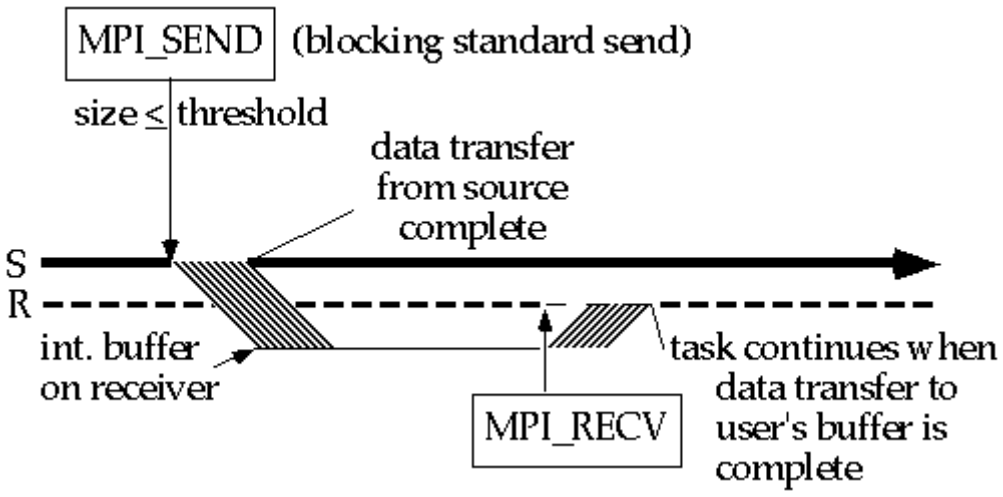
AGH	**Imię i nazwisko:** Bartłomiej Płonka **Kierunek:** Informatyka **Grupa:** wtorek godz. 17:50	**Rok akademicki:** 2015/2016 **Semestr:** 1
Data wykonania: 18-01-2016r.	**Techniki programowania równoległego** *MPI: komunikacja P2P*	

1. CEL ĆWICZENIA

Celem ćwiczenia było zbadanie przepustowości oraz opóźnień występujących w dwóch różnych rodzajach komunikacji typu punkt-punkt. Obiektem badań był klaster *vCluster* znajdujący się na wyposażeniu Akademii Górniczo-Hutniczej. Do badań wykorzystano domyślną implementację protokołu MPI (ang. Message Passing Interface) w klastrze, czyli *MPICH*. Dokonano pomiarów dla komunikacji asynchronicznej i synchronicznej. Wykorzystano jednordzeniowe węzły *vnode-10* oraz *vnode-11.*

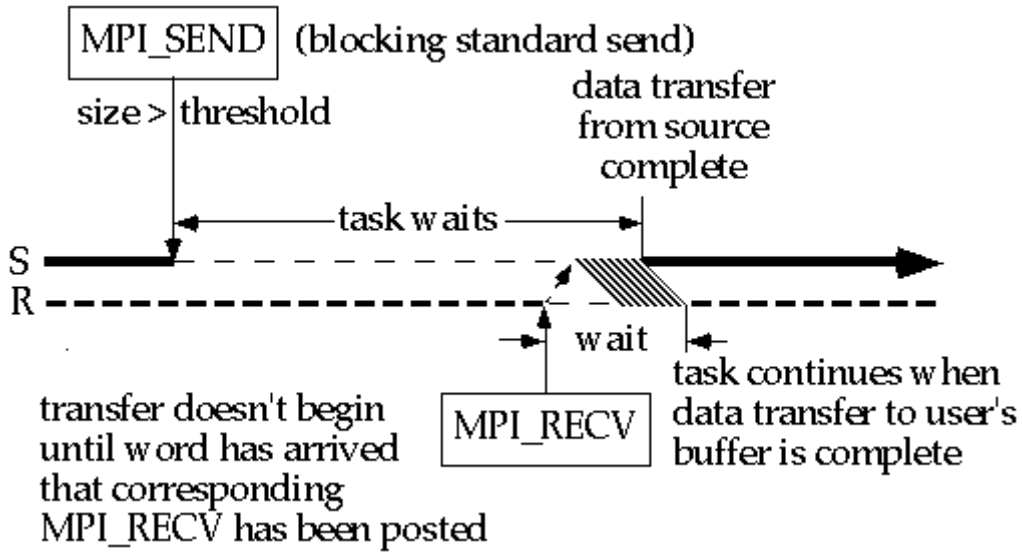
2. WSTĘP TEORETYCZNY

A. Komunikacja standardowa



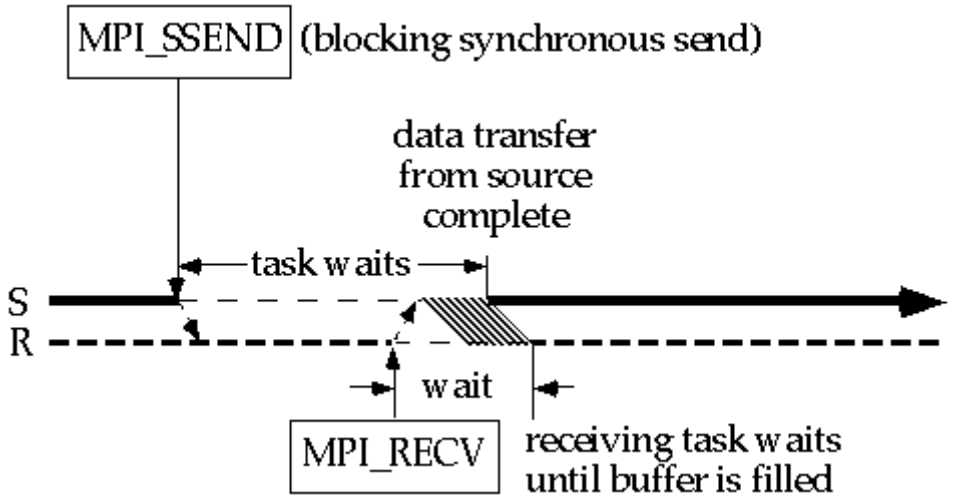
Rysunek 1. Schemat działania komunikacji standardowej przy mniejszej niż progowa wielkości komunikatu.

W komunikacji standardowej proces wysyłający blokuje się na czas przekazania wiadomości do bufora odbiorcy. Jeśli rozmiar wiadomości jest mniejszy niż wielkość bufora, proces wysyłający kontynuuje obliczenia natychmiast po wysłaniu (rys.1). Jeśli jest większy, oczekuje na wywołanie przez odbiorcę funkcji MPI_Recv i rozpoczyna dalsze obliczenia po przekazaniu całej wiadomości (rys.2).



Rysunek 2. Schemat działania komunikacji standardowej przy większej niż progowa wielkości komunikatu.

B. Komunikacja synchroniczna



Rysunek 3. Schemat działania komunikacji synchronicznej.

W komunikacji synchronicznej proces wysyłający oczekuje na gotowość (blokada) procesu odbiorcy do odebrania wiadomości (MPI_Recv). Proces wysyłający kontynuuje obliczenia po przekazaniu całej wiadomości (rys.3).

3. PROGRAMY

A. Komunikacja standardowa

Proces MASTER dla każdego rozmiaru wiadomości z zakresu od 0 do 2kB wykonuje 20 000 przesyłań funkcją blokującą MPI_Send. Odmierzany jest także czas tego działania, a następnie wyznaczana przepustowość ze wzoru:

$$bandwidth = \frac{8 * currentSize * samples}{2^{20} * (t2 - t1)} [Mbps]$$

gdzie: *currentSize – rozmiar wiadomości [B]*

samples – liczba powtórzeń

t1 – czas rozpoczynania operacji [s]

t2 – czas końca operacji [s]

Opóźnienie wyznaczone zostało jako średni czas wykonania funkcji MPI_Send bez żadnej wiadomości z puli 20 000 prób. Wyniki pomiarów zapisywane są do pliku.

Proces SLAVE tylko odbiera komunikaty od procesu MASTER. Wykorzystano blokującą funkcję MPI_Recv. Kod źródłowy obydwu procesów znajduje się w jednym pliku źródłowym, natomiast identyfikacji dokonuje się za pomocą funkcji MPI_Comm_rank.

Kod źródłowy:

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

#define MSG_MAX_LEN     2 * 1024
#define SAMPLES         20000
#define MSG_ID          0
#define MASTER          0
#define SLAVE           1

int main(int argc, char ** argv) {
    char buffer[MSG_MAX_LEN];
    int rank, numprocs, current_size, counter;
    double t1, t2;
    FILE * pFile;

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

```
    if(numprocs != 2) {
        fprintf(stderr, "World size must be 2 for %s\n", argv[0]);
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    if(rank == MASTER) {
        pFile = fopen("std.dat", "w");

        // delay measure
        t1 = MPI_Wtime();
        for(counter = 0; counter < SAMPLES; counter++) {
            MPI_Send(buffer, 0, MPI_CHAR, SLAVE, MSG_ID, MPI_COMM_WORLD);
        }
        t2 = MPI_Wtime();
        fprintf(pFile, "Delay - %f [ms]\n\n", (t2 - t1) * 1000 / SAMPLES);

        // bandwidth measure
        fprintf(pFile, "Message[B]\tBandwidth[Mbps]\n0\t0\n");
        for(current_size = 1; current_size <= MSG_MAX_LEN; current_size++) {
            t1 = MPI_Wtime();
            for(counter = 0; counter < SAMPLES; counter++) {
                MPI_Send(buffer, current_size, MPI_CHAR, SLAVE, MSG_ID, MPI_COMM_WORLD);
            }
            t2 = MPI_Wtime();
            fprintf(pFile, "%d\t%.2f\n", current_size, current_size * SAMPLES / ((t2 - t1) * 131072));
        }
        fclose(pFile);
    }
    else {
        for(counter = 0; counter < SAMPLES; counter++) {
            MPI_Recv(buffer, 0, MPI_CHAR, MASTER, MSG_ID, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        }

        for(current_size = 1; current_size <= MSG_MAX_LEN; current_size++) {
```

```
                for(counter = 0; counter < SAMPLES; counter++) {
                        MPI_Recv(buffer, current_size, MPI_CHAR, MASTER, MSG_ID, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                }
        }

    }

    MPI_Finalize();
}
```

B. Komunikacja synchroniczna

Zasada działania programu jest analogiczna do programu z punktu A, jednakże w tym przypadku testowano funkcję MPI_Ssend oraz zmniejszono liczbę próbek na 1000.

Kod źródłowy:

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

#define MSG_MAX_LEN     2 * 1024
#define SAMPLES         1000
#define MSG_ID          0
#define MASTER          0
#define SLAVE           1

int main(int argc, char ** argv) {
    char buffer[MSG_MAX_LEN];
    int rank, numprocs, current_size, counter;
    double t1, t2;
    FILE * pFile;

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

```
    if(numprocs != 2) {
        fprintf(stderr, "World size must be 2 for %s\n", argv[0]);
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    if(rank == MASTER) {
        pFile = fopen("synch.dat", "w");

        // delay measure
        t1 = MPI_Wtime();
        for(counter = 0; counter < SAMPLES; counter++) {
            MPI_Ssend(buffer, 0, MPI_CHAR, SLAVE, MSG_ID, MPI_COMM_WORLD);
        }
        t2 = MPI_Wtime();
        fprintf(pFile, "Delay - %f [ms]\n\n", (t2 - t1) * 1000 / SAMPLES);

        // bandwidth measure
        fprintf(pFile, "Message[B]\tBandwidth[Mbps]\n0\t0\n");
        for(current_size = 1; current_size <= MSG_MAX_LEN; current_size++) {
            t1 = MPI_Wtime();
            for(counter = 0; counter < SAMPLES; counter++) {
                MPI_Ssend(buffer, current_size, MPI_CHAR, SLAVE, MSG_ID,
MPI_COMM_WORLD);
            }
            t2 = MPI_Wtime();
            fprintf(pFile, "%d\t%.2f\n", current_size, current_size * SAMPLES / ((t2 - t1) * 131072));
        }
        fclose(pFile);
    }
    else {
        for(counter = 0; counter < SAMPLES; counter++) {
            MPI_Recv(buffer, 0, MPI_CHAR, MASTER, MSG_ID, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        }

        for(current_size = 1; current_size <= MSG_MAX_LEN; current_size++) {
```

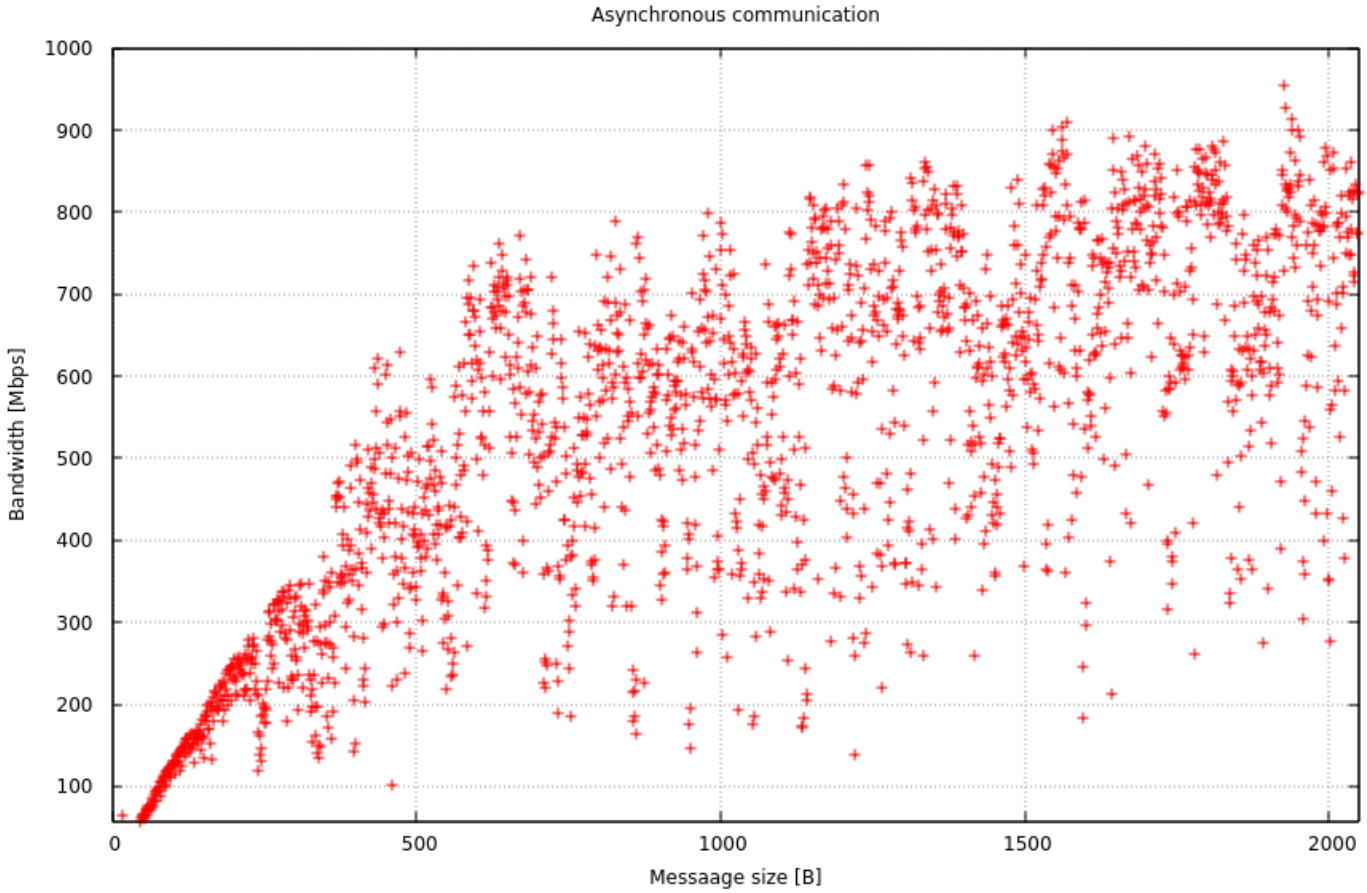
```
                    for(counter = 0; counter < SAMPLES; counter++) {
                        MPI_Recv(buffer, current_size, MPI_CHAR, MASTER, MSG_ID,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                    }
                }

    }

    MPI_Finalize();
}
```

4. WYNIKI POMIARÓW

A. Komunikacja stanardowa

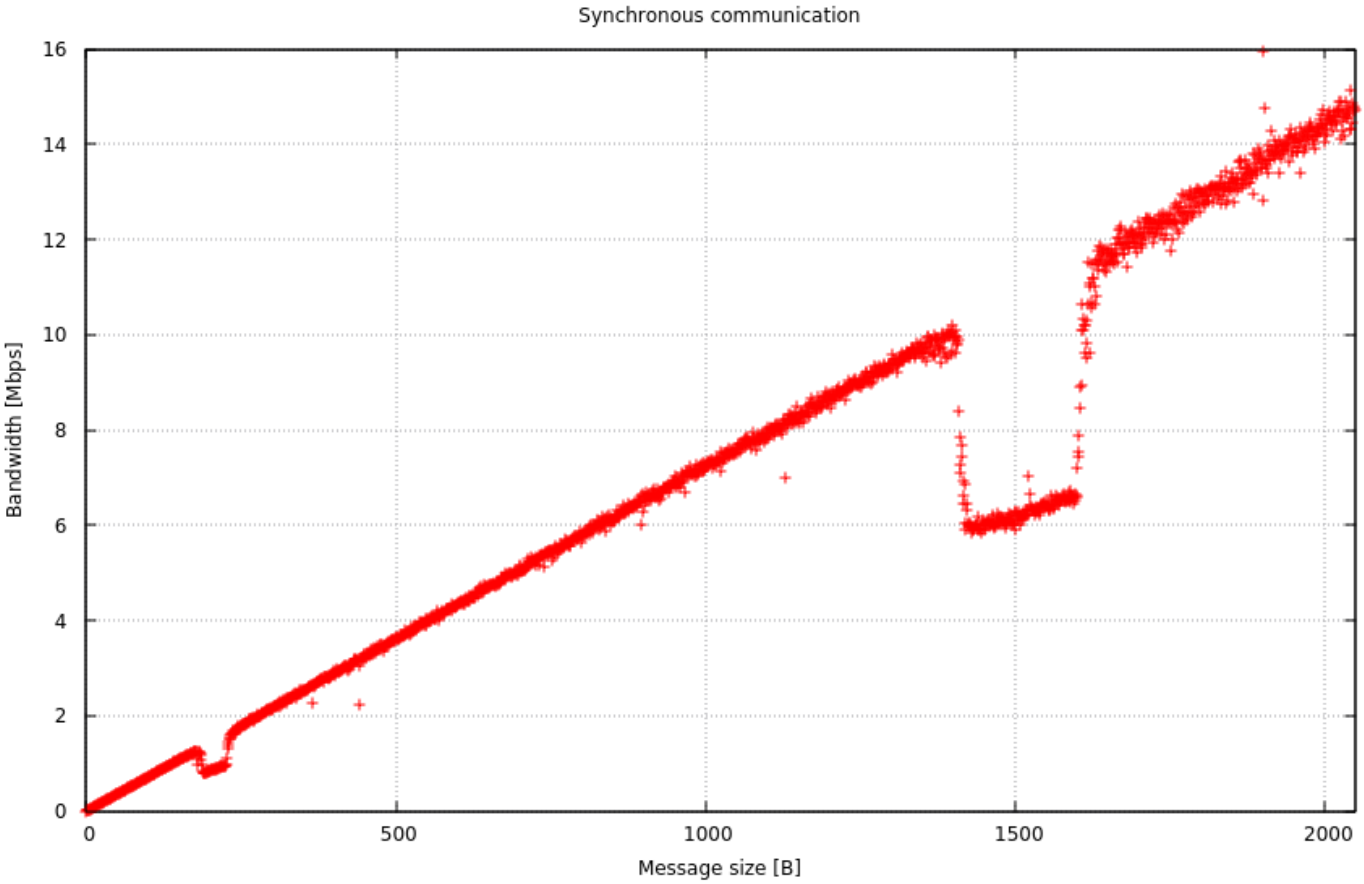


Rysunek 4. Zależność przepustowości połączeń w zależności od rozmiaru wiadomości.

Opóźnienie wyniosło niewiele, bo zaledwie 1µs. Wyniki pomiarów przepustowości przedstawia rys. 4. Można zauważyć deterministyczność przepustowości w zakresie rozmiaru wiadomości od 0 do ok 250 bajtów.

Zwiększając rozmiar wiadomości zwiększamy rozrzut próbek, mimo że każda z nich została obliczona na podstawie 20 000 zbioru wywołań funkcji MPI_Send. Ma to związek z mechanizmem blokowania procesu wywołującego MPI_Send do momentu wywołania przez odbiorcę funkcji MPI_Recv przy rozmiarach wiadomości przekraczających wielkość bufora odbiorcy. Zaletą takiej komunikacji jest duża przepustowość która dla pewnych próbek przekraczała 900 Mbps.

B. Komunikacja synchroniczna



Rysunek 5. Zależność przepustowości połączeń w zależności od rozmiaru wiadomości.

Opóźnienie komunikacji synchronicznej wyniosło aż 1 [ms]. Ma to związek z synchronizacją procesu wywołującego do momentu, w którym proces odbierający będzie gotów na otrzymanie wiadomości. Wyniki pomiarów przepustowości przedstawia rys. 5. Można zauważyć dużą deterministyczność pomiarów w podanym zakresie. Przepustowość rośnie liniowo wraz ze wzrostem rozmiaru wiadomości. Pojawiają się natomiast pewne anomalie w okolicach rozmiaru wiadomości równego ok 200 i 1500 bajtów. Najprawdopodobniej miało to związek z chwilowymi przeciążeniami węzłów klastra.

5. WNIOSKI

Na podstawie przeprowadzonych pomiarów można zauważyć pewne zalety i wady komunikacji normalnej i synchronicznej. Komunikacja normalna zapewnia znacznie większe przepustowości oraz opóźnienia w przesyłaniu komunikatów. Wadą jest słaby determinizm przepustowości. Pod tym względem komunikacja synchroniczna jest lepsza, dodatkowo daje informację o synchronizacji procesów, co w wielu obliczeniach na kolosalne znaczenie. Wadą jest utrata przepustowości i duże opóźnienia