Metody programowania równoległego
Sprawozdanie 1: MPI
Poniedziałek 14.40 24.02.20
Andrzej Małota

1.Komunikacja na 1 nodzie (pamięć dzielona).

W celu obliczenia średniej wartości, zostało wysłanych 10000 wiadomości.

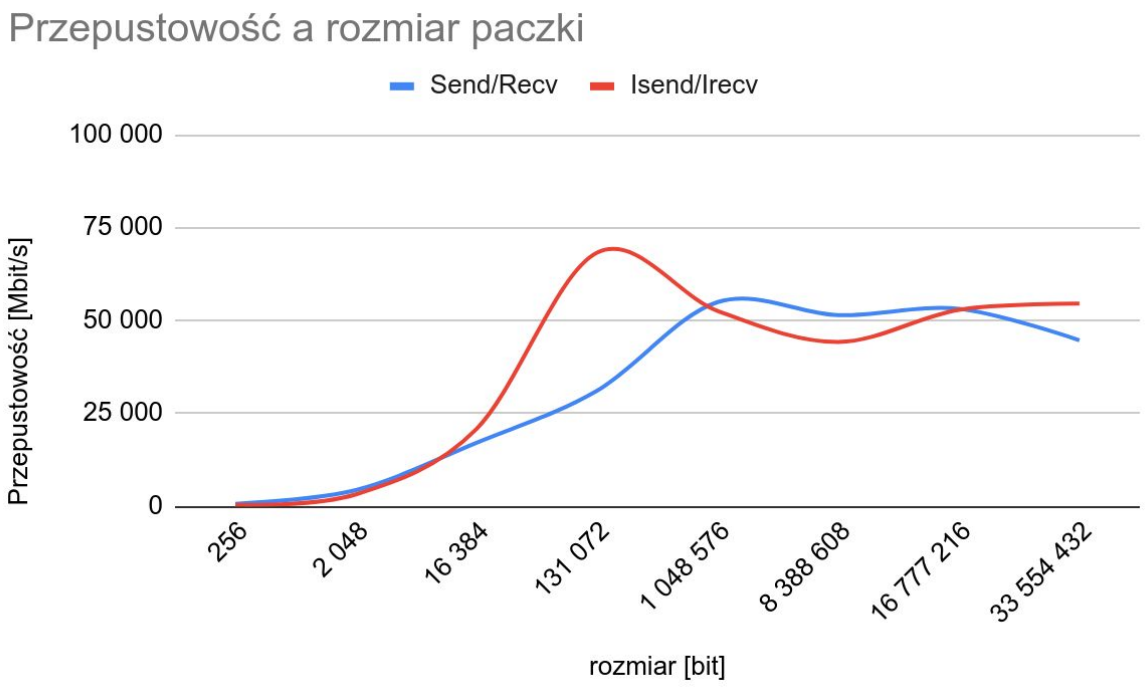
Program włączony na node-04 dla node-04.

Tab 1. Opóźnienia.

Rozmiar [bit]	Opóźnienie dla Send/recv	Opóźnienie dla Isend/Irecv
256	3,3e-07	6,5e-07

Tab 2. Rozmiar a przepustowość.

rozmiar [bit]	Przepustowość dla Send/Recv	Przepustowość dla Isend/Irecv
256	736	422
2 048	4 451	3 287
16 384	17 187	20 911
131 072	31 216	68 371
1 048 576	55 066	52 678
8 388 608	51 549	44 322
16 777 216	53 254	52 988
33 554 432	44 783	54 632



Rys. 1. Przepustowość względem rozmiaru komunikatu.

Można zauważyć że wysyłanie nie synchroniczne osiąga większą przepustowość niż synchroniczne a ponadto szybciej się nasyca.

2. Komunikacja na 1 nodzie (przez sieć).

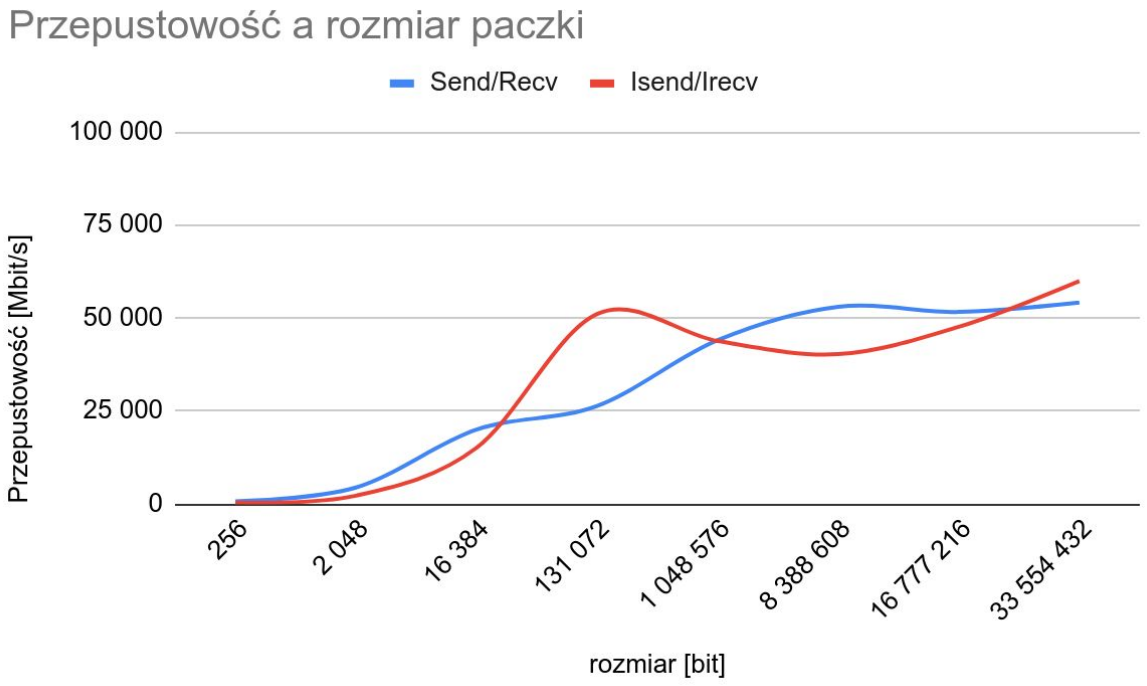
W celu obliczenia średniej wartości, zostało wysłanych 10000 wiadomości.
Program włączony na node-04 dla node-01.

Tab 3. Opóźnienia.

Rozmiar [bit]	**Opóźnienie dla Send/recv**	**Opóźnienie dla Isend/Irecv**
256	5,40E-07	3,30E-07

Tab 4. Rozmiar a przepustowość.

rozmiar [bit]	Przepustowość dla Send/Recv	Przepustowość dla Isend/Irecv
256	776	461
2 048	4 510	2 309
16 384	20 062	15 183
131 072	26 455	51 166
1 048 576	44 144	43 893
8 388 608	53 132	40 368
16 777 216	51 724	47 736
33 554 432	54 264	60 057



Rys. 2. Przepustowość względem rozmiaru komunikatu.

Można zauważyć że tak jak w poprzednim punkcie wysyłanie nie synchroniczne osiąga większą przepustowość niż synchroniczne ale oba wykresy są do siebie dużo bardziej podobne.

3. Komunikacja między 2 nodami na tym samym hoście fizycznym (przez sieć).

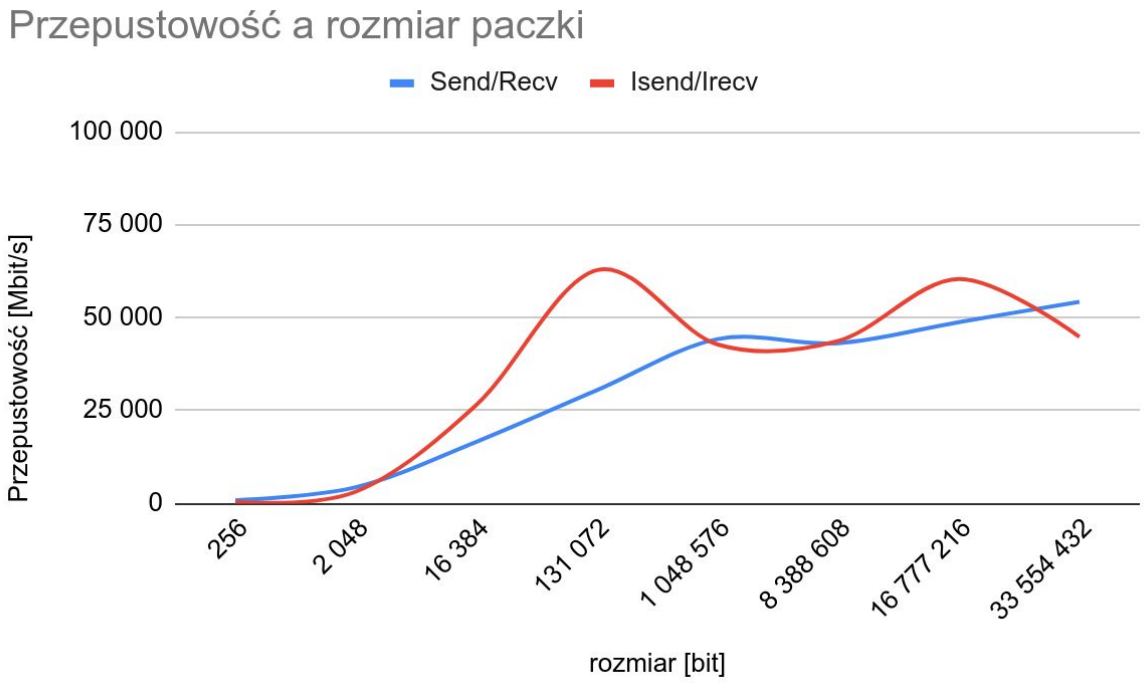
Program włączony na node-04 dla node-01 i node-02.

Tab 5. Opóźnienia.

Rozmiar [bit]	Opóźnienie dla Send/recv	Opóźnienie dla Isend/Irecv
256	2,80E-07	5,89E-07

Tab 6. Rozmiar a przepustowość.

rozmiar [bit]	Przepustowość dla Send/Recv	Przepustowość dla Isend/Irecv
256	882	430
2 048	4 389	3 217
16 384	16 645	26 604
131 072	30 556	62 877
1 048 576	44 288	42 832
8 388 608	43 185	43 798
16 777 216	48 860	60 485
33 554 432	54 339	44 865



Rys. 3. Przepustowość względem rozmiaru komunikatu.

Można zauważyć że tak jak w poprzednich punktach wysyłanie nie synchroniczne osiąga większą przepustowość niż synchroniczne a do tego wykres wysyłania asynchornicznego posiada 2 grzbiety, co wydaje mi się powiązane z obciążeniem maszyny.

4. Komunikacja między 2 nodami na różnych hostach fizycznych (przez sieć).

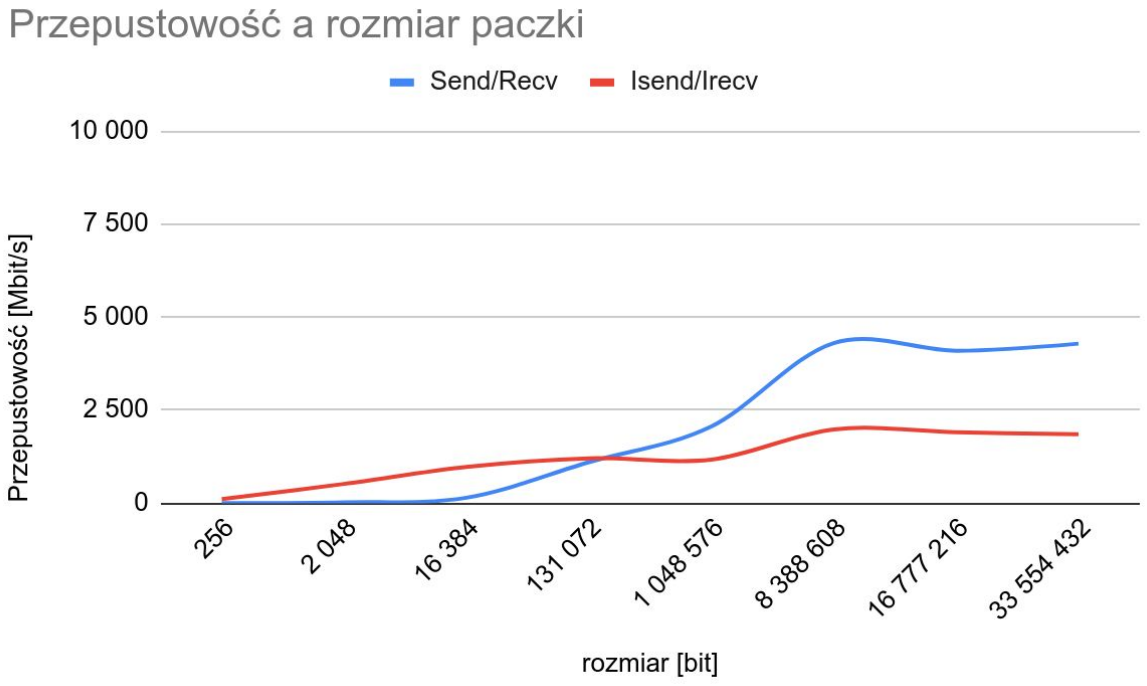
Program włączony na node-04 dla node-11 i node-12.

Tab 5. Opóźnienia.

Rozmiar [bit]	**Opóźnienie dla Send/recv**	**Opóźnienie dla Isend/Irecv**
256	2,80E-07	5,89E-07

Tab 6. Rozmiar a przepustowość.

rozmiar [bit]	Przepustowość dla Send/Recv	Przepustowość dla Isend/Irecv
256	2	113
2 048	24	521
16 384	149	979
131 072	1 105	1 209
1 048 576	2 066	1 175
8 388 608	4 309	1 987
16 777 216	4 102	1 909
33 554 432	4 293	1 853



Rys. 3. Przepustowość względem rozmiaru komunikatu.

Można zauważyć że wysyłanie synchroniczne osiąga większą przepustowość niż asynchroniczne a ponadto obie komunikacje osiągają o wiele mniejsze przepustowości niż w pozostałych punktach.

5. Implementacja.

a. Komunikacja Send/Recv, plik l1_h1.c.

```
#include <stdio.h>
#include "mpi.h"
#include <stdlib.h>

int main(int argc, char** argv)
{
	int rank, size;
	const int N = 10000;
	const int MSG_SIZE = 1048577;
	int msg[MSG_SIZE];
	//int *msg = malloc(MSG_SIZE * sizeof(int));
	int i;
	double  avg_delay, avg_bandwidth;
	double start_t, elapsed_t;

	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &size);

	if (size < 2)
	{
		fprintf(stderr, "Size must be > 2");
		MPI_Abort(MPI_COMM_WORLD, 1);
	}

	MPI_Barrier(MPI_COMM_WORLD);
	if (rank == 0)
	{
		start_t = MPI_Wtime();
		for (i=0; i<N; i++)
		{
			MPI_Send(msg, MSG_SIZE, MPI_INT, 1, 0, MPI_COMM_WORLD);
			MPI_Recv(msg, MSG_SIZE, MPI_INT, 1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
		}
		elapsed_t = MPI_Wtime() - start_t;
		avg_delay = elapsed_t / (2 * N);
		avg_bandwidth = sizeof(msg) / avg_delay * 8 / (1024*1000);
		printf("Size of msg %d bits \n", (int)(sizeof(msg)*8));
		printf("Avg msg delay: %g s \n", avg_delay);
```

```
		printf("Avg bandwidth: %g Mbit/s \n", avg_bandwidth);
	}

	if (rank == 1)
	{
		for (i=0; i<N; i++)
		{
			MPI_Recv(msg, MSG_SIZE, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
			MPI_Send(msg, MSG_SIZE, MPI_INT, 0, 0, MPI_COMM_WORLD);
		}
	}
	//free(msg);
	MPI_Finalize();
	return 0;
}
```

b.Komunikacja Isend/Irecv, plik l1_h1b.c.

```
#include <stdio.h>
#include "mpi.h"
#include <stdlib.h>

int main(int argc, char** argv)
{
	int rank, size;
	const int N = 10000;
	const int MSG_SIZE = 1048576;
	int msg[MSG_SIZE];
	int i;
	double  avg_delay, avg_bandwidth;
	double start_t, elapsed_t;

	MPI_Status stat;
	MPI_Request req1, req2;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &size);

	if (size < 2)
	{
		fprintf(stderr, "Size must be > 2");
		MPI_Abort(MPI_COMM_WORLD, 1);
	}
```

```
	MPI_Barrier(MPI_COMM_WORLD);
	if (rank == 0)
	{
		start_t = MPI_Wtime();
		for (i=0; i<N; i++)
		{
			MPI_Isend(msg, MSG_SIZE, MPI_INT, 1, 0, MPI_COMM_WORLD, &req1);
			MPI_Irecv(msg, MSG_SIZE, MPI_INT, 1, 0, MPI_COMM_WORLD, &req2);
			MPI_Wait(&req1, &stat);
	}
		elapsed_t = MPI_Wtime() - start_t;
		avg_delay = elapsed_t / (2 * N);
		avg_bandwidth = sizeof(msg) / avg_delay * 8 / (1024*1000);
		printf("Size of msg %d bits \n", (int)(sizeof(msg)*8));
		printf("Avg msg delay: %g s \n", avg_delay);
			printf("Avg bandwidth: %g Mbit/s \n", avg_bandwidth);
	}

	if (rank == 1)
	{
		for (i=0; i<N; i++)
		{
			MPI_Isend(msg, MSG_SIZE, MPI_INT, 0, 0, MPI_COMM_WORLD, &req1);
			MPI_Irecv(msg, MSG_SIZE, MPI_INT, 0, 0, MPI_COMM_WORLD, &req2);
			MPI_Wait(&req2, &stat);
		}
	}
	//free(msg);
	MPI_Finalize();
	return 0;
}
```