FFT - Fast Fourier Transform

Użycie

W katalogu projektu:

```
$ source /opt/nfs/config/source_bupc_2022.10.sh
# albo
$ source /opt/nfs/config/source_bupc.sh

# Kompilacja i uruchamianie:
$ make compile BLOCKSIZE=8192
$ make run
```

Uwaga: podane `BLOCKSIZE` musi być równe $\frac{\text{najmniejsza potęga 2 podzielna większa niż długość danych}}{\texttt{PTHREADS}\cdot\texttt{NODES}}$ (wynika to z faktu, że aby wspólne tablic były dzielone między wątki na przylegające obszary - czego wymaga algorytm - a nie metodą *round-robin*, wielkość bloku przekazywana w kodzie musi być stałą czasu kompilacji). Przykładowo dla pliku wejściowego `example`, który zawiera 65536 punktów danych i $\texttt{BLOCKSIZE} = 4096$ iloczyn $\texttt{PTHREADS} \cdot \texttt{NODES}$ musi być równy 16 żeby zagwarantować poprawne działanie programu.
Specyfikacja pliku wejściowego / liczby node'ów / liczby wątków:

```
$ make run PTHREADS=4 NODES=4 INPUT=example2
```

gdzie `PTHREADS` odpowiada liczbie wątków na node (domyślnie i zalecanie 4), a `NODES` liczbie wykorzystanych node'ów. Całkowitą liczbą wątków będzie ich iloczyn.
Uwaga: obie liczby **muszą** być potęgami 2.
Generacja nowego pliku wejściowego:

```
$ python3 signalGenerate.py liczba_pkt nazwa_pliku
```

Do nazwy pliku zostanie dodany przedrostek `generated_`.

Porównanie wykresów dla danych i dla otrzymanych wyników:

```
$ python3 compare.py nazwa_pliku_expected fft_output
```

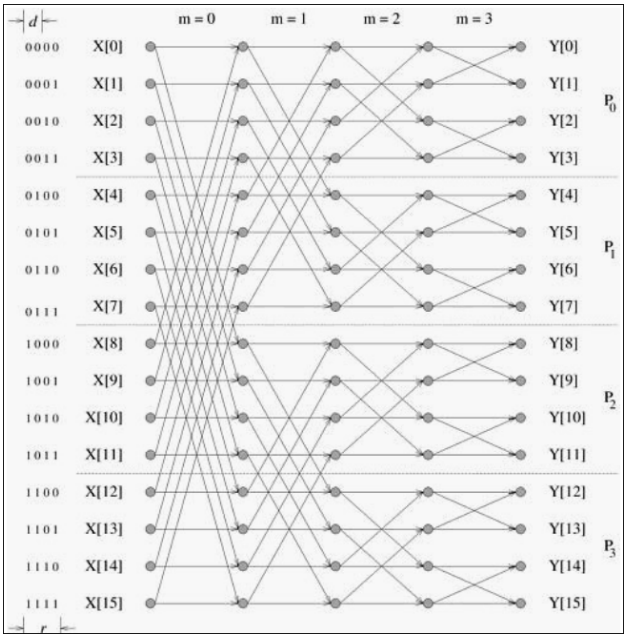
Ze względu na żmudność liczenia `BLOCKSIZE`, poniżej przedstawiono kilka przykładowych działających wywołań dla plików wejściowych `example` i `example2` (drugi jest dwukrotnie krótszy niż pierwszy):

```
# Z domyslnym PTHREAD=4:
$ make compile BLOCKSIZE=4096
$ make run INPUT=example NODES=4
$ make run INPUT=example2 NODES=2

$ make clean

$ make compile BLOCKSIZE=2048
$ make run INPUT=example NODES=8
$ make run INPUT=example2 NODES=4
```

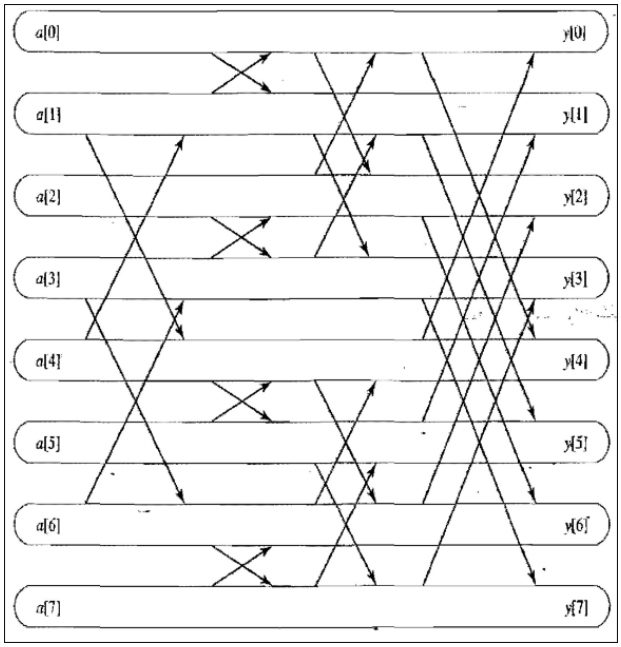
Działanie i budowa programu

Zaimplementowaliśmy wersję *radix-2 Cooley–Tukey FFT*, dodatkowo z wykorzystaniem *bit-reversal permutation* dla zoptymalizowania użycia pamięci. Na figurze 1 umieszczono teoretyczne diagramy tego algorytmu zaczerpnięte z literatury (na części *a)* bez *bit-reversal*, a na części *b)* z. Zdecydowaliśmy się na wybór technologii *Berkeley UPC* jako implementacja *PGAS*, ponieważ pozwala ona na implementację algorytmu w praktycznie identyczny sposób jak zwyczajne wersje wielowątkowe dostępne w literaturze. Na figurze 2 znajduje się schemat z działaniem programu w wersji `MPI`, a na figurze 3 schemat poglądowy z

samą częścią algorytmiczną wersji `UPC`. Jak widać jest on praktycznie identyczny jak schematy poglądowe z literatury.



(a) standardowe *FFT* (Grama, Gupta)



(b) *bit-reversed FFT* (Quinn)

Figura 1: 2 Figures side by side

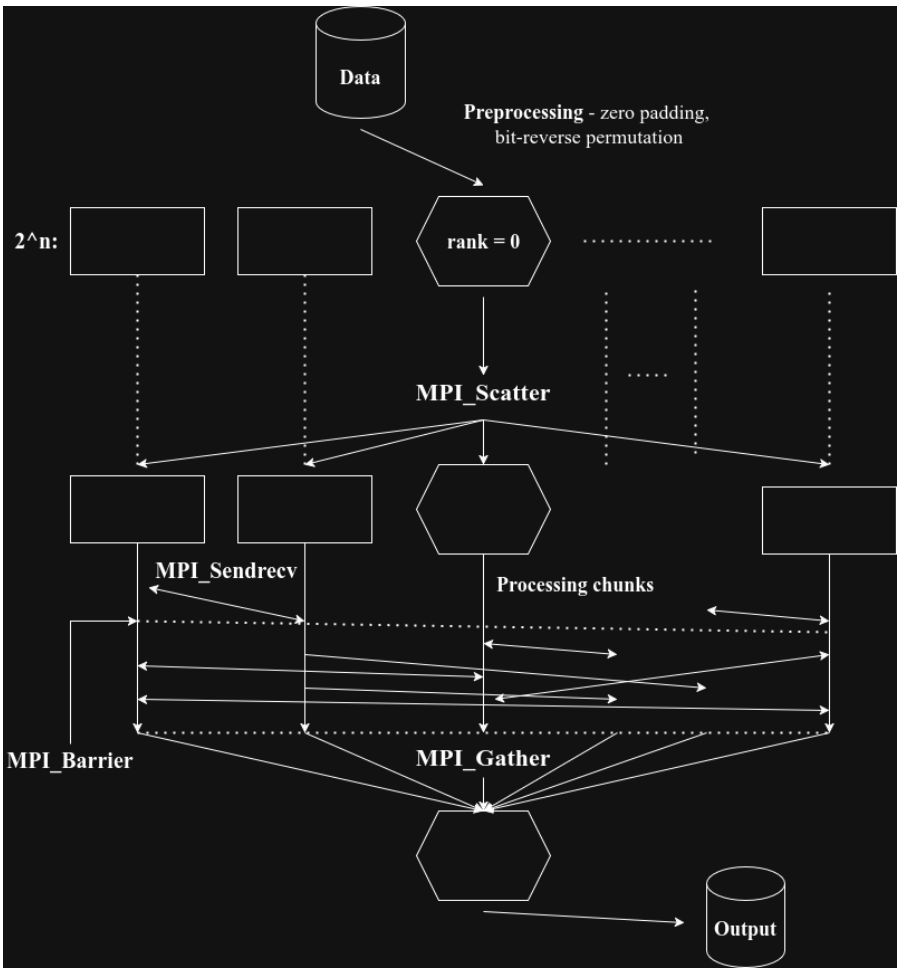


Figura 2: Przybliżony schemat blokowy programu MPI

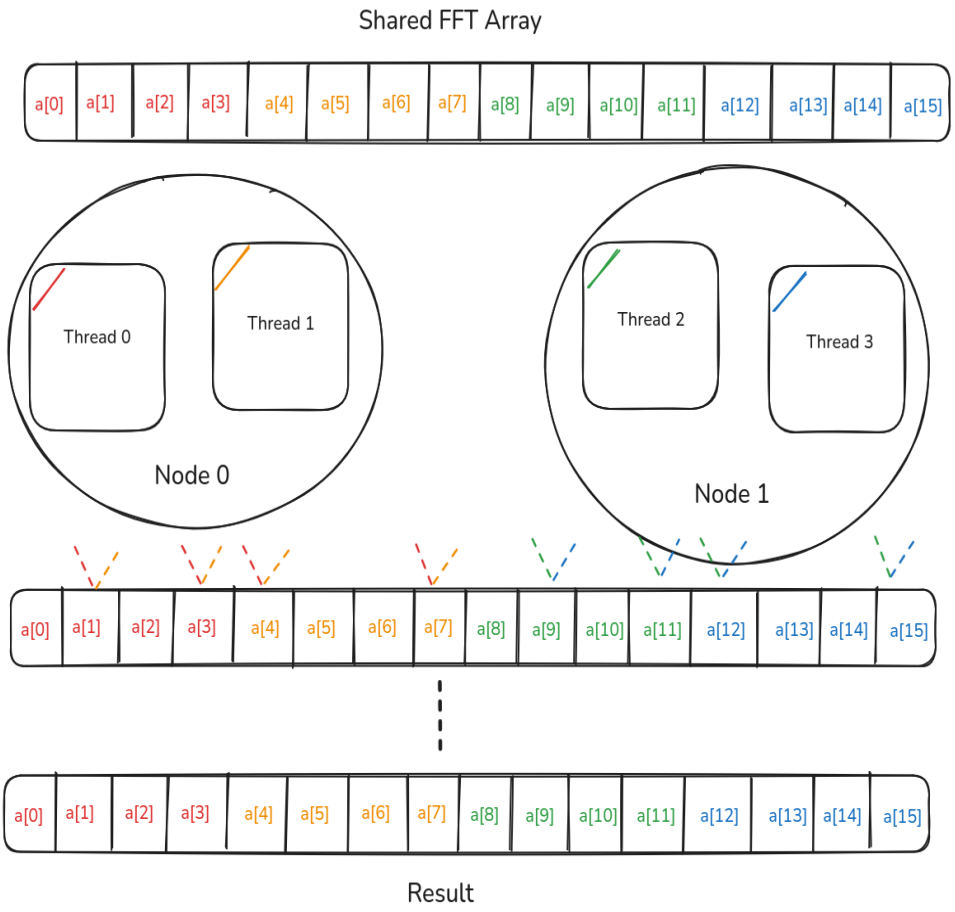


Figura 3: Przybliżony schemat blokowy programu UPC

Krótki opis programu

Pogram na wejściu przyjmuje wygenerowany plik zawierający liczbę próbek n, częstotliwość próbkowania i n próbek z wygenerowanago sygnału jednowymiarowego. Następnie, wątek 0 dokonuje paddingu próbek zerami, tak żeby nowe n było najbliższą, większą lub równą potegą 2 - algorytm `FFT` jest algorytmem typu dziel i zwyciężaj, który w każdym kroku dzieli problem na 2 dokładnie dwa razy mniejsze, więc dodawanie niekosztowynych danych (zera nie wpływają na wynik, poza jego potencjalną reprezentacją) jest warte łatwiejszych obliczeń na potęgach 2. Następnie, wykonywana jest permutacja *bit-reversal* wektora próbek w celu zmaksymalizowania dostępu do lokalnych danych przez każdy wątek i zminimalizowania dostępu do zdalnych danych (różne części wektora próbek).Dane przechowywane są w dzielonej tablicy, gdzie każdy wątek jest odpowiedzialny za obliczanie i zapis zmiennych do swojego fragmentu. Wątki wykonują swoją część algorytmu `FFT` w wersji iteracyjnej (a zatem *bottom-up*), sumując odpowiednie pary wartości (początkowo próbek) zgodnie ze standardowym algorytmem `FFT`. W tej wersji dodaliśmy podział na tablice do odczytu i zapisu (zamieniane na każdym poziomie `FFT`) w celu uniknięcia race condition na odczyt/zapis do tablicy (nie ma tu wzajemnego blokowania na odczyt/zapis, który występował w wersji `MPI` przy operacji `MPI_Sendrecv`). Wątki obliczają i zapisują wartości tylko do swoich części tablicy dzięki użyciu dyrektywy `upc_forall`. Program jest blokowany - `upc_barrier;` - po każdej iteracji w celu synchronizacji danych w tablicy zapisu. Gdy w każdym procesie iteracja dobiegnie końca, wątek 0 kopiuję wartości z tablicy wspólnej do lokalnej, a wynikowy wektor zostaje zapisywany w postaci par części rzeczywistych i urojonych wyniku.