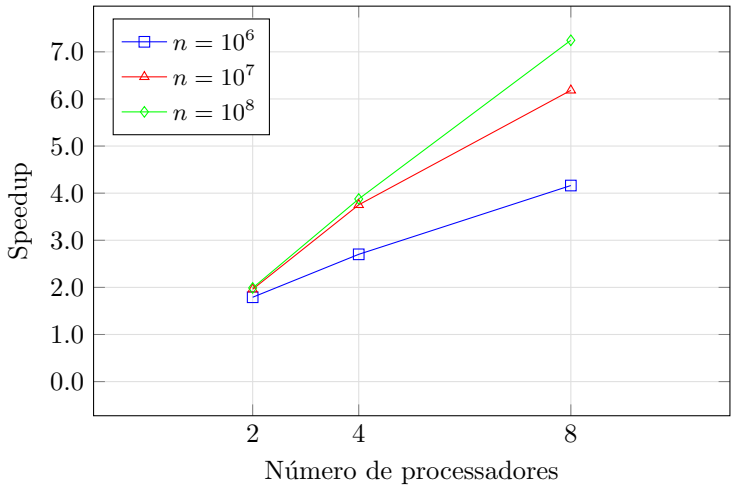

Figure 15: Speedup vs. Número de processadores - p-Rsend_Irecv

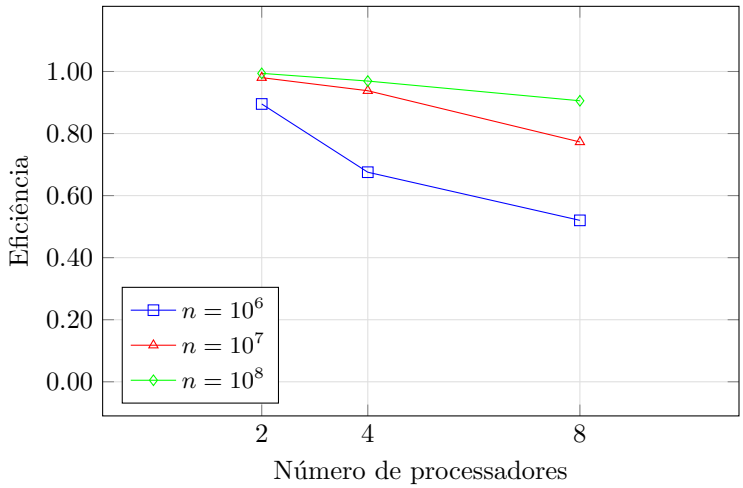

Figure 16: Eficiência vs. Número de processadores - p_Rsend_Irecv

3 Resultados para abordagem Bag of Tasks

3.1 b-Send

MPI_Send e MPI_Recv.

```
    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);
            }
        }
```

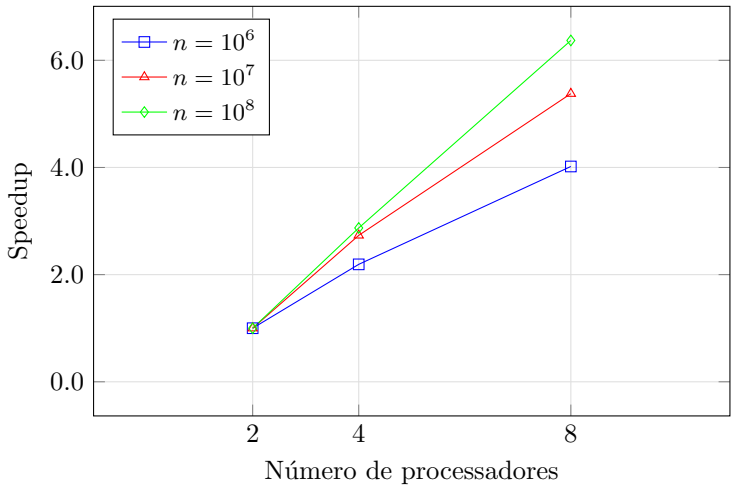

Figure 17: Speedup vs. Número de processadores - b_Send

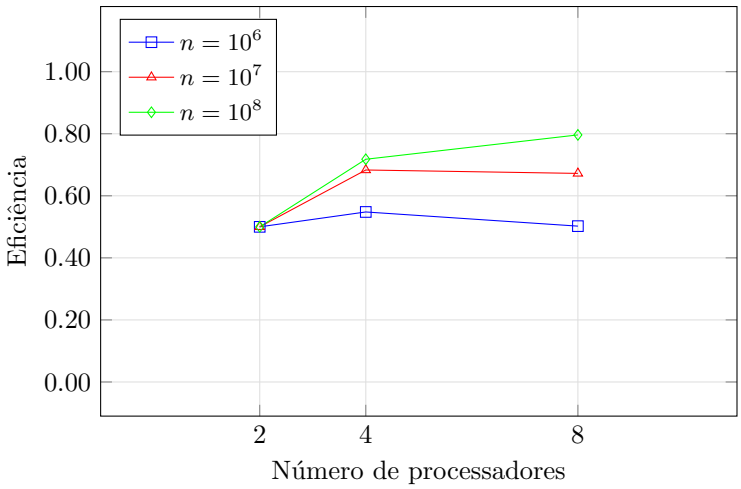

Figure 18: Eficiência vs. Número de processadores - b_Send

3.2 b-Send_Irecv

MPI_Send e MPI_Irecv, 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);
            }
```

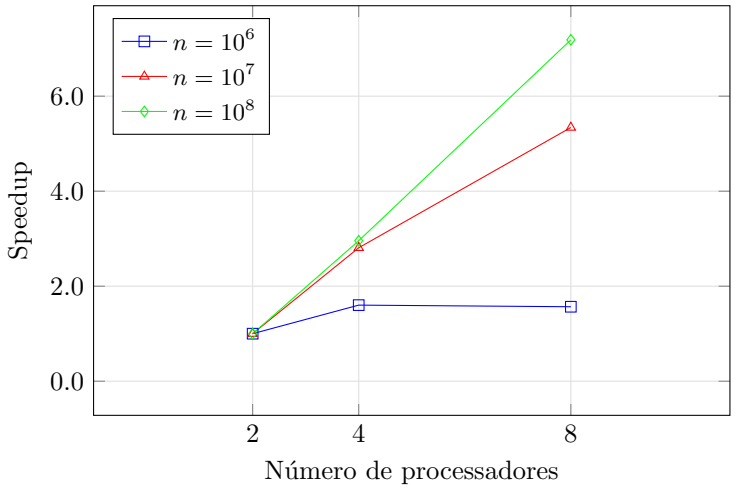

Figure 19: Speedup vs. Número de processadores - b_Send_Irecv

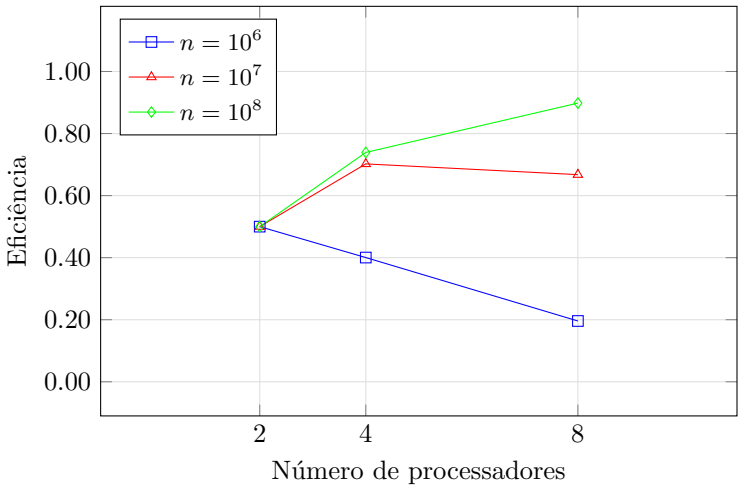

Figure 20: Eficiência vs. Número de processadores - b_Send_Irecv

3.3 b-Isend

MPI_Isend e MPI_Recv.

```
/* 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);
            }
        }
```

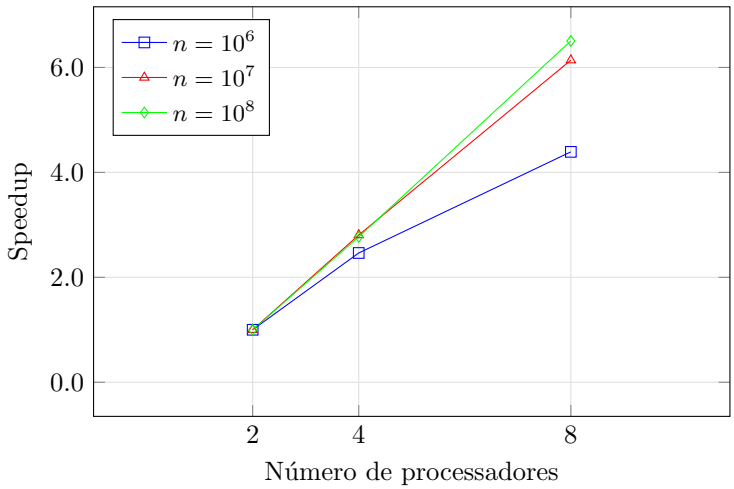

Figure 21: Speedup vs. Número de processadores - b_Isend

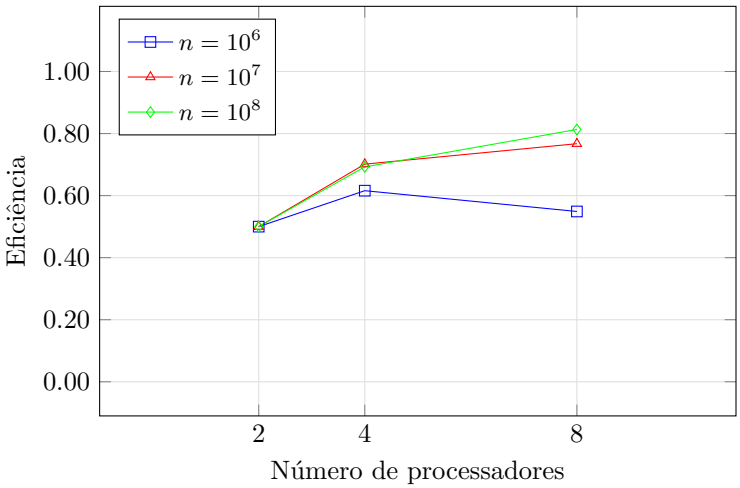

Figure 22: Eficiência vs. Número de processadores - b_Isend

3.4 b-Isend_Irecv

MPI_Isend e MPI_Irecv, 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);
```

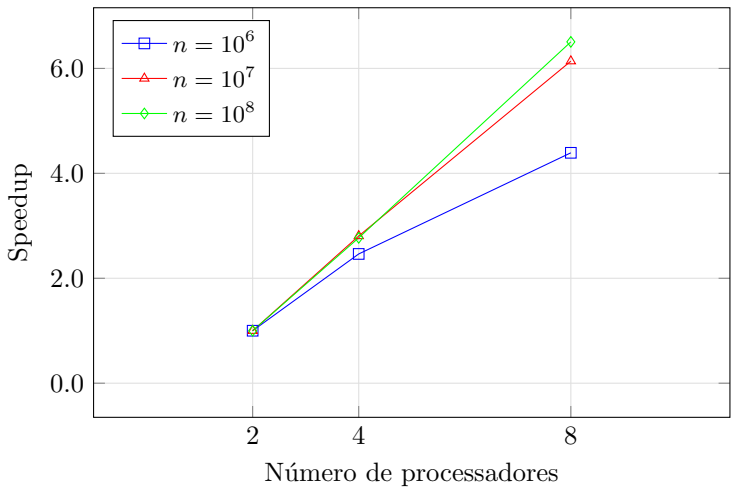

Figure 23: Speedup vs. Número de processadores - b_Isend_Irecv.c

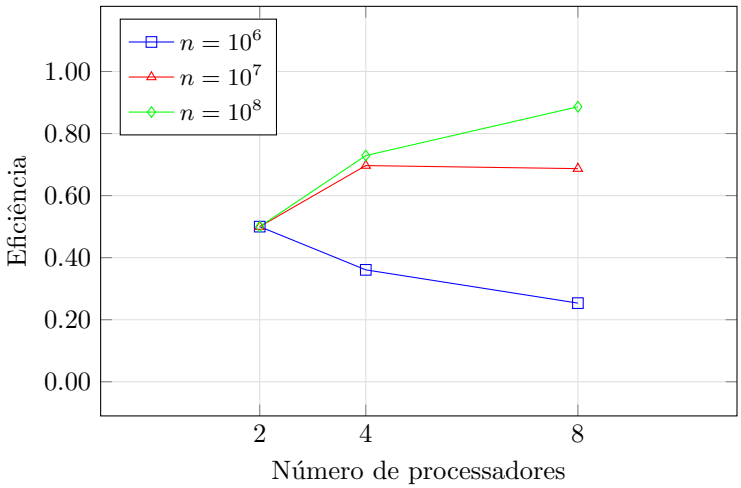

Figure 24: Eficiência vs. Número de processadores - b_Isend_Irecv.c

3.5 b-Ssend

MPI_Ssend e MPI_Recv.

```
/* 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);
            }
```

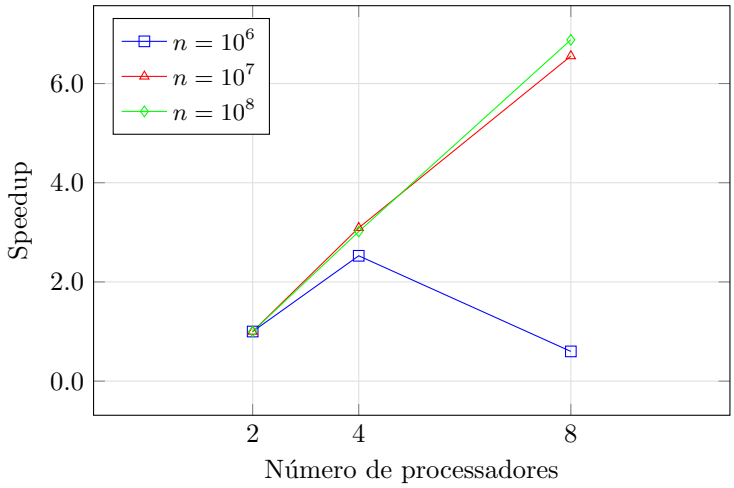

Figure 25: Speedup vs. Número de processadores - b_Ssend

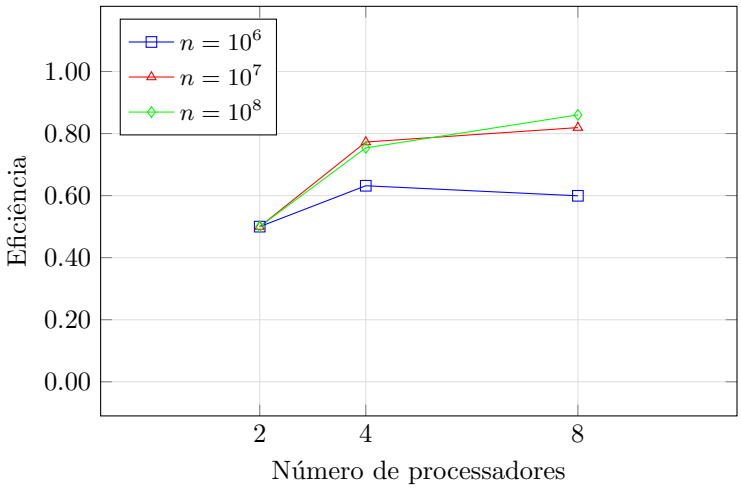

Figure 26: Eficiência vs. Número de processadores - b_Ssend

3.6 b-Ssend_Irecv

MPI_Ssend e MPI_Irecv, MPI_Wait.

```
    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);
```

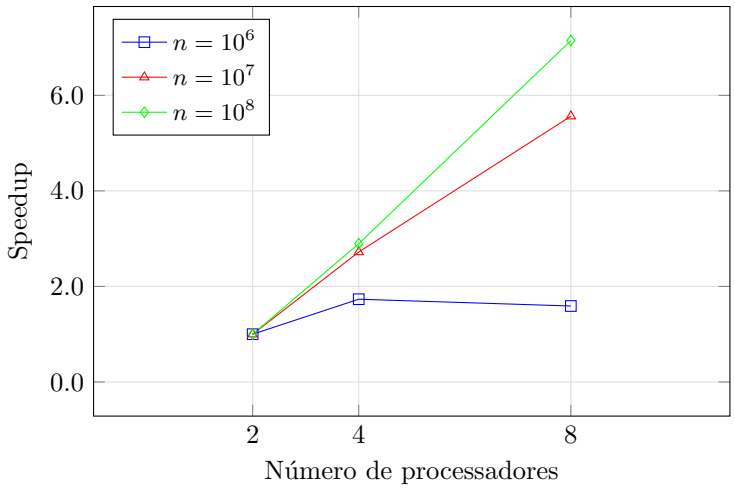

Figure 27: Speedup vs. Número de processadores - b_Ssend_Irecv

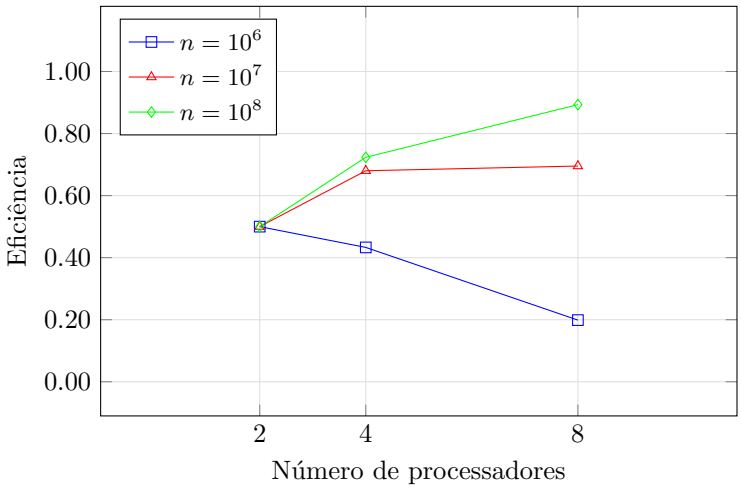

Figure 28: Eficiência vs. Número de processadores - b_Ssend_Irecv