파이선 병렬프로그래밍: 수치 해석 예제 실습

2024.10.

한국과학기술정보연구원 강지훈

필요 패키지

- mpi4py
- numpy
- random
- scikit-learn
- matplotlib

---

1. 벡터와 행렬 연산 (I)

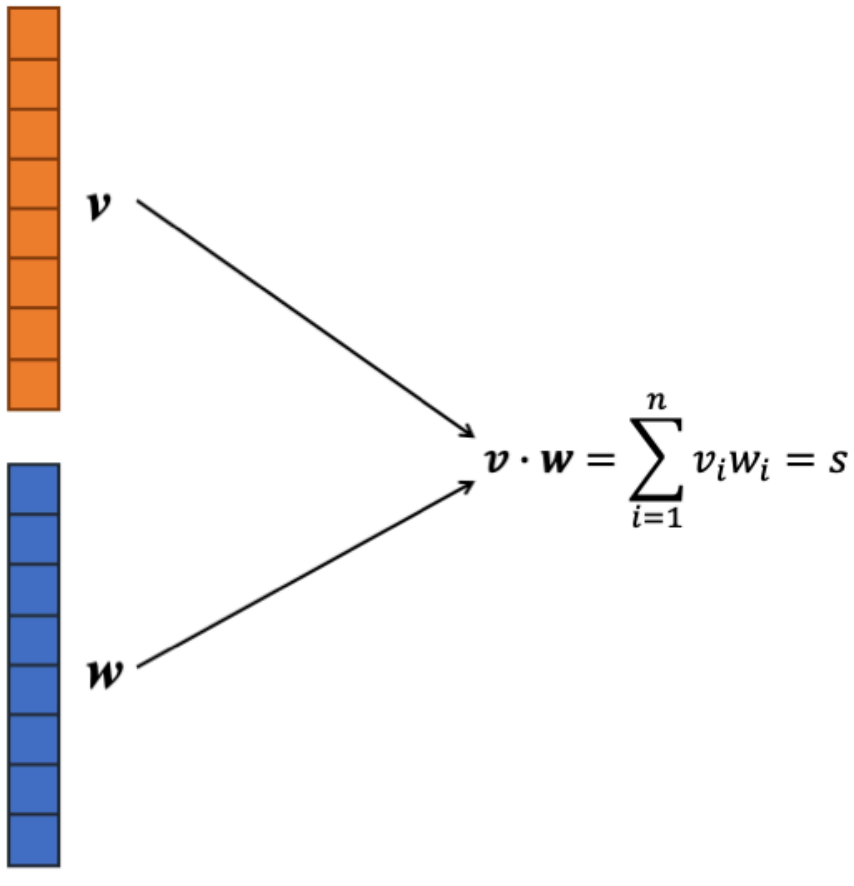
1.1. 행렬/벡터 만들기

In [ ]:
```
!mkdir examples
```

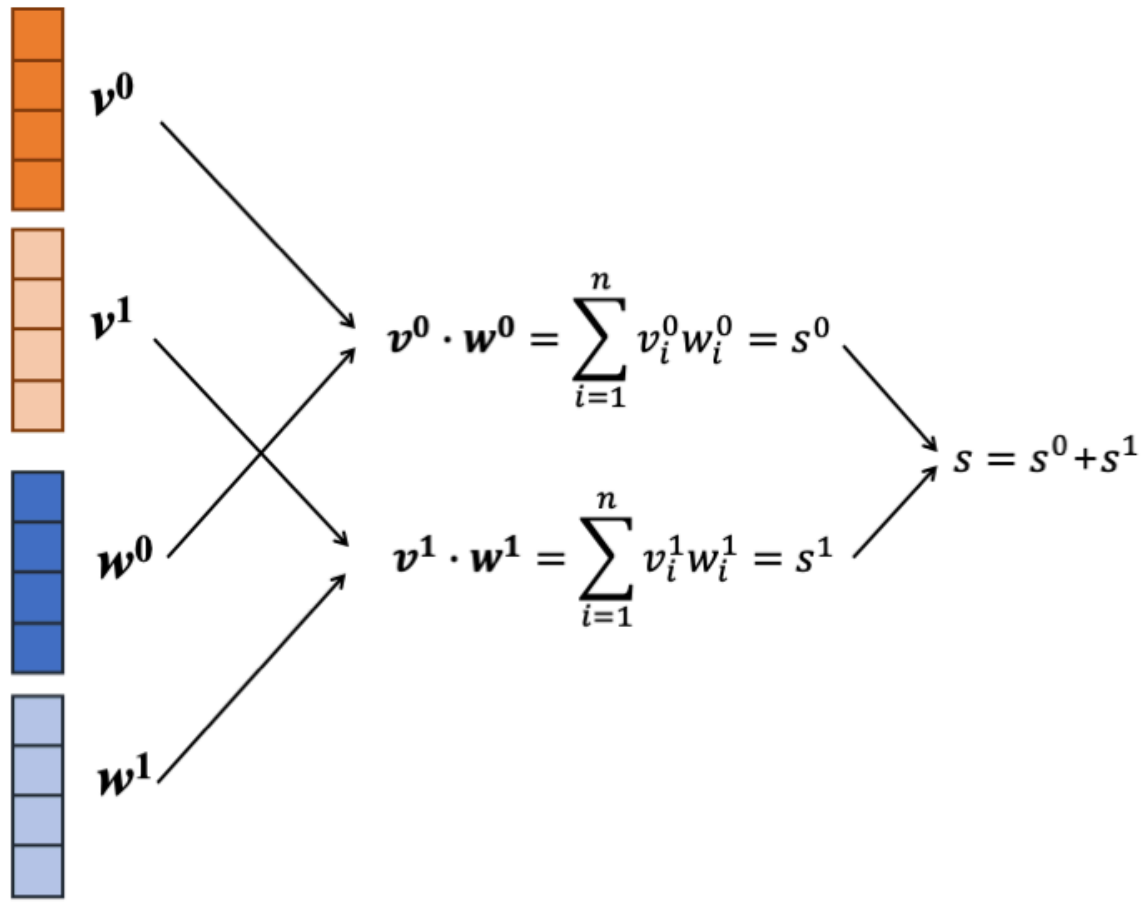
In [2]:
```
import numpy as np

np.set_printoptions(linewidth=np.inf)

n = 10

A = np.random.rand(n, n)
B = np.random.rand(n, n)
v = np.random.rand(n)
w = np.random.rand(n)

np.save("examples/A", A)
np.save("examples/B", B)
np.save("examples/v", v)
np.save("examples/w", w)
```

1.2. 벡터 내적

1. 순차코드



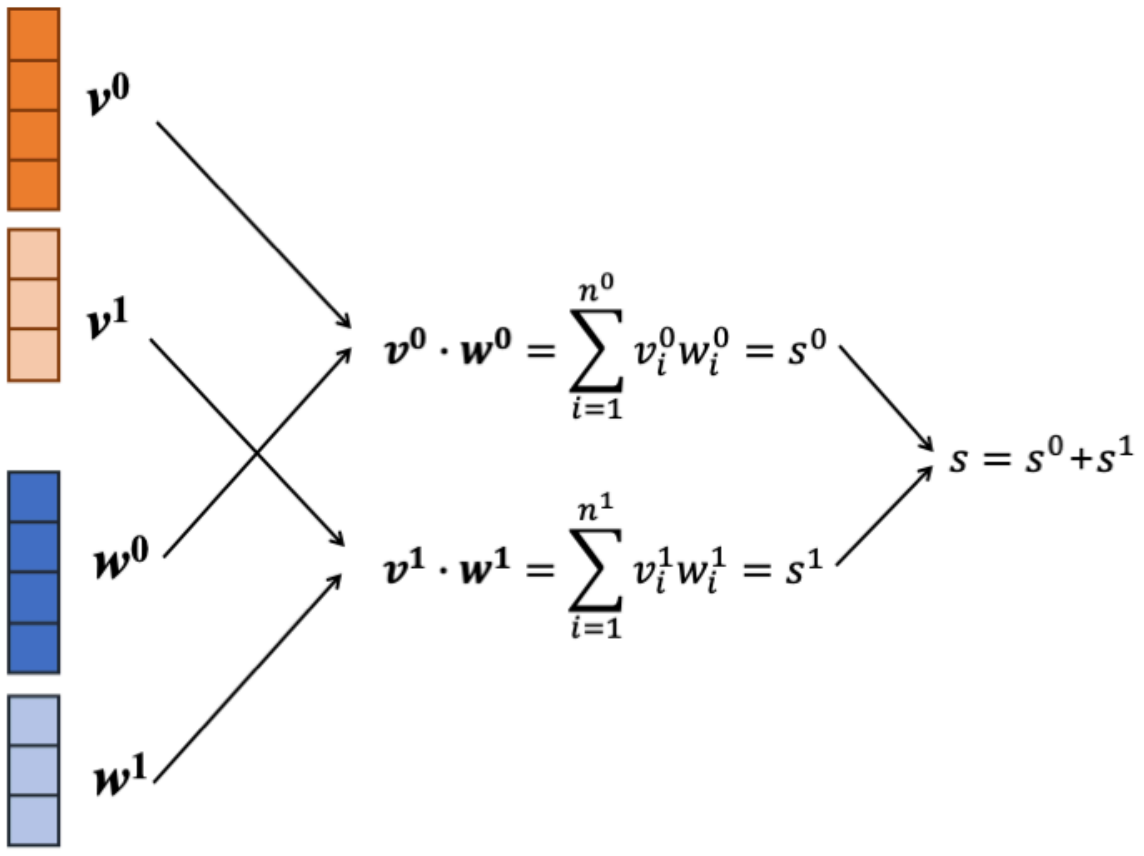
2. 병렬코드 - 등분할



In [ ]:
```
%%writefile examples/v.py
import numpy as np
from mpi4py import MPI

np.set_printoptions(linewidth=np.inf,precision=3)

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0:
    v = np.load("examples/v.npy")
    w = np.load("examples/w.npy")
    n = v.size
else :
    v = None
    w = None
    n = 0

n = comm.bcast(n, root = 0)


##### n_local 크기 정하기
n_local = int(n / size) # FIX ME

##### 분할된 n_local 크기만큼 배열 생성
v_local = np.empty(n_local, dtype = np.float64)
w_local = np.empty(n_local, dtype = np.float64)

##### Scatter 함수로 벡터 분할
comm.Scatter(v, v_local, root = 0) # FIX ME
comm.Scatter(w, w_local, root = 0) # FIX ME

##### 프로세스별 Local sum
s_local = np.dot(v_local, w_local) # FIX ME

##### reduce를 이용한 Global sum
s_global = comm.allreduce(s_local, MPI.SUM) # FIX ME

#if rank == 1:
print(rank, s_global)
```

In [ ]:
```
! mpiexec -np 2 python examples/v.py
```

3. 병렬코드 - 비등분할



In [ ]:
```
%%writefile examples/v_var.py
import numpy as np
from mpi4py import MPI

##### 시작점과 끝점의 인덱스를 반환
def para_range(n, size, rank) :
    iwork = divmod(n, size)
    ista = rank * iwork[0] + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1
    return ista, iend

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0:
    v = np.load("examples/v.npy")
    w = np.load("examples/w.npy")
    n = v.size
else :
    v = None
    w = None
    n = 0

##### 벡터의 전체 크기를 broadcast
n = comm.bcast(n, root = 0)

##### 프로세스별 범위 할당
ista, iend = para_range(n, size, rank) # FIX ME
n_local = iend - ista + 1# FIX ME


##### Scatterv를 위해 n_local로부터 n_local_cnts 리스트 생성
n_local_cnts = comm.gather(n_local, root = 0)

v_local = np.empty(n_local, dtype = np.float64)
w_local = np.empty(n_local, dtype = np.float64)

##### n_local_cnts 리스트를 이용하여 Scatter로 벡터의 비균등 할당
comm.Scatterv((v, n_local_cnts), v_local, root = 0) #FIX ME
comm.Scatterv((w, n_local_cnts), w_local, root = 0) #FIX ME

##### 분할된 벡터의 내적
s_local = np.dot(v_local,w_local)

##### reduce를 이용한 Global sum
s_global = comm.reduce(s_local, MPI.SUM) #FIX ME

if rank == 0:
    print(n_local_cnts)
    print(s_global)
```

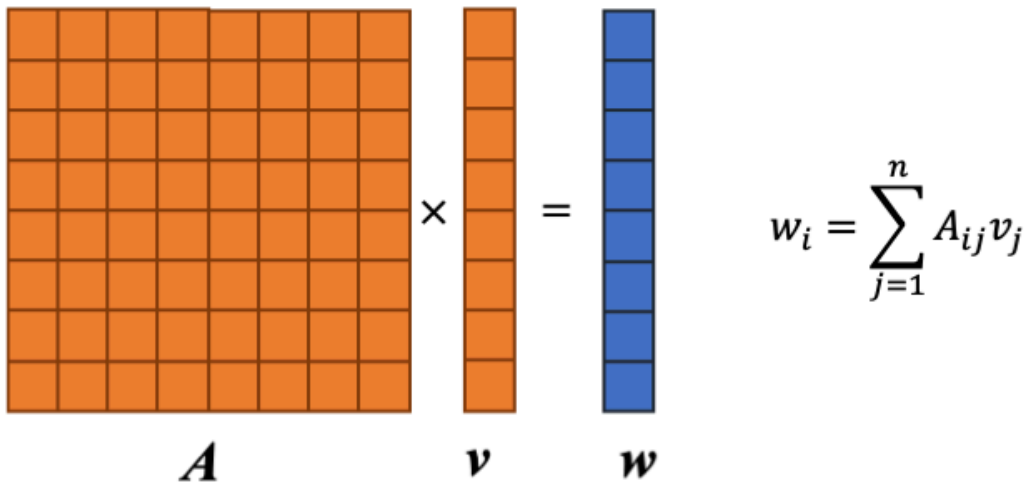
In [ ]:
```
! mpiexec -np 3 python examples/v_var.py
```

4. para_range 저장

In [ ]:
```
%%writefile examples/tools.py

def para_range(n, size, rank) :
    iwork = divmod(n, size)
    ista = rank * iwork[0] + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1
    return ista, iend
```

1.3. 행렬-벡터곱

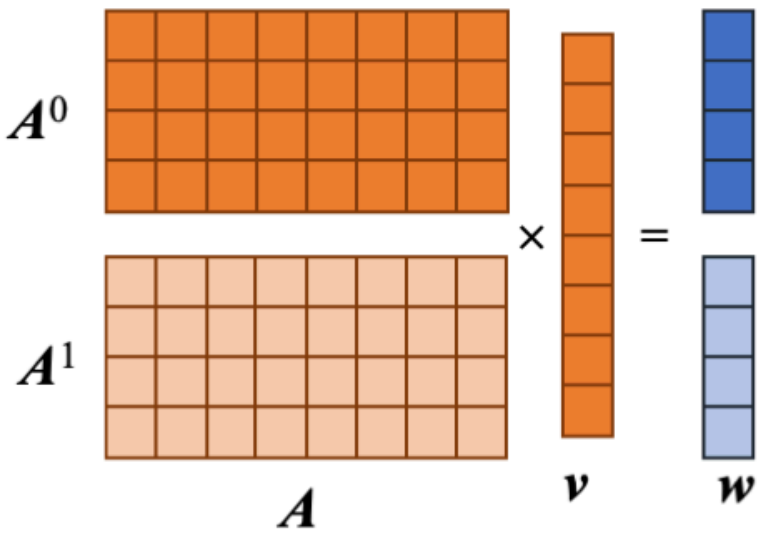
1. 순차코드



$$w_i = \sum_{j=1}^{n} A_{ij} v_j$$

In [ ]:
```
A = np.load("examples/A.npy")
v = np.load("examples/v.npy")

b = np.matmul(A,v)
print (b)
```

2. 행렬의 행 등분할

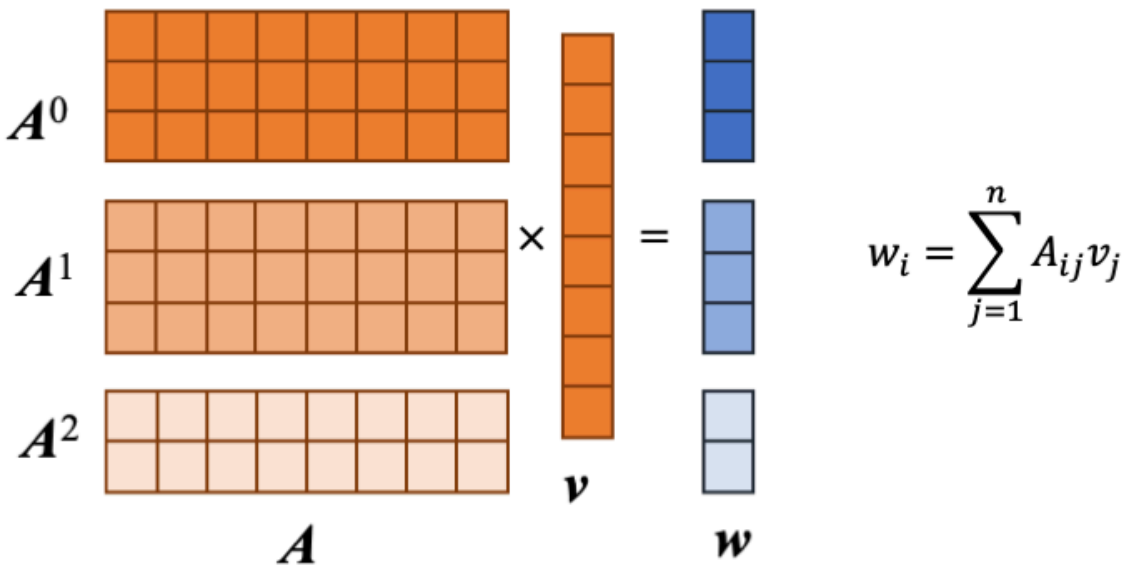


$$w_i = \sum_{j=1}^{n} A_{ij} v_j$$

In [ ]:
```
%%writefile examples/Av.py

import numpy as np
from mpi4py import MPI

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

##### 행렬 및 벡터 불러오기
##### 크기 n은 broadcast
if rank == 0 :
    A = np.load("examples/A.npy")
    v = np.load("examples/v.npy")
    n = v.size
    n = comm.bcast(n, root = 0)
else :
    A = None
    n = 0
    n = comm.bcast(n, root = 0)
    v = np.empty(n, dtype = np.float64)

##### n_local 크기 정하기
n_local = int(n / size)

##### n_local 만큼 부분배열 선언
A_local = np.empty((n_local, n), dtype = np.float64)

##### 행렬의 행 분할
comm.Scatter(A, A_local, root = 0) #FIX ME

##### 벡터 v는 broadcast
comm.Bcast(v, root = 0)

##### 분할된 행렬과의 연산
b = np.matmul(A_local, v) #FIX ME

print(b, rank)
```

In [ ]:
```
! mpiexec -np 2 python examples/Av.py
```

3. 행렬의 행 비등분할

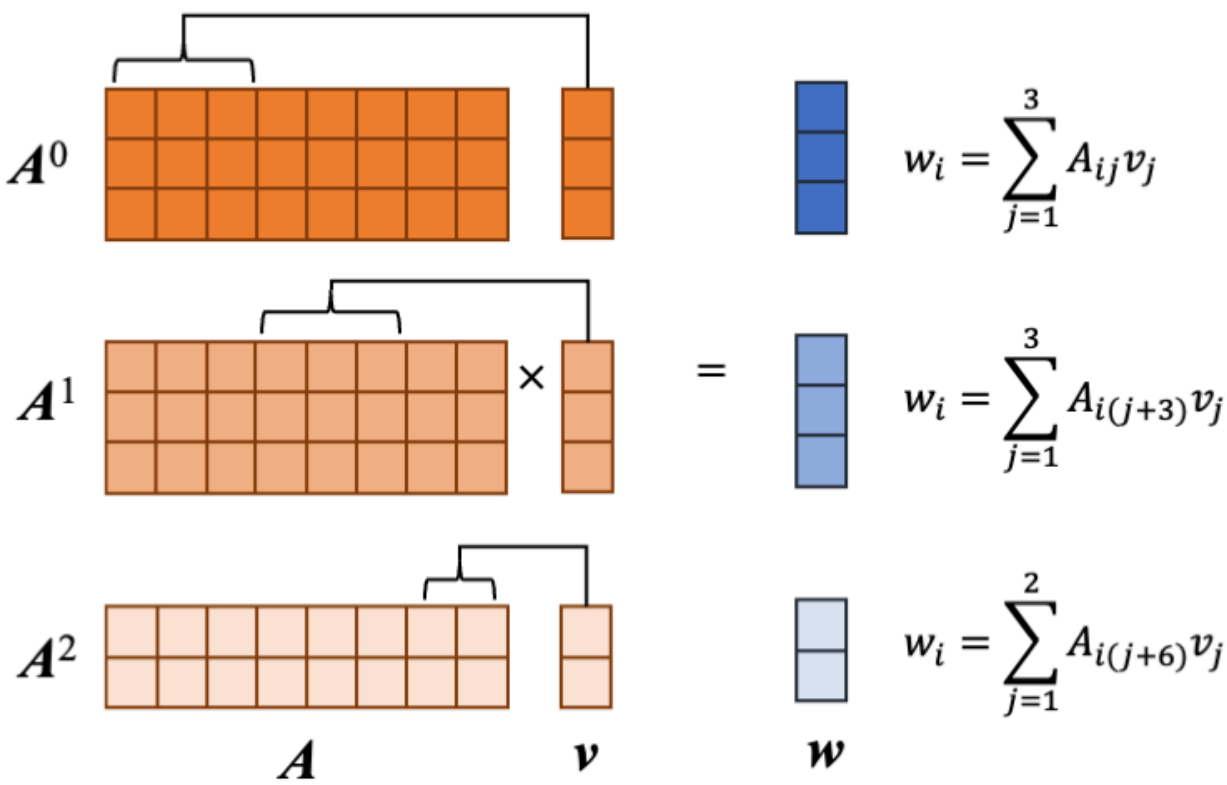


In [ ]:
```
%%writefile examples/Avar.py

# Matrix A의 Row decomposition

import numpy as np
from mpi4py import MPI
from tools import para_range

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
```

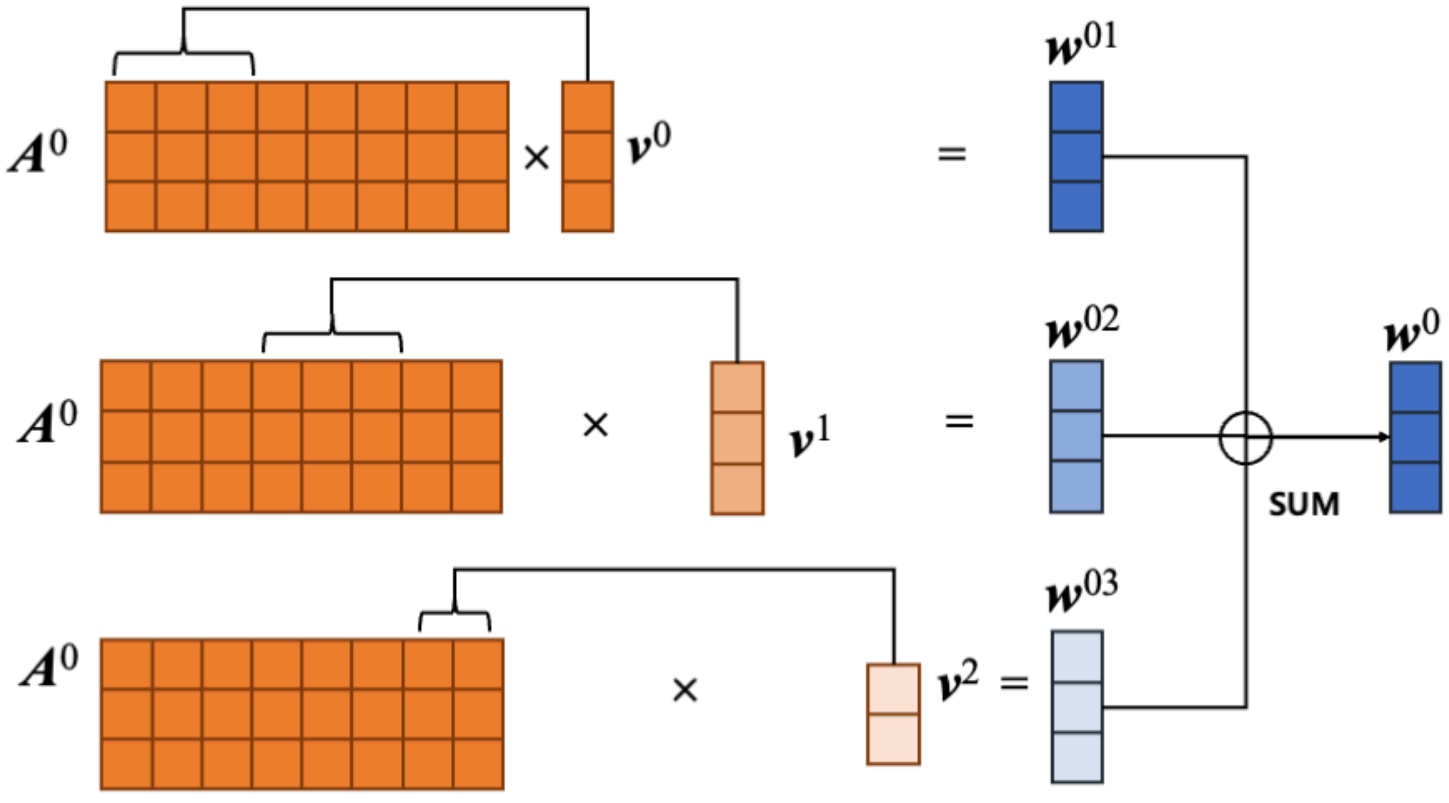
```python
size = comm.Get_size()

##### 행렬 및 벡터 불러오기
##### 크기 n은 broadcast
if rank == 0 :
    A = np.load("examples/A.npy")
    v = np.load("examples/v.npy")
    n = v.size
    n = comm.bcast(n, root = 0)

else :
    A = None
    n = 0
    n = comm.bcast(n, root = 0)
    v = np.empty(n, dtype = np.float64)

##### 프로세스별 범위 할당
ista, iend = para_range(n, size, rank)

##### 프로세스별 행수 설정
n_local = (iend - ista + 1)

##### 분할된 행렬 선언
A_local = np.empty((n_local, n), dtype = np.float64)

##### Scatterv를 위해 n_local를 이용한 리스트 생성. 이 때 부분행렬 행수가 아닌 전체 크기를 계산
n_local_chunks = comm.gather(n_local * n, root = 0) #FIX ME

##### 행렬의 행 분할
comm.Scatterv((A, n_local_chunks), A_local, root = 0) #FIX ME

comm.Bcast(v, root = 0)

b = np.matmul(A_local,v)

print(b, rank)
```

In [ ]:
```
! mpiexec -np 3 python examples/Avar.py
```

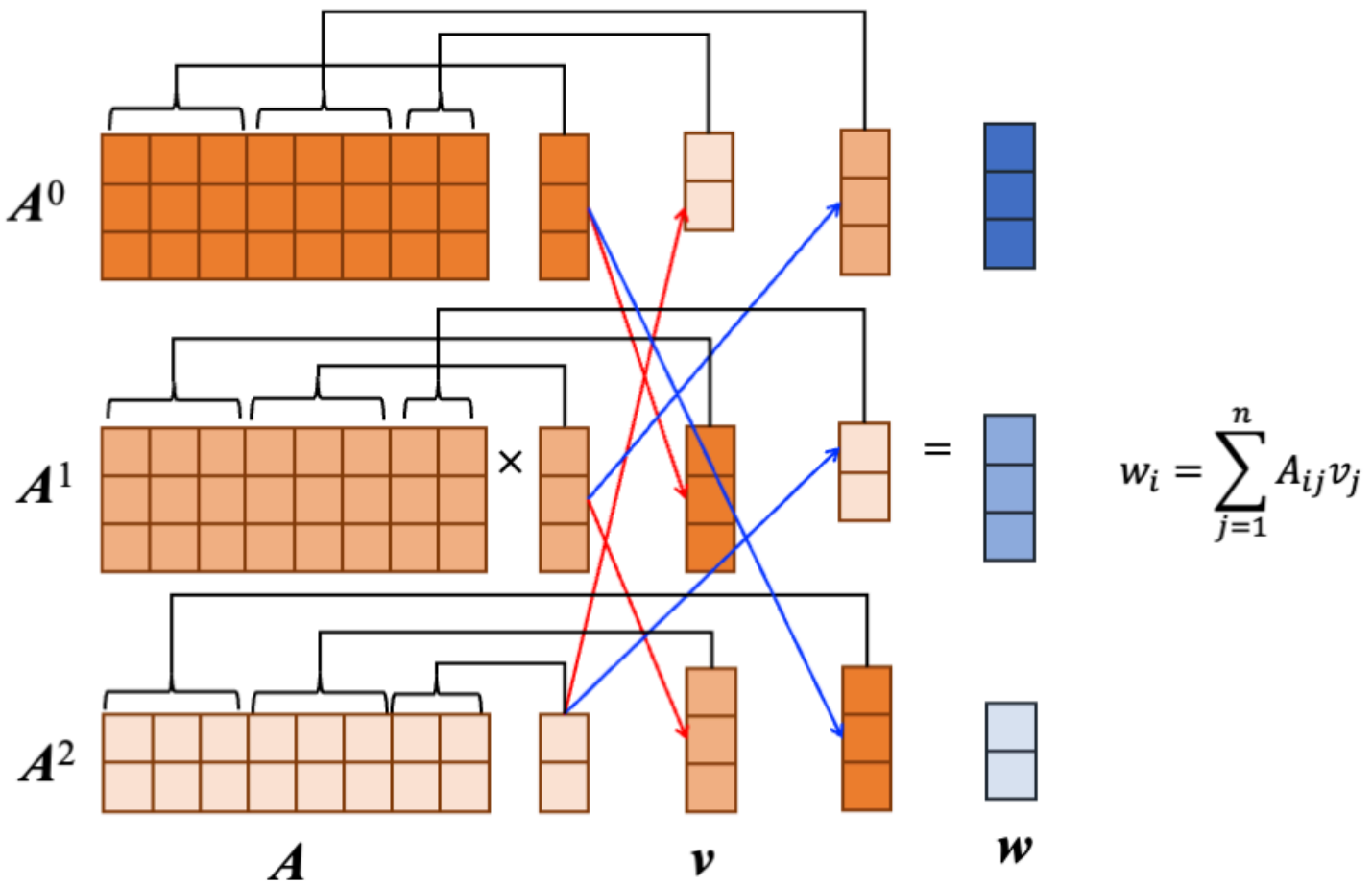
4. 행렬/벡터의 행 비등분할



- w_0계산을 위해 각 프로세스가 소유한 v들이 필요

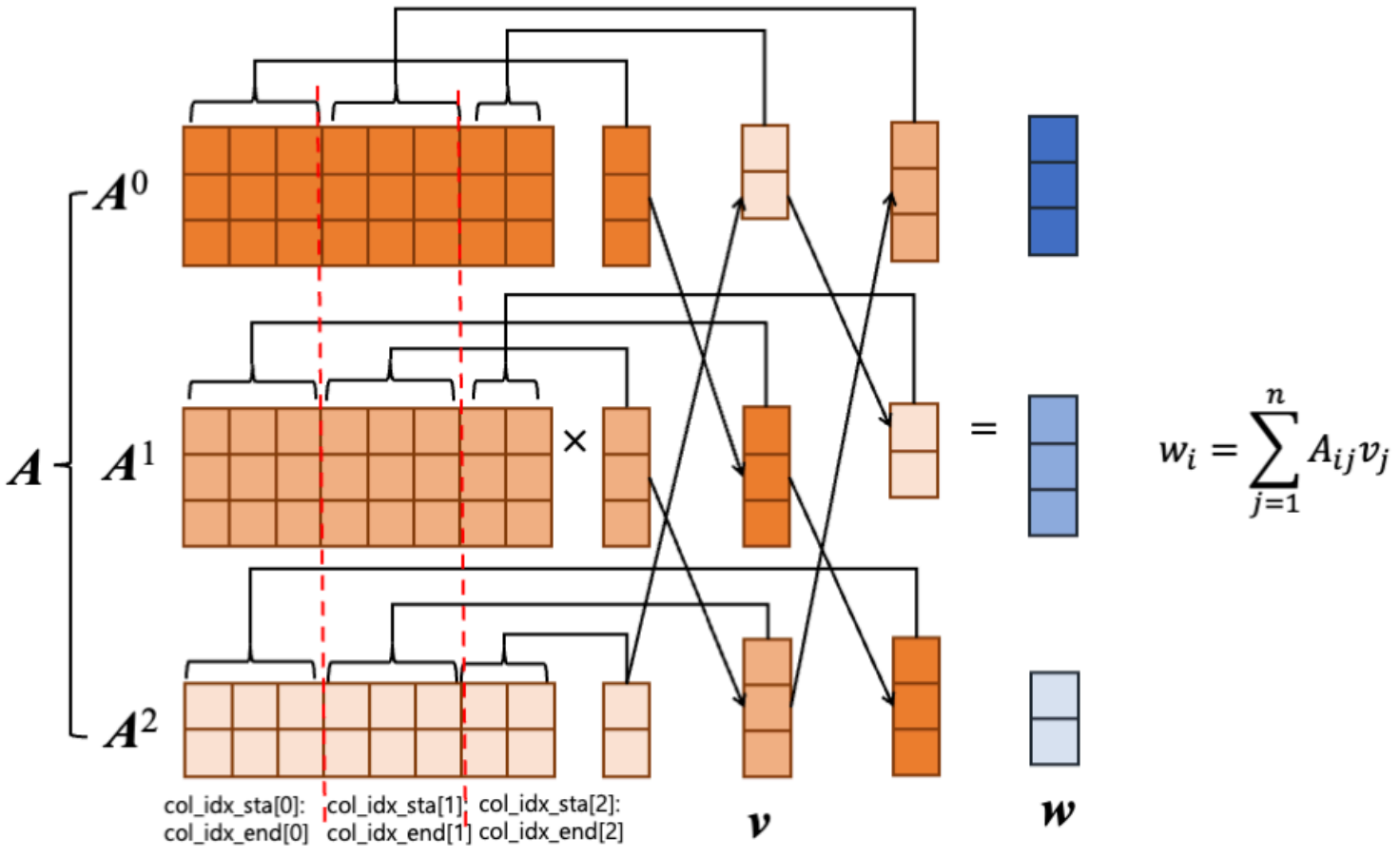


- 각 랭크가 가진 분할된 벡터를 다른 모든 랭크로 보내서 행렬-벡터 곱을 수행해야함



- 단점
  - 주고 받는 랭크 번호와 통신 대상이 매번 달라지게 됨
  - 프로세스수가 많을 경우 통신 거리가 멀어지며, 통신성능 저하

- 최종 형태
  - 순환 형태로 통신을 구현
  - 매번 곱하는 행렬의 열범위를 정확하게 설정해야 함



In [ ]:

```
%%writefile examples/Av_var.py

# Matrix A의 Row decomposition

from tools import para_range
import numpy as np
from mpi4py import MPI

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

##### 행렬 및 벡터 불러오기
##### 크기 n은 broadcast
if rank == 0 :
    A = np.load("examples/A.npy")
    v = np.load("examples/v.npy")
    n = v.size

else :
    A = None
    v = None
    n = 0

n = comm.bcast(n, root = 0)

##### 프로세스별 범위 할당
ista, iend = para_range(n, size, rank)

##### 프로세스별 행수 설정
n_local = (iend - ista + 1)

##### 분할된 행렬 선언
A_local = np.empty((n_local, n), dtype = np.float64)
v_local = np.empty(n_local, dtype = np.float64)

##### Scatterv를 위해 n_local를 이용한 리스트 생성. 이 때 부분행렬 행수가 아닌 전체 크기를 계산
##### 분할된 벡터의 크기는 모든 프로세스가 알고 있어야 하므로 allgather를 이용
n_local_chunks = comm.gather(n_local * n, root = 0) #FIX ME
n_local_cnts = comm.allgather(n_local) #FIX ME

##### 행렬과 벡터 분할
comm.Scatterv([A, n_local_chunks], A_local, root = 0) #FIX ME
comm.Scatterv([v, n_local_cnts], v_local, root = 0) #FIX ME

##### 분할된 벡터 곱 범위 지정
col_idx_sta = []
col_idx_end = []

##### i번째 랭크가 곱해질 열 위치의 시작점과 끝점을 리스트 형태로 저장. 모든 랭크들이 계산
for i in range(size) :
    col_idx_sta.append(sum(n_local_cnts[:i]))
    col_idx_end.append(sum(n_local_cnts[:i])+n_local_cnts[i])

##### Local MV (최초 자신의 벡터부분)
b = np.matmul(A_local[:,col_idx_sta[rank]:col_idx_end[rank]], v_local)
```

```
##### 송수신 프로세스 지정
inext = rank + 1 if rank < size - 1 else 0
iprev = rank - 1 if rank > 0 else size - 1

##### (전체 프로세스 크기 -1) 번만큼 통신을 수행하고 부분행렬-부분벡터 곱을 수행
for i in range(size - 1) :
##### 몇 번째 이전 랭크로부터 받았는지를 iloc을 이용하여 확인
    iloc = iprev - i if iprev >= i else iprev - i + size
    v_recv = np.empty(n_local_cnts[iloc], dtype = np.float64)
##### 다음 랭크로 부분 벡터를 전달하며, 이전 랭크로부터 부분 벡터를 받음
    comm.Sendrecv(v_local, inext, 1, v_recv, iprev, 1) #FIX ME
##### 받은 벡터는 복사하여, 행렬-벡터 곱에 사용하는 한편, 다음 랭크로 전달할 수 있게 함
    v_local = np.copy(v_recv)
    b += np.matmul(A_local[:,col_idx_sta[iloc]:col_idx_end[iloc]], v_local)

print(b, rank)
```

In [ ]:
```
! mpiexec -np 3 python examples/Av_var.py
```

In [ ]:
```
```

2. Gram-Schmidt

2.1. 개요

- 직교화(orthogonalization)는 벡터공간에서 서로 직교하는 정규 기저(orthonormal basis)를 찾는 과정으로서 수치적 선형대수가 활용되는 대부분의 분야에서 빈번하게 사용됨.

- Gram-Schmidt 과정은 직교화의 방법 중 하나로 다음과 같은 알고리즘으로 진행.

Input : 선형독립적 N개의 벡터 $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_N\}$

For $i = 1..N$

$\boldsymbol{u}_i \leftarrow \boldsymbol{v}_i - \sum_{j=1}^{i-1} \frac{\langle \boldsymbol{u}_j, \boldsymbol{v}_i \rangle}{\langle \boldsymbol{u}_j, \boldsymbol{u}_j \rangle} \boldsymbol{u}_j$

$\boldsymbol{e}_i \leftarrow \frac{\boldsymbol{u}_i}{\|\boldsymbol{u}_i\|}$

End for

Output : 서로 직교하는 N개의 단위벡터 $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \dots, \boldsymbol{e}_N\}$

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \begin{bmatrix} u_1 & u_2 & \cdots & u_M \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_M \end{bmatrix} = \sum_{k=1}^{M} u_k v_k$$

$$\|\boldsymbol{u}\| = \sqrt{\langle \boldsymbol{u}, \boldsymbol{u} \rangle}$$

- 이를 전개하여 표현하면 다음곽 같음.

$$\mathbf{u_1} = \mathbf{v_1}$$

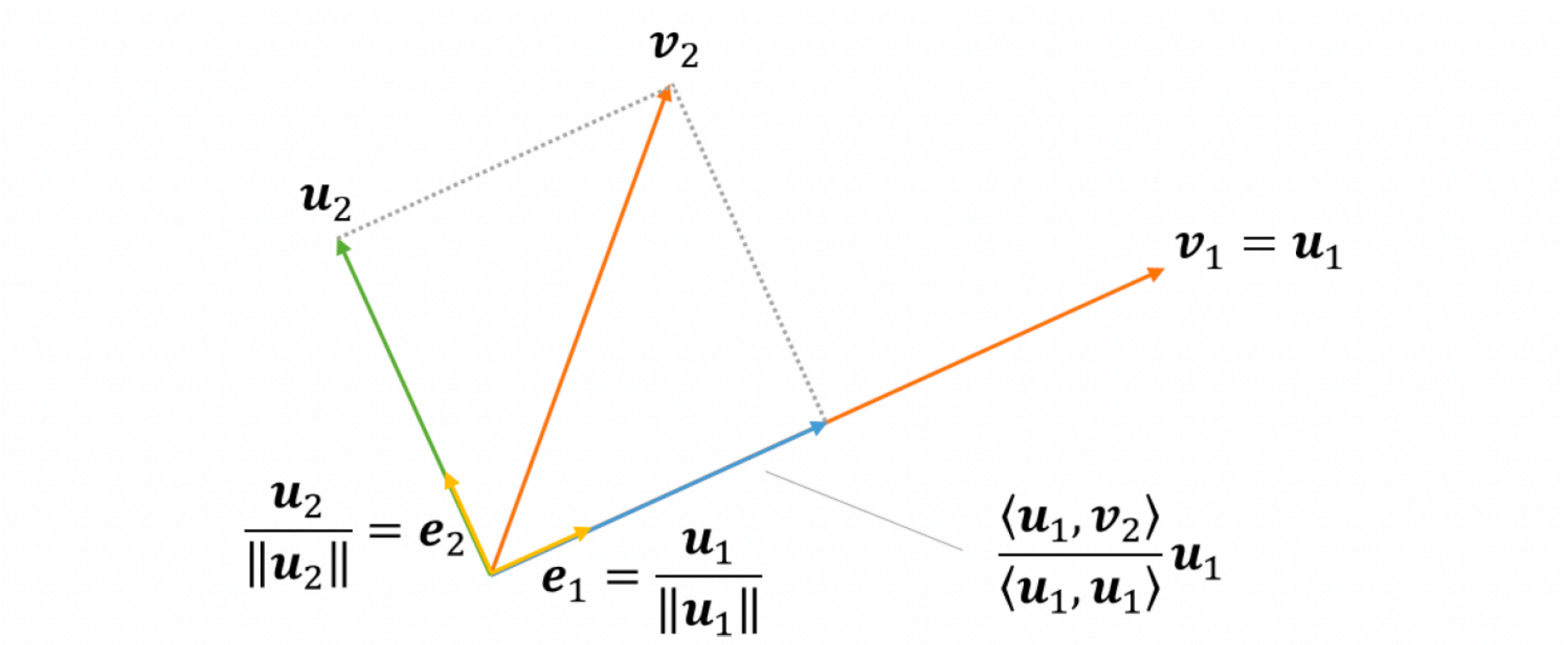
$$\mathbf{u_2} = \mathbf{v_2} - \frac{\langle \mathbf{v_2}, \mathbf{u_1} \rangle}{\|\mathbf{u_1}\|^2} \mathbf{u_1}$$

$$\mathbf{u_3} = \mathbf{v_3} - \frac{\langle \mathbf{v_3}, \mathbf{u_2} \rangle}{\|\mathbf{u_2}\|^2} \mathbf{u_2} - \frac{\langle \mathbf{v_3}, \mathbf{u_1} \rangle}{\|\mathbf{u_1}\|^2} \mathbf{u_1}$$

$$\vdots$$

$$\mathbf{u_n} = \mathbf{v_n} - \frac{\langle \mathbf{v_n}, \mathbf{u_{n-1}} \rangle}{\|\mathbf{u_{n-1}}\|^2} \mathbf{u_{n-1}} - \frac{\langle \mathbf{v_n}, \mathbf{u_{n-2}} \rangle}{\|\mathbf{u_{n-2}}\|^2} \mathbf{u_{n-2}} - \cdots - \frac{\langle \mathbf{v_n}, \mathbf{u_1} \rangle}{\|\mathbf{u_1}\|^2} \mathbf{u_1}$$

- 2차원 벡터에 대한 Gram-Schmidt 과정



2.2. QR 분해

- 다음과 같이 $\mathbf{A}$를 분해

$$\mathbf{A} = \mathbf{QR}$$

- 여기서 $\mathbf{Q}$의 열벡터는 $\mathbf{A}$의 열벡터가 span하는 공간 S의 직교 기저 벡터이며, $\mathbf{R}$는 Upper triangular matrix임

- $\mathbf{A}$ 의 열벡터 $\mathbf{a_i}$ 들은 Gram-Schmidt 과정을 거쳐 얻어진 정규 직교 벡터 $(\mathbf{u_1}, \mathbf{u_2}, \cdots, \mathbf{u_n})$ 에 대해 다음을 만족

$$\mathbf{a_i} = \langle \mathbf{a_i}, \mathbf{u_1} \rangle \mathbf{u_1} + \langle \mathbf{a_i}, \mathbf{u_2} \rangle \mathbf{u_2} + \cdots + \langle \mathbf{a_i}, \mathbf{u_n} \rangle \mathbf{u_n}$$

In [16]:
```
import numpy as np

# 10개의 20차원 열벡터
N = 10
D = 20
v = np.random.rand(D, N)

np.save("examples/v", v)
```

In [ ]:
```
# QR 분해 : Q의 열벡터가 정규기저벡터
import numpy as np
np.set_printoptions(linewidth=np.inf)

v = np.load("examples/v.npy")
print("v :")
print(v)

Q, R = np.linalg.qr(v)

print("Q :")
print(Q)

print("R :")
print(R)

print("Norm Q[0] :")
print(np.linalg.norm(Q[0]))
```

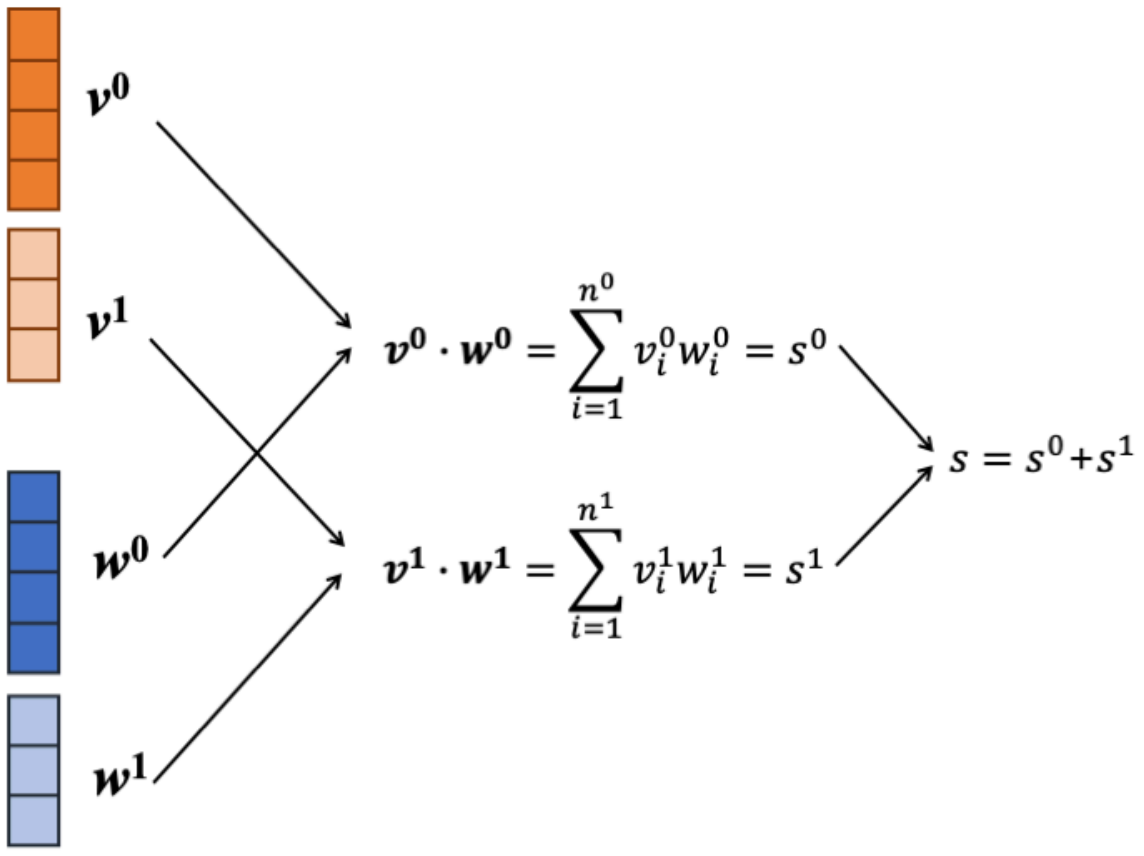
1. 순차코드

- 열벡터를 행벡터로 전환
- 메모리 접근이 효율적임

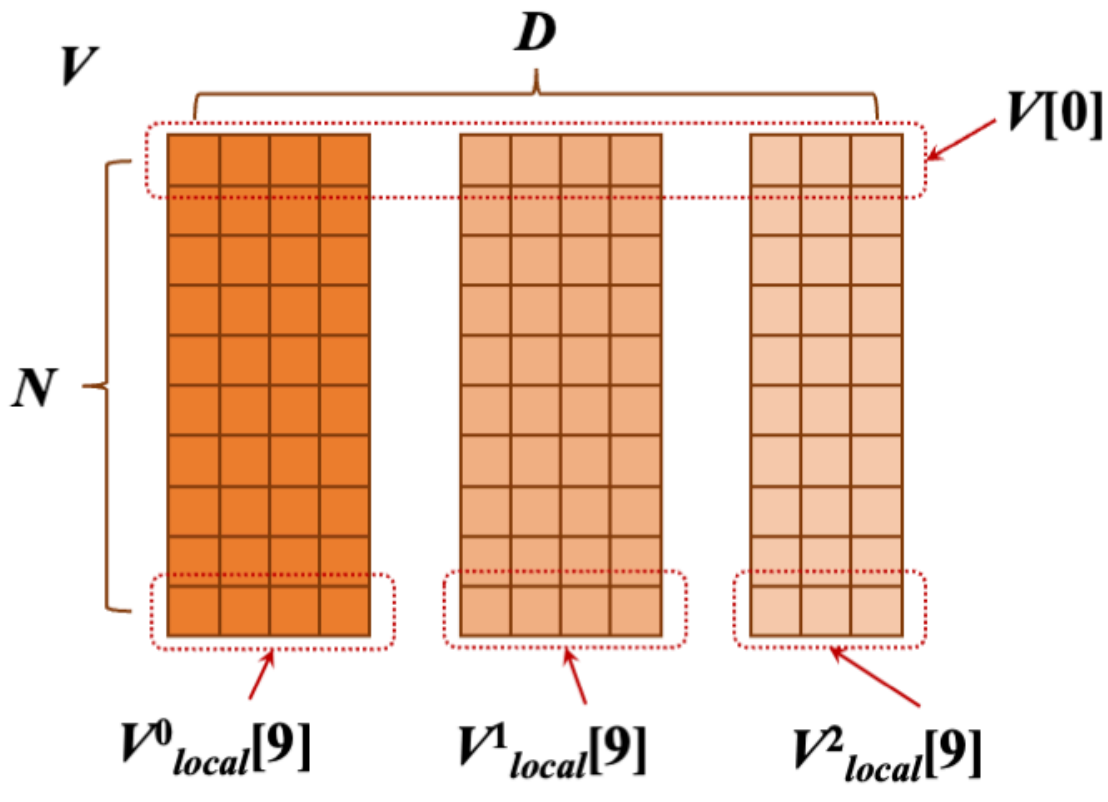
In [ ]:
```
import numpy as np

V = np.transpose(v).copy()
for j in range (N) :
    for i in range (j) :
        coef = -np.dot(V[i], V[j])
        V[j] = V[j] + coef * V[i]
    coef = np.linalg.norm(V[j])
    V[j] = V[j] / coef

np.set_printoptions(linewidth=np.inf)
print("V transpose : ")
print(V)

print("Norm ( V_T [0] ) : ")
print(np.linalg.norm(np.transpose(V)[0]))
```

2. 병렬코드

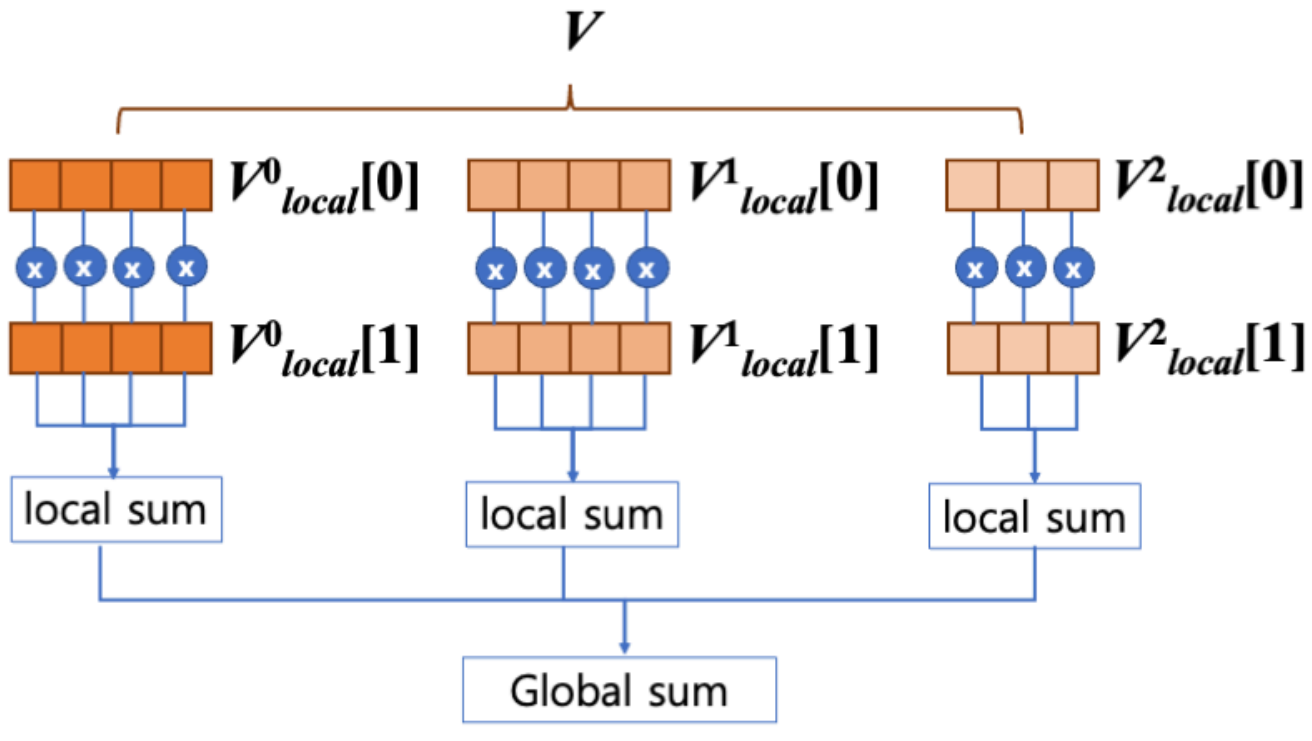
- 의존성 분석
  - u_i들은 순차적으로 계산됨 : u_i를 위해 $u_{i-1}, \cdots, u_1$이 필요
  - u_i, v_i의 각 i 성분 계산은 독립적임 : 이를 나누어 계산하며 벡터 분할과 동일



- 행 대신 열방향으로 분해 : 각각의 벡터를 chunk단위로 분해



- 벡터의 내적을 병렬로 처리



In [ ]:
```python
%%writefile examples/gs.py
import numpy as np
from mpi4py import MPI
from tools import para_range

np.set_printoptions(linewidth=np.inf)

N = 10
D = 20

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0 :
    v = np.load("examples/v.npy")
    V = np.transpose(v).copy()
else :
    V = np.empty((N, D), dtype = np.float64)

##### 벡터를 분할할 크기를 설정
ista, iend = para_range(D, size, rank)
D_local = iend - ista + 1

##### 벡터 분할을 위해 Scatterv에 필요한 cnts와 disp 계산
D_local_cnts = comm.allgather(D_local) #FIX ME

##### 부분벡터를 N개만큼 생성
V_local = np.empty([N, D_local], dtype = np.float64)

# V의 분할. 모든 V[i]에 대한 분할 필요
for i in range (N) :
    comm.Scatterv((V[i], D_local_cnts), V_local[i], root=0) #FIX ME

# 분할된 벡터에 대한 계산후 내적 값을 reduction
for j in range (N) :
    for i in range (j) :
        coef = -np.dot(V_local[i], V_local[j]) #FIX ME
        coef_all = comm.allreduce(coef, MPI.SUM)#FIX ME
        V_local[j] = V_local[j] + coef_all * V_local[i]
    coef = np.dot(V_local[j], V_local[j])
    coef_all = comm.allreduce(coef, MPI.SUM) #FIX ME
    V_local[j] = V_local[j] / np.sqrt(coef_all)

# print(V_local)

for i in range (N) :
    comm.Gatherv( V_local[i], (V[i], D_local_cnts), root = 0)

if rank == 0 :
    print(V)
```

In [ ]:
```
! mpiexec -np 4 python examples/gs.py
```

- 행렬의 전치를 이용하여 열분해를 행분해로 전환
- 계산과 통신이 필요한 시점에서 전치를 이용

In [ ]:
```
%%writefile examples/gs_tr.py

# Transpose를 이용하여 행렬을 손쉽게 열분해 하는 방법
import numpy as np
from mpi4py import MPI
from tools import para_range

np.set_printoptions(linewidth=np.inf)

N = 10
D = 20

comm = MPI.COMM_WORLD

size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0 :
    v = np.load("examples/v.npy")
else :
    v = np.empty((D, N), dtype = np.float64)

# 읽어온 행렬을 transpose시키지 않고 열벡터를 그대로 이용. 따라서 행분해를 할 수 있음
ista, iend = para_range(D, size, rank)

D_local = iend - ista + 1

# Scatterv에 필요한 cnts와 disp 계산
D_local_chunks = comm.allgather(D_local * N) #FIX ME

v_local = np.empty([D_local, N], dtype = np.float64)

comm.Scatterv( (v, D_local_chunks), v_local, root = 0 )

# Transpose를 이용하여 행분해를 열분해로 전환
V_local = np.transpose(v_local).copy()

# 분할 계산후 reduction
for j in range (N) :
    for i in range (j) :
        coef = -np.dot( V_local[i], V_local[j] ) #FIX ME
        coef_all = comm.allreduce(coef)
        V_local[j] = V_local[j] + coef_all * V_local[i]
    coef = np.dot(V_local[j], V_local[j])
    coef_all = comm.allreduce(coef)
    V_local[j] = V_local[j] / np.sqrt(coef_all)

v_local = np.transpose(V_local).copy()

comm.Gatherv( v_local, (v, D_local_chunks), root = 0 )

if rank == 0 :
    print(v)
```

In [ ]:
```
! mpiexec -np 6 python examples/gs.py
```

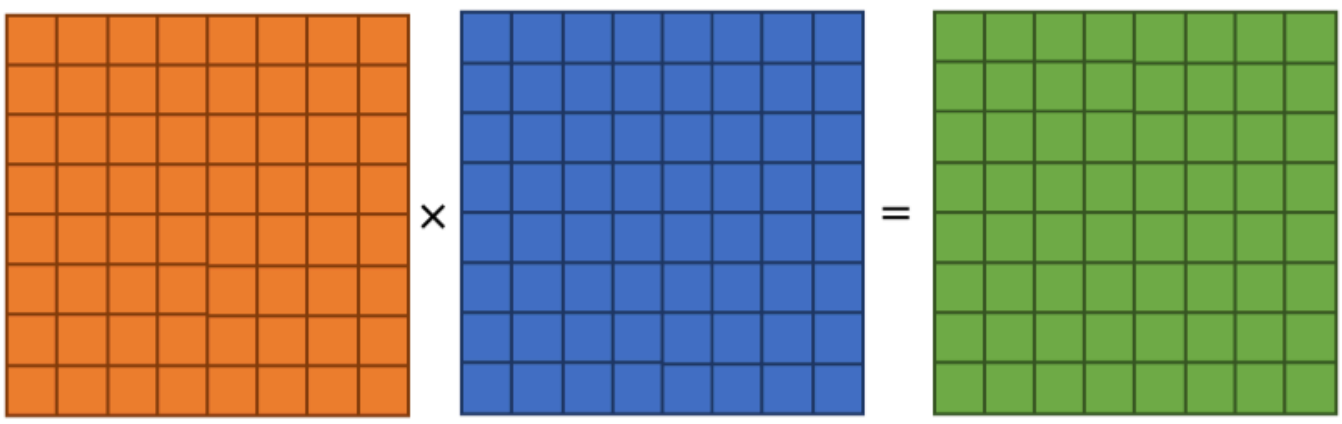
In [ ]:

In [ ]:

3. 벡터와 행렬 연산 (II)

3.1. 행렬-행렬 곱

1. 순차코드

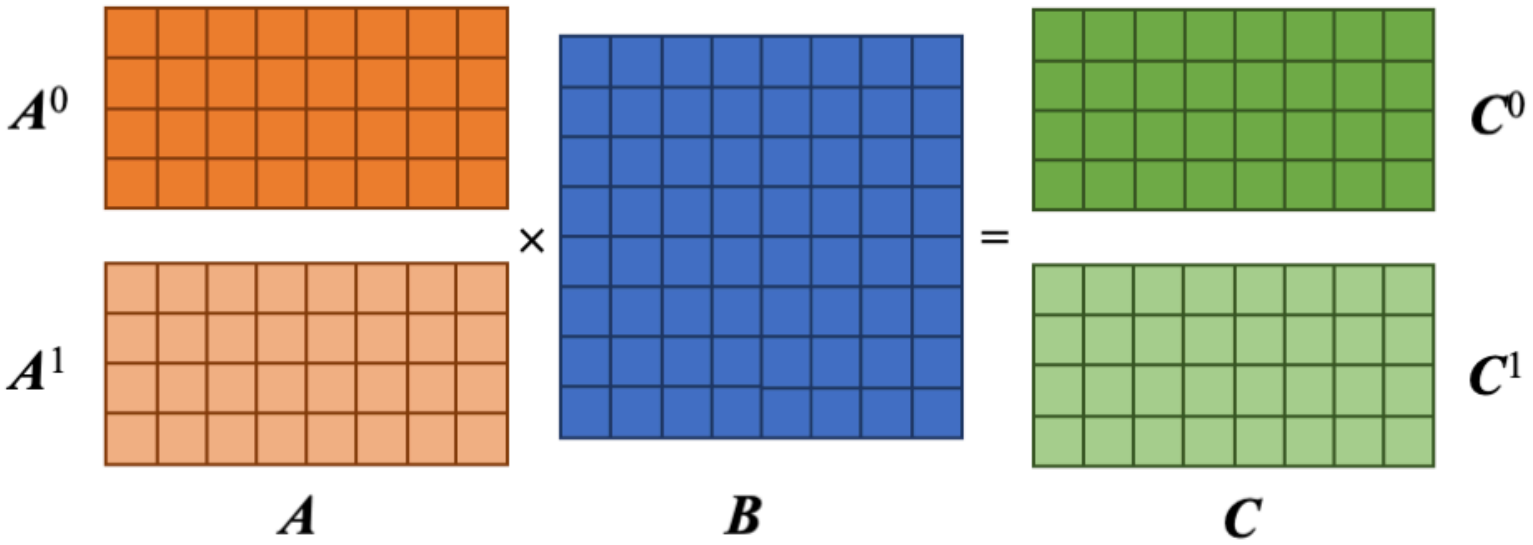


$A \quad B \quad C$

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

In [ ]:
```python
import numpy as np

A = np.load("examples/A.npy")
B = np.load("examples/B.npy")

C = np.matmul(A, B)
print (C)
```

2. 행렬A의 행분할



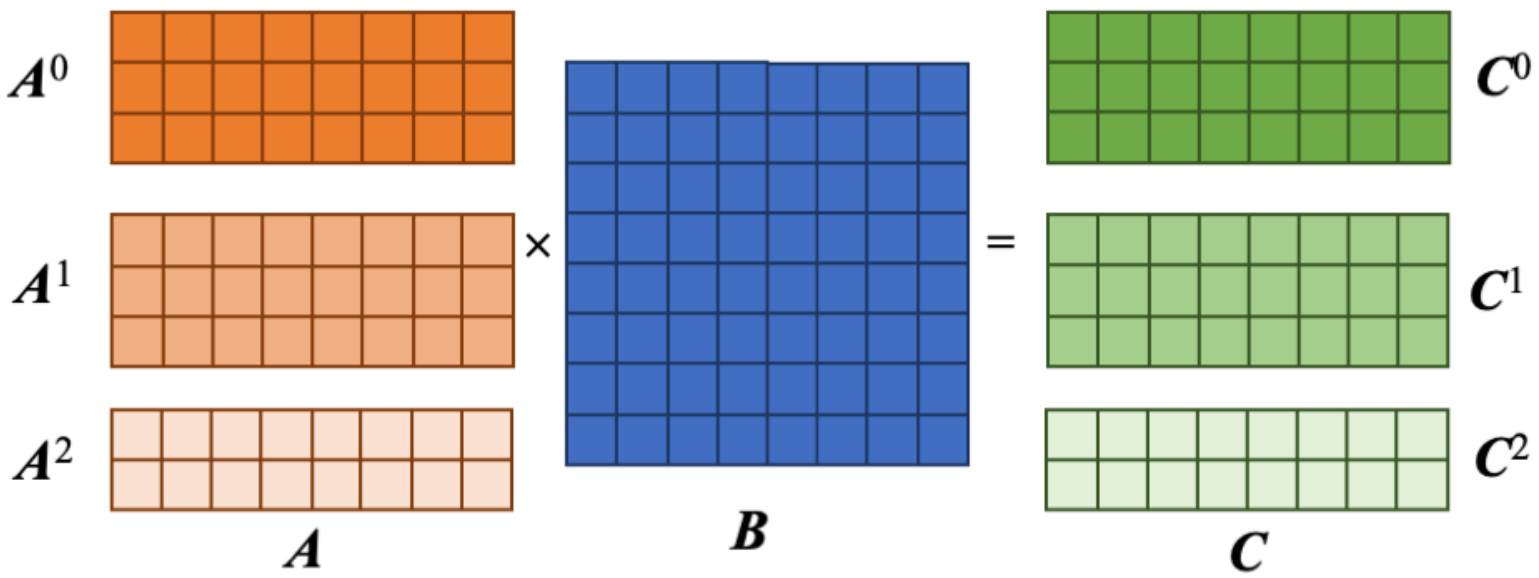
$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

In [ ]:
```python
%%writefile examples/AB.py

import numpy as np
from mpi4py import MPI
```

```
comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

if rank == 0 :
    A = np.load("examples/A.npy")
    B = np.load("examples/B.npy")
    n = A[0].size
    n = comm.bcast(n, root = 0)

else :
    n = 0
    n = comm.bcast(n, root = 0)
    A = None
    B = np.empty((n, n), dtype = np.float64)

n_local = int(n / size)

A_local = np.empty((n_local, n), dtype = np.float64)

##### 행렬 A의 분할
comm.Scatter(A, A_local, root = 0) #FIX ME
comm.Bcast(B, root = 0)

C = np.matmul(A_local,B)

print(C, rank)
```

In [ ]:
```
! mpiexec -np 2 python examples/AB.py
```

3. 행렬A의 행분할 (비등분할)



$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

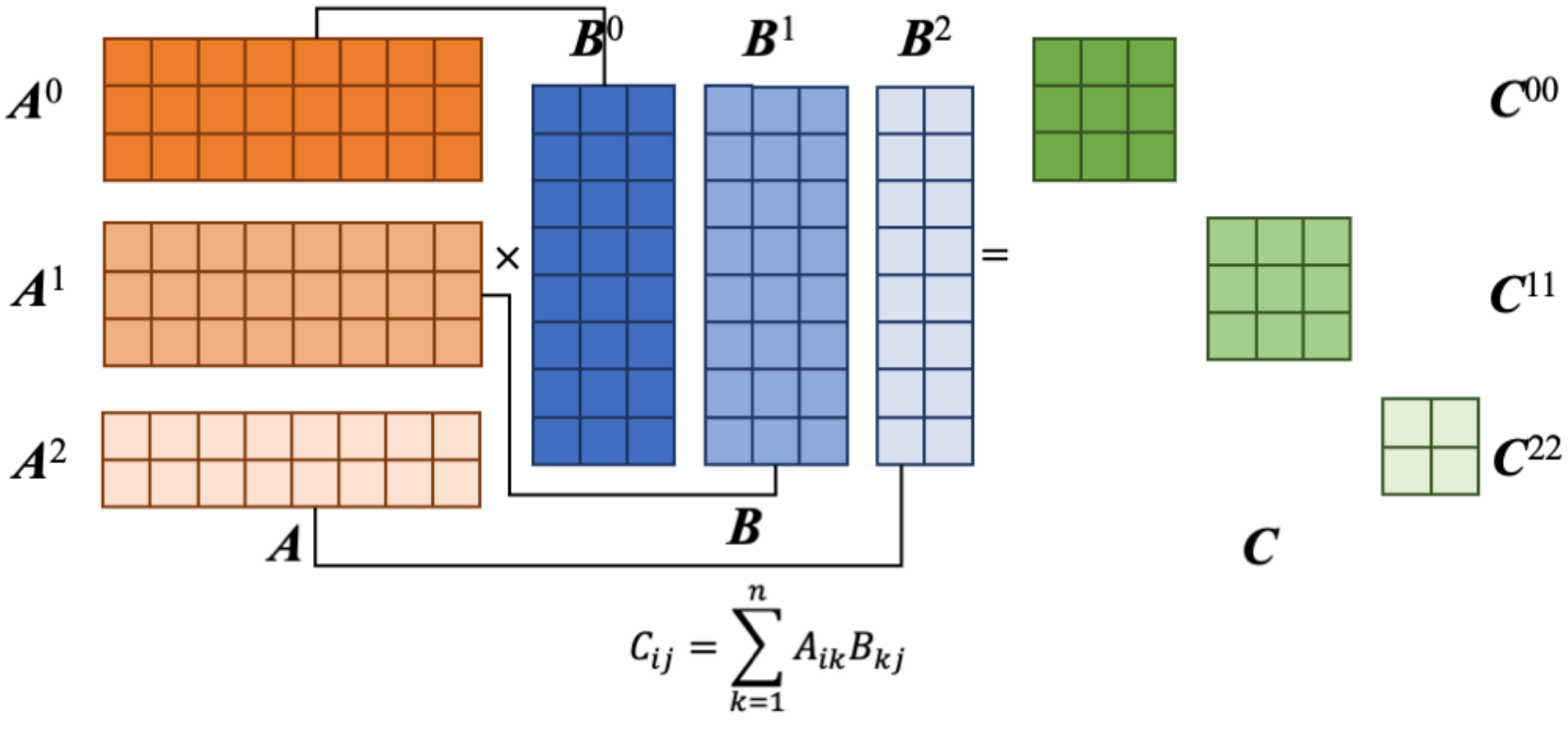
In [ ]:
```
%%writefile examples/AvarB.py

# Matrix A의 Row decomposition

import numpy as np
from mpi4py import MPI
from tools import para_range

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

##### 행렬 및 벡터 불러오기
##### 크기 n은 broadcast
if rank == 0 :
    A = np.load("examples/A.npy")
    B = np.load("examples/B.npy")
    n = A[0].size
    n = comm.bcast(n, root = 0)
else :
    n = 0
    n = comm.bcast(n, root = 0)
```

```
    A = None
    B = np.empty((n, n), dtype = np.float64)

##### 행렬 A의 분할 범위를 설정
ista, iend = para_range(n, size, rank) #FIX ME
n_local = (iend - ista + 1) #FIX ME

##### Scatterv를 위해 n_local를 이용한 리스트 생성. 이 때 부분행렬 행수가 아닌 전체 크기를 계산
A_local = np.empty((n_local, n), dtype = np.float64)
n_local_chunks = comm.gather(n_local * n, root = 0)

##### 행렬 A의 분할
comm.Scatterv((A, n_local_chunks), A_local) #FIX ME
comm.Bcast(B, root = 0)

##### 분할된 부분행렬과 행렬B의 곱
C_local = np.matmul(A_local,B)

print(C_local, rank)
```

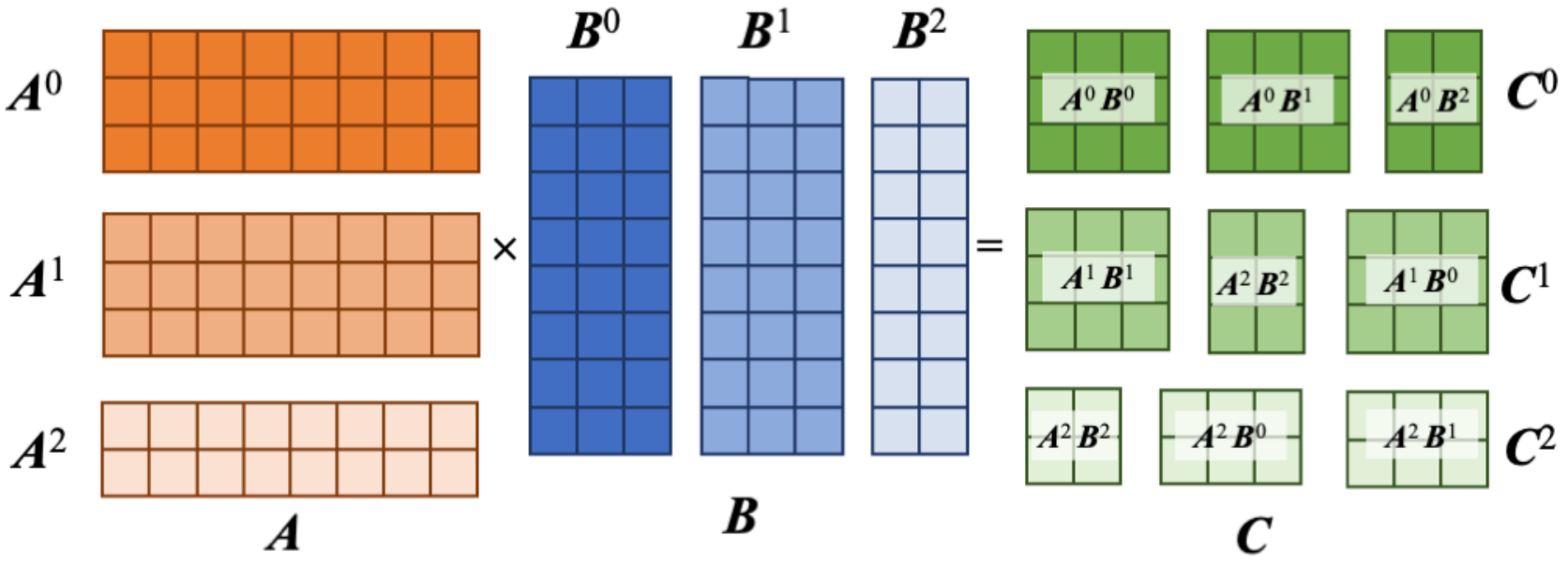
In [ ]:
```
! mpiexec -np 3 python examples/AvarB.py
```

4. 행렬A의 행분할, 행렬 B의 열분할

- 각 프로세스가 가진 부분행렬만 곱할 경우



- 통신을 이용하여 B_i에 대한 부분행렬을 수신하여 부분행렬 곱을 실행



$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

In [ ]:

```
%%writefile examples/ABvar.py

import numpy as np
from mpi4py import MPI
from tools import para_range

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

##### 행렬 및 벡터 불러오기
##### 크기 n은 broadcast
if rank == 0 :
    A = np.load("examples/A.npy")
    B = np.load("examples/B.npy")
    BT = np.transpose(B).copy()
    n = A[0].size
    n = comm.bcast(n, root = 0)

else :
    n = 0
    n = comm.bcast(n, root = 0)
    A = None
    BT = None

##### 행렬 A와 B의 분할 범위를 설정
ista, iend = para_range(n, size, rank)
n_local = (iend - ista + 1)

##### 부분행렬의 선언. 행렬 B의 부분행렬은 전치행렬을 이용
A_local = np.empty((n_local, n), dtype = np.float64)
BT_local = np.empty((n_local, n), dtype = np.float64)

##### Scatterv를 위해 n_local를 이용한 리스트 생성. 이 때 부분행렬 행수가 아닌 전체 크기를 계산
n_local_chunks = comm.allgather(n_local * n)

##### 행렬 A, B의 분할
comm.Scatterv([A, n_local_chunks], A_local, root = 0) #FIX ME
comm.Scatterv([BT, n_local_chunks], BT_local, root = 0) #FIX ME

##### 행렬 B의 부분행렬을 전치하여 원래 형태로 만듦
B_col = np.transpose(BT_local)

##### 다음 및 이전 랭크 설정
inext = rank + 1 if rank < size - 1 else 0
iprev = rank - 1 if rank > 0 else size - 1

##### 계산결과는 정렬되지 않으므로 unordered로 취급
C_unordered_local = np.matmul(A_local, B_col)


##### (전체 프로세스 크기 -1) 번만큼 통신을 수행하고 부분행렬-부분벡터 곱을 수행
for i in range(size - 1) :
##### 몇 번째 이전 랭크로부터 받았는지를 iloc을 이용하여 확인
    iloc = iprev - i if iprev >= i else iprev - i + size
    B_recv = np.empty(n_local_chunks[iloc], dtype = np.float64)
```

```
    ##### 분할된 행렬 B를 송수신하고 A의 분할된 부분과 곱하여 C에 저장
    comm.Sendrecv(B_col, inext, 1, B_recv, iprev, 1) #FIX ME
    B_col = np.copy(B_recv)
    B_col = np.reshape(B_col, (n, int(n_local_chunks[iloc]/n))) #FIX ME
    C_block = np.matmul(A_local, B_col) #FIX ME
##### 현재 정렬되지 않은 순서로 C_unordered_local_chunks에 저장됨
    C_unordered_local = np.append(C_unordered_local, C_block, axis = 1) #FIX ME

print(C_unordered_local, rank)
```

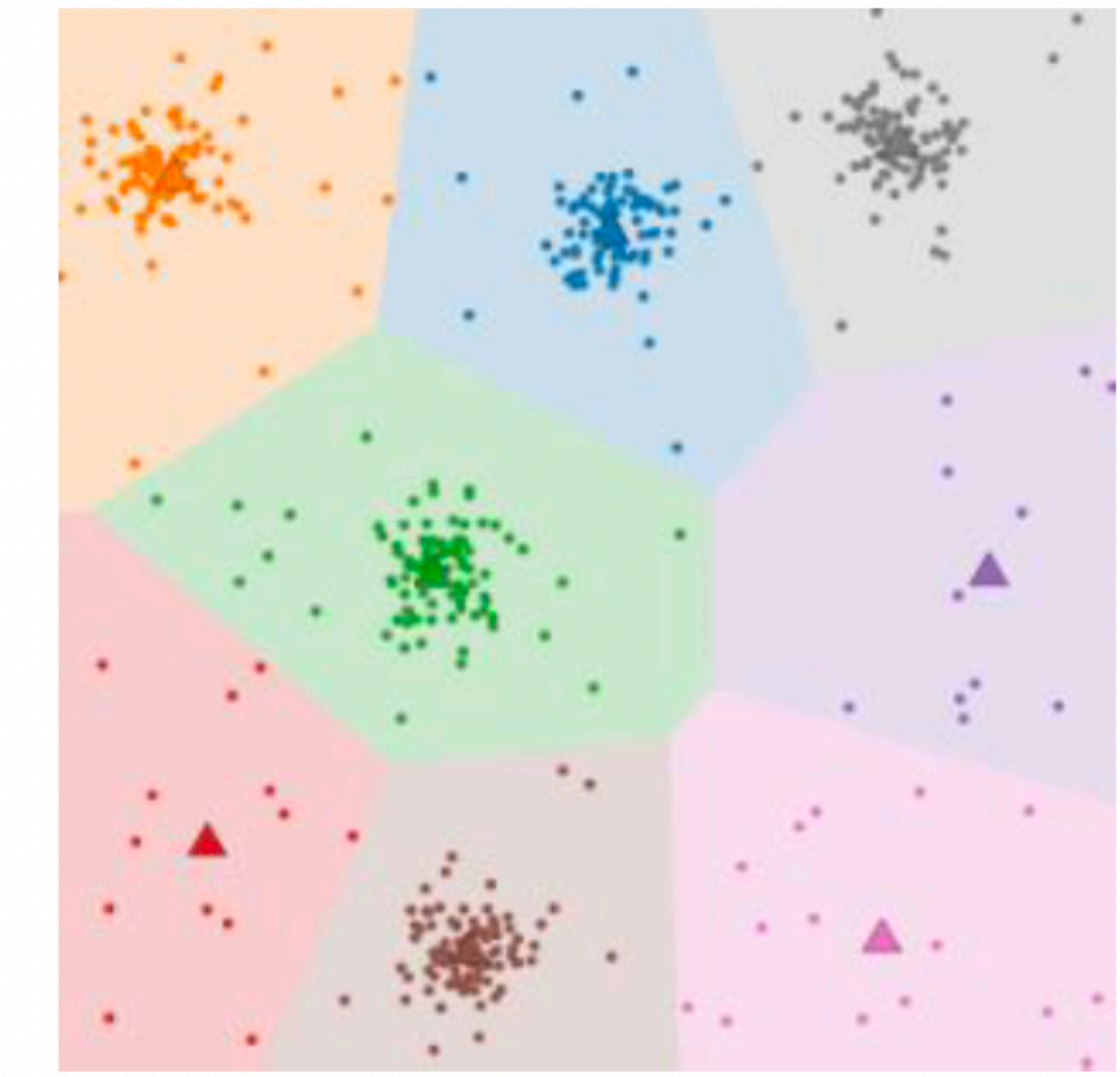
In [ ]:
```
! mpiexec -np 3 python examples/ABvar.py
```

5. 직접 해보기
   - 올바른 크기의 C를 미리 선언하고 적절한 위치에 C_block 을 배치
   - 3의 C_rows와 같은 결과를 얻음

4. K-means

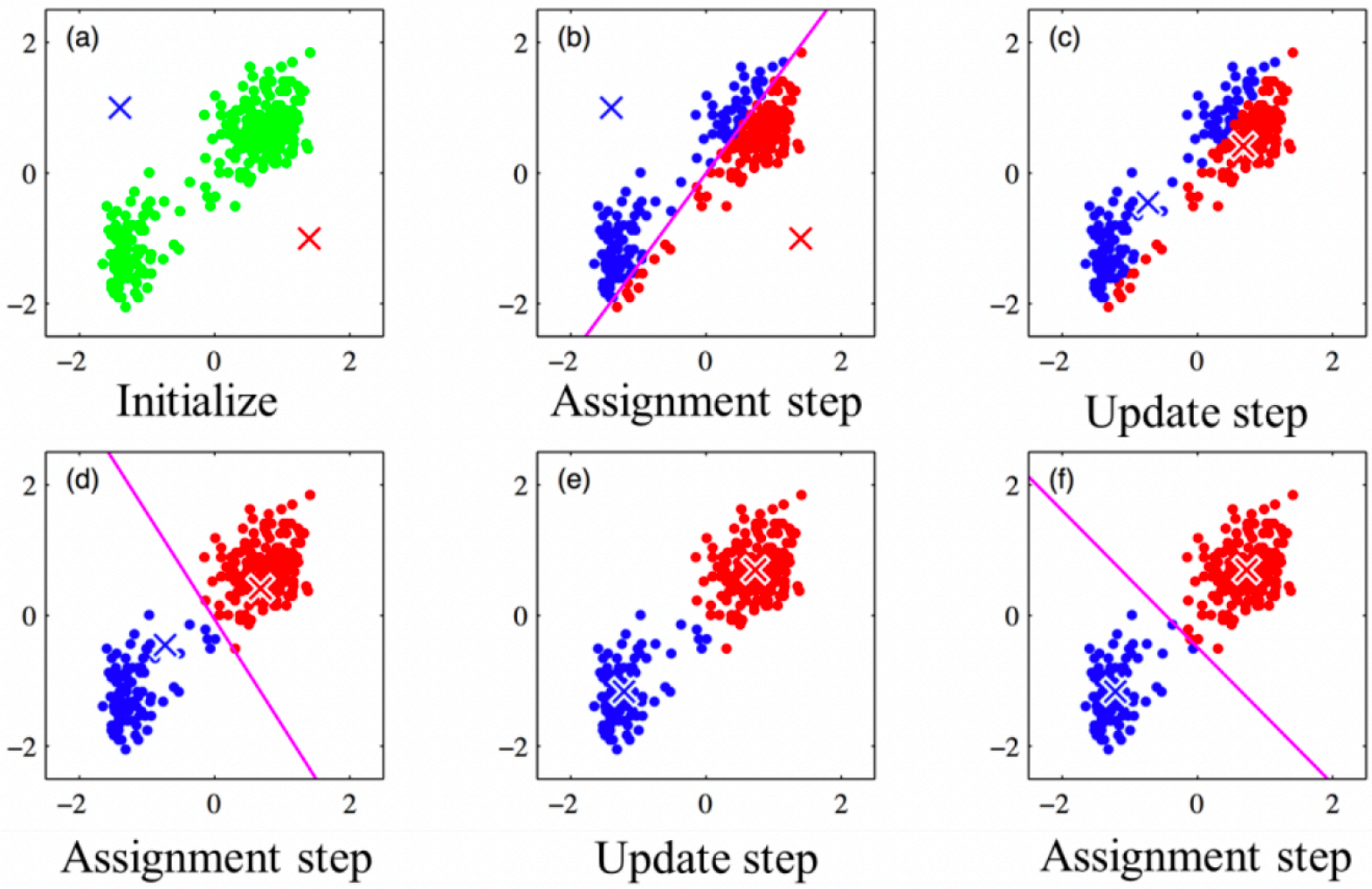
4.1. 개요

- K-평균 알고리즘(K-means algorithm)은 주어진 데이터를 k개의 클러스터로 묶는 방법으로 각 클러스터와의 거리 차이 분산을 최소화
- 본 알고리즘은 비지도학습(Unsupervised Learning)의 일종으로, 레이블이 없는 입력 데이터에 레이블을 달아주는 역할을 수행
- K-평균 알고리즘은 공간데이터 분석(물류센터 위치선정), 웹문서 분류 등 다양한 분야 에서 이용
- (위키피디아)



4.2. K-means 알고리즘

- Assignment step 과 update step 으로 구성
- Assignment step 에서는 점(point)으로부터 각 클러스터의 중심점까지의 직선 거리를 계산하고, 그 점에서 가장 가까운 클러스터를 찾아 점을 배당
- Update step 에서는 각 클러스터에 있는 점들의 무게중심 값으로 해당 클러스터의 중심위치를 재설정.
- 클러스터의 중심위치가 변하지 않을 때까지 반복



1. Scipy 함수를 이용하여 데이터 생성

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
```

In [ ]:
```
X,y = make_blobs(n_samples = 500,n_features = 2,centers = 2,random_state = 23)
fig = plt.figure(0)
plt.grid(True)
plt.scatter(X[:,0], X[:,1])
plt.show()
```

In [ ]:
```
X,y = make_blobs(n_samples = 500,n_features = 2,centers = 10,random_state = 23)

np.save("examples/X", X)
np.save("examples/y", y)

fig = plt.figure(0)
plt.grid(True)
plt.scatter(X[:,0], X[:,1])
plt.show()
```

2. 순차코드

- 7개로 생성된 클러스터를 분류
- 초기 무게중심 값은 (-10, 10) 사에에서,랜덤하게 생성

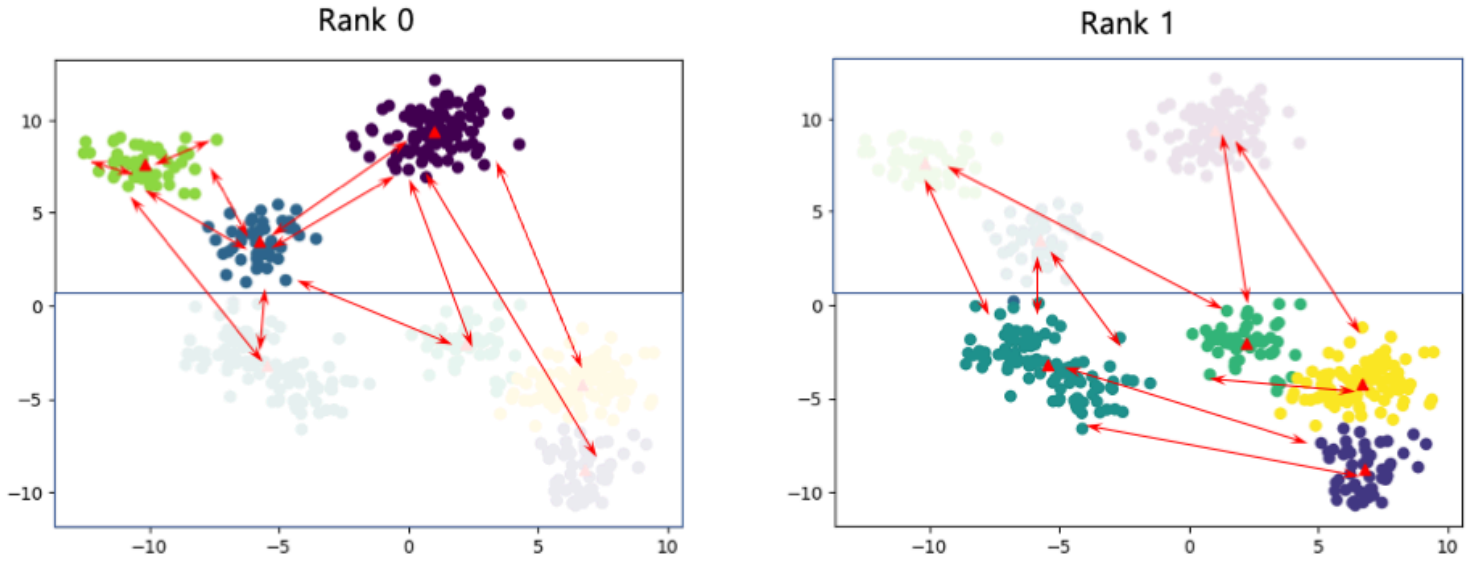
In [ ]:
```
k = 7

clusters = {}
np.random.seed(23)

##### 딕셔너리 자료형을 이용하여 k개의 클러스터의 중심 좌표와 point 좌표 저장할 수 있는 객체 생성
for idx in range(k):
    center = 10*(2*np.random.random((X.shape[1],))-1)
    points = []
    cluster = {
        'center' : center,
        'points' : []
    }

    clusters[idx] = cluster

clusters
```

In [ ]:
```
def distance(p1,p2):
    return np.sqrt(np.sum((p1-p2)**2))

#Implementing E step
def assign_clusters(X, clusters):
    # 각 데이터 포인트에 대해 반복
```

```
    for idx in range(X.shape[0]):
        dist = []

        curr_x = X[idx]

        # k번째 클러스터 센터와 각 데이터 포인트의 거리 계산
        for i in range(k):
            dis = distance(curr_x,clusters[i]['center'])
            dist.append(dis)

        # 계산된 클러스터와의 거리중에 최소값으로 클러스터를 배정
        curr_cluster = np.argmin(dist)

        # 현재 데이터 포인트의 좌표를 속한 클러스터에 추가
        clusters[curr_cluster]['points'].append(curr_x)
    return clusters

#Implementing the M-Step
def update_clusters(X, clusters):
    # 클러스터에 대해 반복
    for i in range(k):

        # 현재 클러스터에 포함된 데이터 포인트의 좌표를 불러옴
        points = np.array(clusters[i]['points'])

        # 데이터 포인트들의 평균을 취해 현재 클러스터 중심을 업데이트
        if points.shape[0] > 0:
            new_center = points.mean(axis =0)
            clusters[i]['center'] = new_center

            clusters[i]['points'] = []
    return clusters

# 데이터 포인트 X에 대해 가장 거리가 가까운 클러스터를 예측
def pred_cluster(X, clusters):
    pred = []
    for i in range(X.shape[0]):
        dist = []
        for j in range(k):
            dist.append(distance(X[i],clusters[j]['center']))
        pred.append(np.argmin(dist))
    return pred

# Repeat E-M steps
for i in range (10) :
    clusters = assign_clusters(X,clusters)
    clusters = update_clusters(X,clusters)
    print(clusters)

# Data point clustering
pred = pred_cluster(X,clusters)

plt.scatter(X[:,0],X[:,1],c = pred)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.show()
```

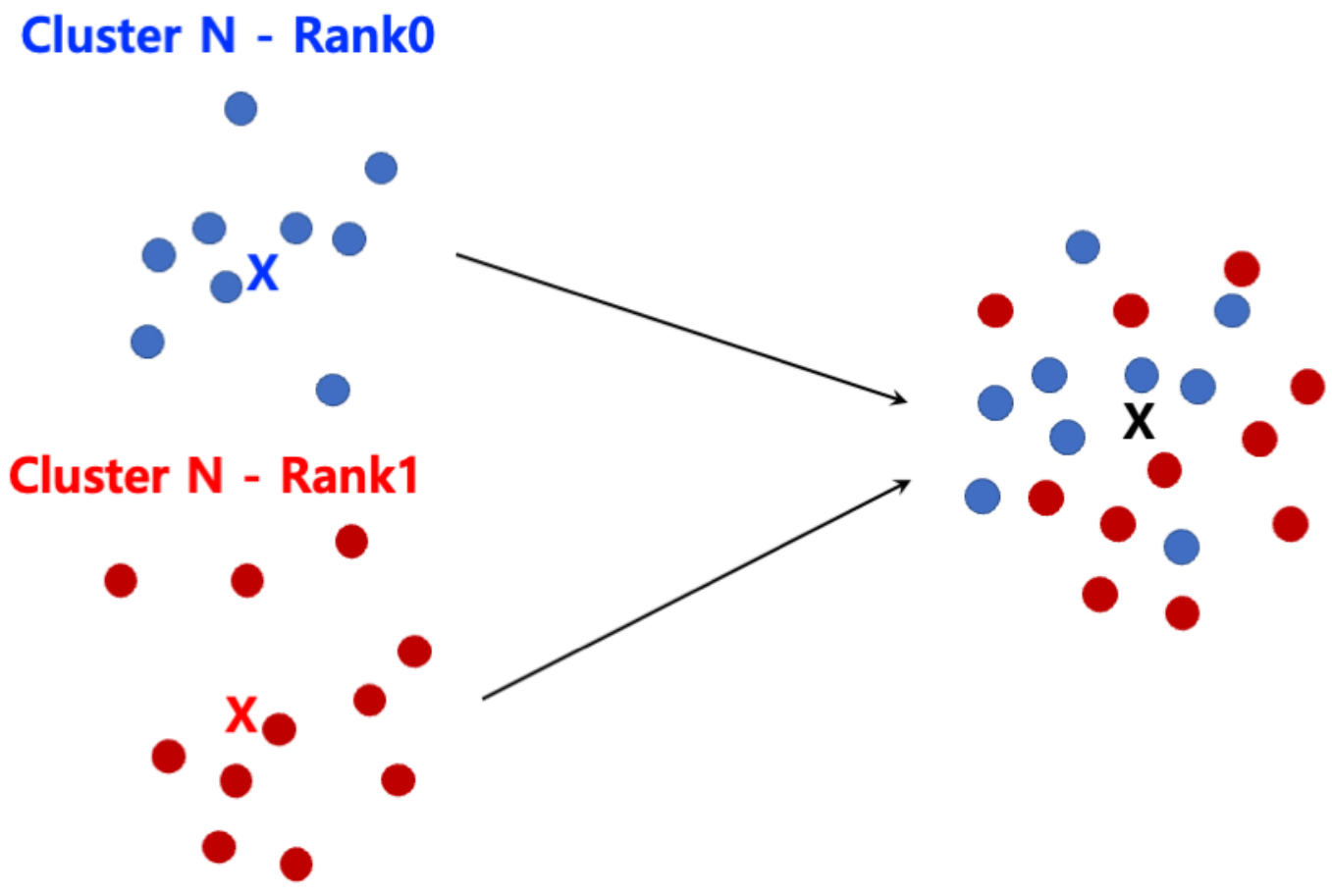
3. 병렬코드

- 의존성 분석
  - 각 점의 거리계산은 독립적임
  - 거리계산으로부터 무게중심 계산시 reduction 연산이 필요
  - 출력을 위해 각 프로세스의 결과를 gather



- 클러스터 정보는 모든 프로세스가 공유
- 데이터 포인트는 분할
- 각 프로세스가 가진 데이터 포인트와 모든 클러스터 중심간의 거리를 계산
- 무게 중심 업데이트를 위해 프로세스별로 거리 정보를 합한 후, reduce 연산 수행

- 하나의 클러스터에서 중심을 계산하는 방법



In [ ]:
```
%%writefile examples/KM.py
import numpy as np
import matplotlib.pyplot as plt
from mpi4py import MPI
from sklearn.datasets import make_blobs
from tools import para_range

def distance(p1,p2):
    return np.sqrt(np.sum((p1-p2)**2))

#Implementing E step
def assign_clusters(X, clusters):
    # 각 데이터 포인트에 대해 반복
    for idx in range(X.shape[0]):
        dist = []

        curr_x = X[idx]

        # k번째 클러스터 센터와 각 데이터 포인트의 거리 계산
        for i in range(k):
            dis = distance(curr_x,clusters[i]['center'])
            dist.append(dis)

        # 계산된 클러스터와의 거리중에 최소값으로 클러스터를 배정
        curr_cluster = np.argmin(dist)

        # 현재 데이터 포인트의 좌표를 속한 클러스터에 추가
```

```
            clusters[curr_cluster]['points'].append(curr_x)
    return clusters

#Implementing the M-Step
def update_clusters(X, clusters):
    # 클러스터에 대해 반복
    for i in range(k):

        # 현재 클러스터에 포함된 데이터 포인트의 좌표를 불러옴
        points = np.array(clusters[i]['points'])

        # 데이터 포인트 좌표의 합을 먼저 구함
        # Local sum
        coord_sum_local = points.sum(axis = 0)
        # 평균 계산을 위해 데이터 포인트 좌표의 합과 데이터 포인트 개수의 global sum을 구함
        n_local_points = comm.allreduce(points.shape[0], MPI.SUM) #FIX ME
        coord_sum_global = comm.allreduce(coord_sum_local, MPI.SUM) #FIX ME

        # 데이터 포인트들의 평균을 취해 현재 클러스터 중심을 업데이트. 모든 랭크들이 같은 값을 가짐
        if n_local_points > 0:
            new_center = coord_sum_global / n_local_points
            clusters[i]['center'] = new_center
            clusters[i]['points'] = []

    return clusters

# 데이터 포인트 X에 대해 가장 거리가 가까운 클러스터를 예측
def pred_cluster(X, clusters):
    pred = []
    for i in range(X.shape[0]):
        dist = []
        for j in range(k):
            dist.append(distance(X[i],clusters[j]['center']))
        pred.append(np.argmin(dist))
    return pred

comm = MPI.COMM_WORLD

rank = comm.Get_rank()
size = comm.Get_size()

n = 500
k = 7

##### 데이터 포인트 불러오기
if rank == 0 :

    X = np.load("examples/X.npy")
    y = np.load("examples/y.npy")

    fig = plt.figure(0)
    plt.grid(True)
    plt.scatter(X[:,0], X[:,1])
    plt.show()

else :
    X = np.empty((0, 0), dtype = np.float64)

clusters = {}
np.random.seed(23)

##### 임의로 클러스터 중심을 생성. 모든 랭크들은 동일한 값을 가지고 있음
for idx in range(k):
    center = 10*(2*np.random.random((2,))-1)
    points = []
    cluster = {
        'center' : center,
        'points' : []
    }

    clusters[idx] = cluster

##### 프로세스별 범위 할당
ista, iend = para_range(n, size, rank)
chunk = iend - ista + 1

#### 각 랭크가 가질 데이터 포인트를 선언
X_chunk = np.empty([chunk,2], np.float64)

##### 데이터를 Scatterv로 분할하기 위해 chunk를 이용한 리스트 생성. 이 때 x,y좌표를 위해 2를 곱함
chunk_cnts = comm.allgather(chunk*2)

##### 데이터를 chunk로 분할. cluster 정보는 공유 (데이터 양이 크지 않음)
comm.Scatterv([X, chunk_cnts], X_chunk, root = 0) #FIX ME
comm.bcast(clusters, root = 0)

for i in range (10) :
```

```
    clusters = assign_clusters(X_chunk, clusters)
    clusters = update_clusters(X_chunk,clusters)

    if rank == 0 :
        print(clusters)

# Data point clustering
pred = pred_cluster(X_chunk,clusters)

##### prediction gather
pred_all = comm.gather(pred, root = 0) #FIX ME

if rank == 0 :
    plt.figure(1)
    plt.scatter(X[:,0],X[:,1], c = pred_all)
    for i in clusters:
        center = clusters[i]['center']
        plt.scatter(center[0],center[1],marker = '^',c = 'red')
    plt.savefig("examples/KM_paralle.png")
```

In [ ]:
```
! mpiexec -np 4 python examples/KM.py
```

In [ ]:
```

```

5. 2-D Laplace equation

5.1. 개요

- 2차 편미분 방정식 하나로 특정 상태의 공간에 대한 표현이며, 구체적으로는 어떤 물리 현상의 steady state를 표현.
  - steady state temperature
  - steady state stress
  - steady state potential distribution
  - steady state flow
- 편미분 방정식은 포괄적으로는 '주변 값과의 관계'라는 의미로 생각할 수 있으며, 특히 Laplace 방정식은 '주변 값의 평균값'이라는 의미를 가짐
- 경계 조건이 주어져있을 때 유일한 해를 가짐

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

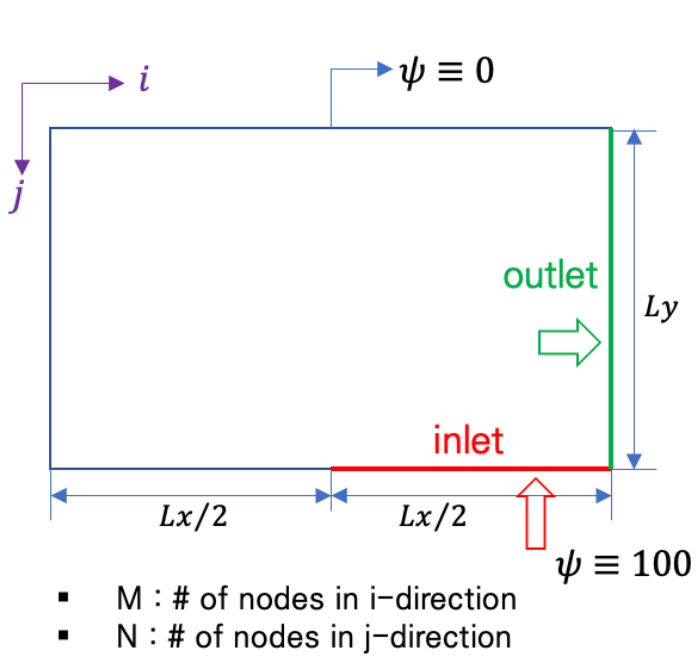
$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} = 0$$

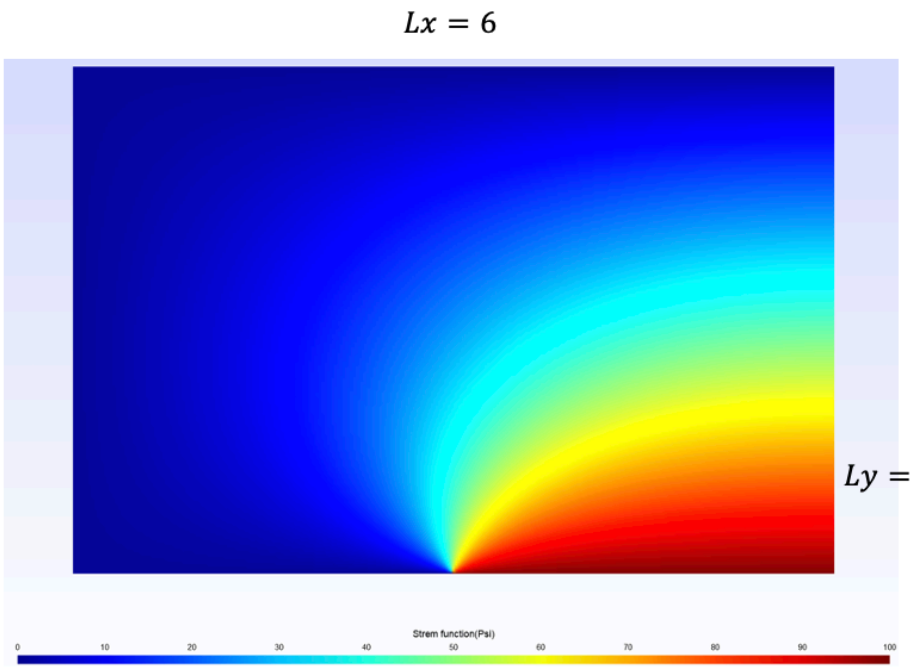
$$u_{i+1,j} + u_{i-1,j} + \beta^2 u_{i,j+1} + \beta^2 u_{i,j-1} - 2(1+\beta^2)u_{i,j} = 0$$

$$u_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)}\left[u_{i+1,j}^{k} + u_{i-1,j}^{k} + \beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k}\right)\right]$$

5.2. Model problem

- https://github.com/vishalk2/Computational-Fluid-Dynamics-CFD/tree/main/FDM/Elliptic/Stream_Function_Examples/Example_1
- 경계조건을 일부 수정





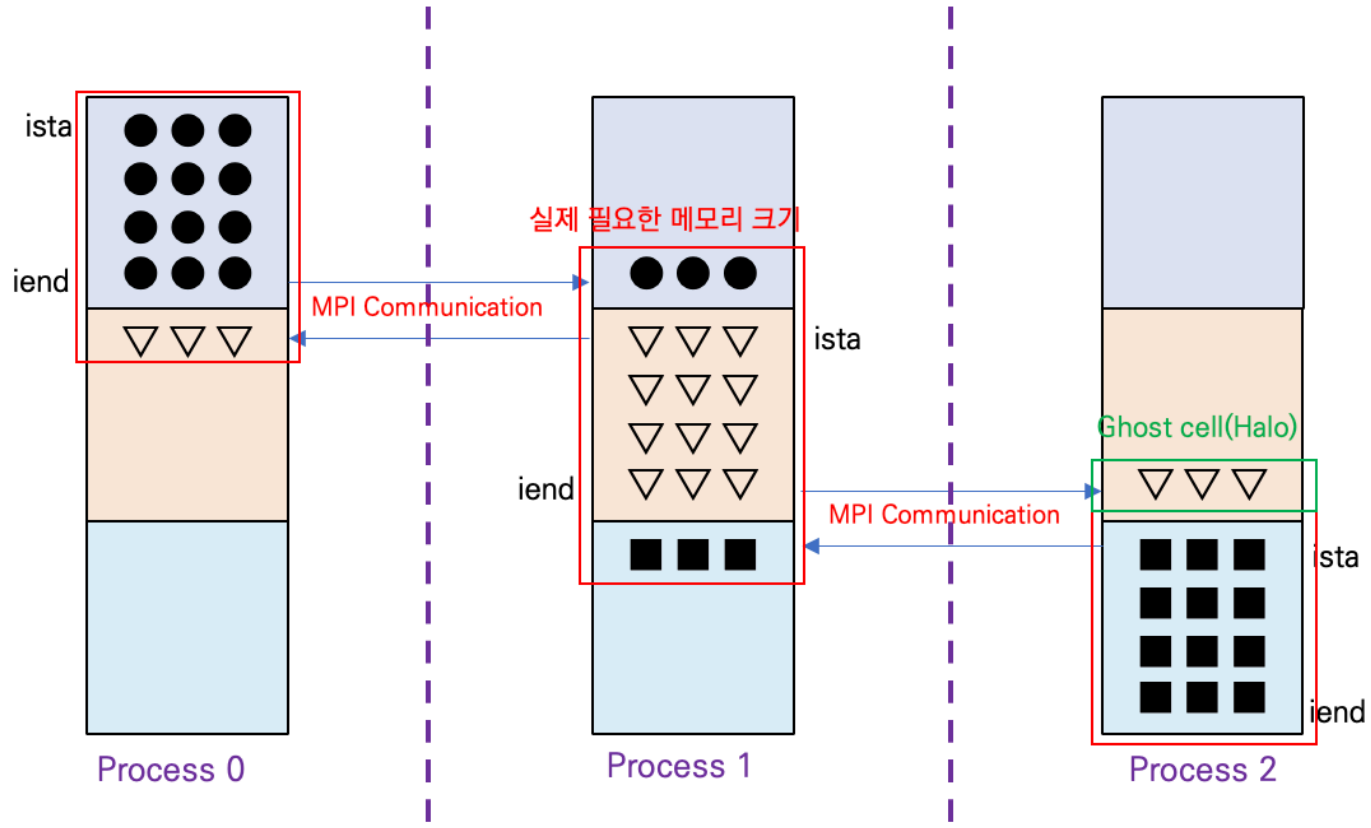
1. 순차코드

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

# Constants
LX = 6.0  # length of domain along x-direction
LY = 4.0  # length of domain along y-direction
EPSILON = 1e-5  # tolerance
MAX_ITER = 100000

M = 90 + 1
N = 60 + 1
```

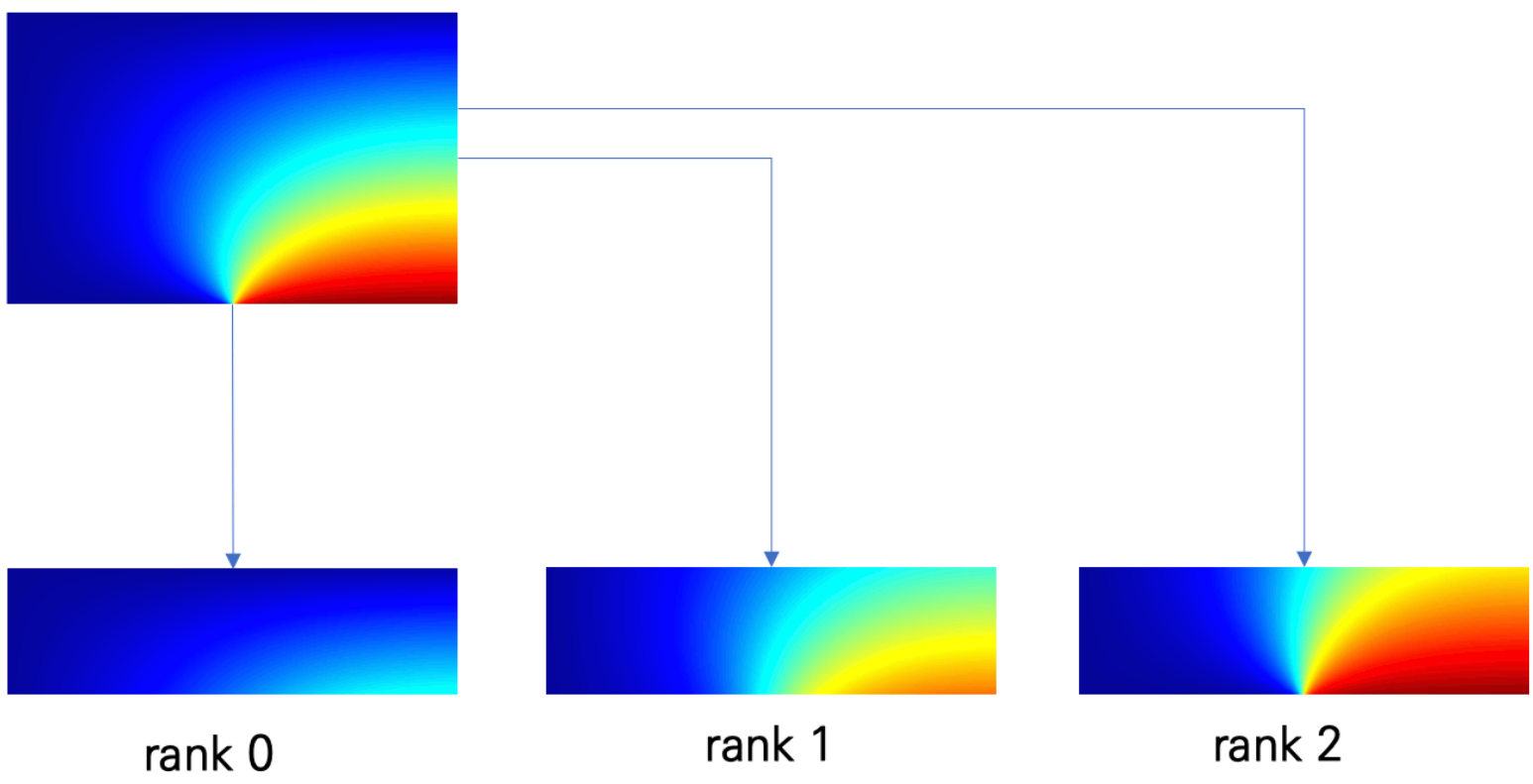
```
dx = LX / (M - 1)
dy = LY / (N - 1)
beta = dx / dy
beta_1 = 1.0 / (2.0 * (1.0 + beta * beta))

def plot(figID, psi, zmin, zmax):
    plt.figure(figID)
    plt.clf()
    plt.pcolormesh(psi, cmap=plt.cm.jet, vmin=zmin, vmax=zmax)
    plt.colorbar()
    plt.draw()
    plt.xlim([0, M])
    plt.ylim([0, N])
    plt.pause(0.1)

def Jacobi_iter(N, M, psi_new, beta, beta_1):

    error = 0.0
    psi_old = np.zeros((N, M))

    for iter in range(0, MAX_ITER):

        psi_old = np.copy(psi_new)

        for i in range(1, N - 1):
            for j in range(1, M - 1):
                psi_new[i][j] = beta_1 * (psi_old[i][j + 1] + psi_old[i][j - 1] +
                                          beta * beta * (psi_old[i + 1][j] + psi_old[i - 1][j]))

        # Right Neumann Boundary Condition
        for i in range(N):
            psi_new[i][M - 1] = psi_new[i][M - 2]

        error = 0.0
        for i in range(N):
            for j in range(M):
                error += (psi_new[i][j] - psi_old[i][j]) * (psi_new[i][j] - psi_old[i][j])
        error = math.sqrt(error / (M * N))

        if iter % 100 == 0:
            print(f"Iteration = {iter}, Error = {error:.6e}")
        if iter % 1000 == 0:
            plot(1, psi_new, -1, 101)

        if error <= EPSILON:
            break

    print(f"Iteration = {iter}, Error = {error:.6e}")

psi_new = np.zeros((N, M))

for i in range(N):
    for j in range(M):
        psi_new[i][j] = 0.0

# Boundary conditions
divide = int((M - 1) * 0.5)
for i in range(divide):
    psi_new[N - 1][i] = 0.0  # upper (left)
for i in range(divide, M):
    psi_new[N - 1][i] = 100.0  # upper (right)
for i in range(N):
    psi_new[i][0] = 0.0  # left wall
for i in range(M):
    psi_new[0][i] = 0.0  # lower wall

# Jacobi iteration
Jacobi_iter(N, M, psi_new, beta, beta_1)
```

2. 병렬코드

- Row-wise decomposition



- 계산 영역의 분할



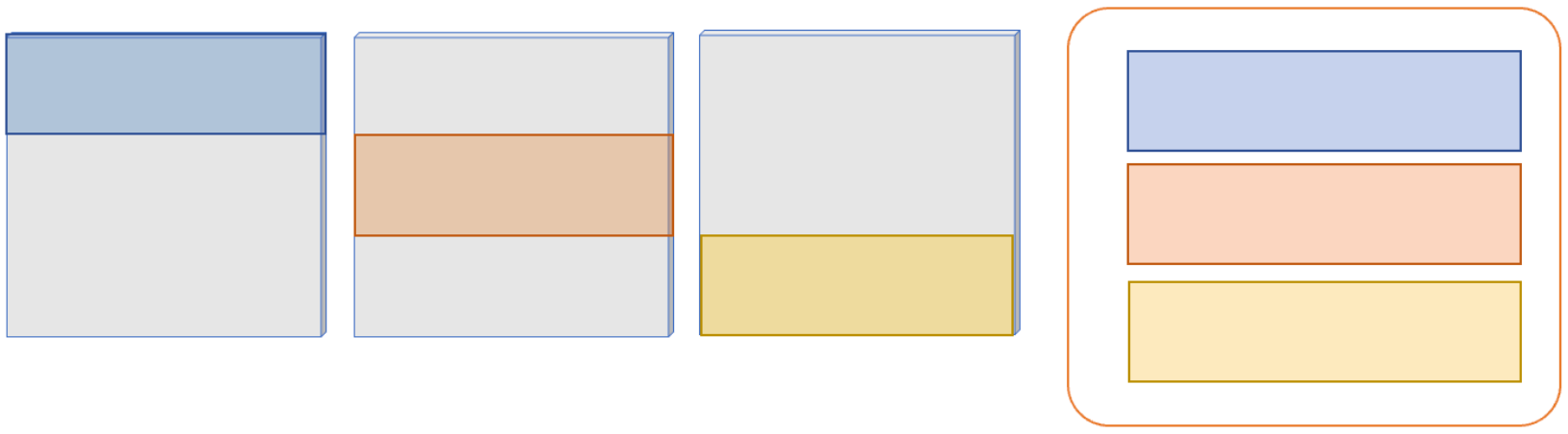
In [ ]:
```
%%writefile examples/jacobi.py
import numpy as np
import matplotlib.pyplot as plt
import math
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Constants
LX = 6.0  # length of domain along x-direction
LY = 4.0  # length of domain along y-direction
EPSILON = 1e-5  # tolerance
MAX_ITER = 100000

M = 90 + 1
N = 60 + 1
dx = LX / (M - 1)
dy = LY / (N - 1)
beta = dx / dy
beta_1 = 1.0 / (2.0 * (1.0 + beta * beta))

def para_range(n1, n2, size, rank) :
    iwork = divmod((n2 - n1 + 1), size)
    ista = rank * iwork[0] + n1 + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1
    return ista, iend
```

```
##### 분할된 영역을 그림 출력을 위해 취합하는 함수
def gather_data(psi) :

    ista, iend = para_range(0, N - 1, size, rank)

    ##### 각 랭크별로 분할된 영역의 인덱스를 루트 랭크로 취합
    istas = comm.gather(ista, root=0)
    iends = comm.gather(iend, root=0)

    ##### Gather 대신 send/recv를 이용하는 예
    ##### 루트 랭크에서 리스트로 취합한 각 랭크별 시작/종료 인덱스를 이용하여 수신한 데이터를 올바른 위치로 복사
    if rank == 0:
        for i in range(1, size):
            comm.Recv(psi[istas[i]:iends[i]+1,:], source=i, tag=i) #FIX ME
    else:
        comm.Send(psi[ista:iend+1,:], dest=0, tag=rank) #FIX ME

def plot(figID, psi, zmin, zmax):
    plt.figure(figID)
    plt.clf()
    plt.pcolormesh(psi, cmap=plt.cm.jet, vmin=zmin, vmax=zmax)
    plt.colorbar()
    plt.savefig("examples/jacobi_paralle_sol.png")
    plt.xlim([0, M])
    plt.ylim([0, N])
    plt.pause(0.1)

def Jacobi_iter(N, M, psi_new, beta, beta_1):

    ista, iend = para_range(0, N - 1, size, rank)

    ##### 각 랭크별 영역 범위와 이웃 랭크들을 설정
    ista_p1 = ista
    iend_m1 = iend
    inext = rank + 1
    iprev = rank - 1

    ##### 경계 랭크는 시작점 및 끝점에 대한 수정이 필요
    ##### 경계에 위치한 랭크는 MPI.PROC_NULL로 없음을 표시
    if rank == 0:
        ista_p1 = ista + 1
        iprev = MPI.PROC_NULL
    elif rank == size - 1:
        iend_m1 = iend - 1
        inext = MPI.PROC_NULL

    error = 0.0
    error_local = 0.0
    psi_old = np.zeros((N, M))

    for iter in range(0, MAX_ITER):

        for i in range(ista, iend + 1):
            for j in range(M):
                psi_old[i][j] = psi_new[i][j]

        # MPI Communication : request 설정
        reqs1 = []
        reqs2 = []

        ##### 경계에 대한 송수신을 수행. 실제 데이터가 있는 인덱스와 고스트 셀의 인덱스에 주의
        if inext != MPI.PROC_NULL:
            reqs1.append(comm.Isend(psi_old[iend], inext, tag=1))
            reqs1.append(comm.Irecv(psi_old[iend + 1], inext, tag=2))

        if iprev != MPI.PROC_NULL:
            reqs2.append(comm.Isend(psi_old[ista], iprev, tag=2))
            reqs2.append(comm.Irecv(psi_old[ista - 1], iprev, tag=1))

        ##### 통신 완료 대기
        if inext != MPI.PROC_NULL:
            for req in reqs1:
                MPI.Request.Wait(req)

        if iprev != MPI.PROC_NULL:
            for req in reqs2:
                MPI.Request.Wait(req)

        ##### 각 랭크별로 해당 영역을 업데이트
        for i in range(ista_p1, iend_m1 + 1): #FIX ME
            for j in range(1, M - 1):
                psi_new[i][j] = beta_1 * (psi_old[i][j + 1] + psi_old[i][j - 1] +
                                          beta * beta * (psi_old[i + 1][j] + psi_old[i - 1][j]))

        # Right Neumann Boundary Condition : 분할 방향을 고려하면 모든 랭크에서 수행
        for i in range(ista, iend + 1): #FIX ME
            psi_new[i][M - 1] = psi_new[i][M - 2]
```

```
        error_local = 0.0
        error = 0.0
        # 각 랭크별로 오차를 계산
        for i in range(ista, iend + 1):
            for j in range(M):
                error_local += (psi_new[i][j] - psi_old[i][j]) * (psi_new[i][j] - psi_old[i][j])
        ##### error reduction : 모든 랭크들이 같은 오차를 갖도록 함
        error = comm.allreduce(error_local, op=MPI.SUM) #FIX ME
        error = math.sqrt(error / (M * N))

        if iter % 10 == 0:
            if rank == 0:
                print(f"Iteration = {iter}, Error = {error:.6e}")

        if error <= EPSILON:
            break

    if rank == 0:
        print(f"Iteration = {iter}, Error = {error:.6e}")

psi_new = np.zeros((N, M))

for i in range(N):
    for j in range(M):
        psi_new[i][j] = 0.0

# Boundary conditions
divide = int((M - 1) * 0.5)

for i in range(divide):
    psi_new[N - 1][i] = 0.0  # upper (left)
for i in range(divide, M):
    psi_new[N - 1][i] = 100.0  # upper (right)
for i in range(N):
    psi_new[i][0] = 0.0  # left wall
for i in range(M):
    psi_new[0][i] = 0.0  # lower wall

# Jacobi iteration
Jacobi_iter(N, M, psi_new, beta, beta_1)

gather_data(psi_new)
if rank == 0:
    plot(1, psi_new, -1, 101)
```

In [ ]:
```
!mpiexec -np 4 python examples/jacobi.py
```

3. 병렬코드 - 영역분할
   - 분할의 두 가지 방식



	Shared data decomp.	Domain decomp.
Memory space	All data	Its own data
Calculation	Its own data	Its own data
Communication pattern	Update its own data to shared data	Exchange necessary data
Implementation	Simple (It is like OpenMP)	Complicated

- 영역분할 (Domain decomposition)
  - 계산영역만 메모리 할당 : 메모리 절감
  - 인덱스 수정

In [ ]:

```
%%writefile examples/jacobi2.py
import numpy as np
import matplotlib.pyplot as plt
import math
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Constants
LX = 6.0  # length of domain along x-direction
LY = 4.0  # length of domain along y-direction
EPSILON = 1e-5  # tolerance
MAX_ITER = 100000

M = 90 + 1
N = 60 + 1
dx = LX / (M - 1)
dy = LY / (N - 1)
beta = dx / dy
beta_1 = 1.0 / (2.0 * (1.0 + beta * beta))

def para_range(n1, n2, size, rank) :
    iwork = divmod((n2 - n1 + 1), size)
    ista = rank * iwork[0] + n1 + min(rank, iwork[1])
    iend = ista + iwork[0] - 1
    if iwork[1] > rank :
        iend = iend + 1
    return ista, iend

##### 분할된 영역을 그림 출력을 위해 취합하는 함수
def gather_data(psi, psi_all) :

    ista, iend = para_range(0, N - 1, size, rank)
    domain_size = iend - ista + 1

    ##### 각 랭크별 영역의 데이터 영역 정의 : 고스트셀을 제외해야 함
    psi_tmp = np.array(psi[1:domain_size + 1,:])

    ##### 각 랭크별로 보낼 데이터 개수를 리스트로 생성
    cound_send = np.array(comm.allgather(domain_size))

    ##### Gatherv를 이용하여 데이터 취합
    comm.Gatherv(psi_tmp, recvbuf = (psi_all, cound_send*M), root = 0)

def plot(figID, psi, zmin, zmax):
    plt.figure(figID)
    plt.clf()
    plt.pcolormesh(psi, cmap=plt.cm.jet, vmin=zmin, vmax=zmax)
    plt.colorbar()
    plt.savefig("examples/jacobi2_paralle_sol.png")
    plt.xlim([0, M])
    plt.ylim([0, N])
    plt.pause(0.1)

def Jacobi_iter(N, M, psi_new, beta, beta_1):
    ista, iend = para_range(0, N - 1, size, rank)
    domain_size = iend - ista + 1

    ##### 각 랭크별 영역 범위와 이웃 랭크들을 설정
    i_s = 1
    i_e = domain_size
    inext = rank + 1
    iprev = rank - 1

    ##### 경계 랭크는 시작점 및 끝점에 대한 수정이 필요
    ##### 경계에 위치한 랭크는 MPI.PROC_NULL로 없음을 표시
    if rank == 0:
        i_s = 2
        iprev = MPI.PROC_NULL
    elif rank == size - 1:
        i_e = domain_size - 1
        inext = MPI.PROC_NULL

    error = 0.0
    error_local = 0.0
    psi_old = np.zeros((domain_size + 2, M))

    for iter in range(0, MAX_ITER):

        for i in range(1, domain_size + 1):
            for j in range(M):
                psi_old[i][j] = psi_new[i][j]

        # MPI Communication : request 설정
        reqs1 = []
```

```
        reqs2 = []

        ##### 경계에 대한 송수신을 수행. 실제 데이터가 있는 인덱스와 고스트 셀의 인덱스에 주의
        if inext != MPI.PROC_NULL:
            reqs1.append(comm.Isend(psi_old[domain_size], inext, tag=1))
            reqs1.append(comm.Irecv(psi_old[domain_size + 1], inext, tag=2))

        if iprev != MPI.PROC_NULL:
            reqs2.append(comm.Isend(psi_old[1], iprev, tag=2))
            reqs2.append(comm.Irecv(psi_old[0], iprev, tag=1))

        ##### 통신 완료 대기
        if inext != MPI.PROC_NULL:
            for req in reqs1:
                MPI.Request.Wait(req)

        if iprev != MPI.PROC_NULL:
            for req in reqs2:
                MPI.Request.Wait(req)

        ##### 각 랭크별로 해당 영역을 업데이트
        for i in range(i_s, i_e + 1): #FIX ME
            for j in range(1, M - 1):
                psi_new[i][j] = beta_1 * (psi_old[i][j + 1] + psi_old[i][j - 1] +
                                          beta * beta * (psi_old[i + 1][j] + psi_old[i - 1][j]))

        # Right Neumann Boundary Condition : 분할 방향을 고려하면 모든 랭크에서 수행
        for i in range(1, domain_size + 1):
            psi_new[i][M - 1] = psi_new[i][M - 2]

        error_local = 0.0
        error = 0.0
        # 각 랭크별로 오차를 계산
        for i in range(1, domain_size + 1):
            for j in range(M):
                error_local += (psi_new[i][j] - psi_old[i][j]) * (psi_new[i][j] - psi_old[i][j])
        ##### error reduction : 모든 랭크들이 같은 오차를 갖도록 함
        error = comm.allreduce(error_local, op=MPI.SUM)
        error = math.sqrt(error / (M * N))

        psi_all = np.zeros((N, M))

        if iter % 10 == 0:
            if rank == 0:
                print(f"Iteration = {iter}, Error = {error:.6e}")

        if error <= EPSILON:
            gather_data(psi_new, psi_all)
            if rank == 0:
                plot(1, psi_all, -1, 101)
            break

    if rank == 0:
        print(f"Iteration = {iter}, Error = {error:.6e}")

ista, iend = para_range(0, N - 1, size, rank)
domain_size = iend - ista + 1

##### 계산할 영역과 고스트셀만큼만 영역으로 선언
psi_new = np.zeros((domain_size + 2, M))

for i in range(1, domain_size + 1):
    for j in range(M):
        psi_new[i][j] = 0.0

# Boundary conditions
divide = int((M - 1) * 0.5)

if rank == size - 1:
    for i in range(divide):
        psi_new[domain_size][i] = 0.0  # upper (left)
    for i in range(divide, M):
        psi_new[domain_size][i] = 100.0  # upper (right)
for i in range(1, domain_size + 1):
    psi_new[i][0] = 0.0  # left wall
if rank == 0 :
    for i in range(M):
        psi_new[1][i] = 0.0  # bottom wall

# Jacobi iteration
Jacobi_iter(N, M, psi_new, beta, beta_1)
```

In [ ]:
```
!mpiexec -n 4 python examples/jacobi2.py
```

In [ ]:
```
```

6. 고속 푸리에 변환 (Fast Fourier Transform, FFT)

6.1. 개요

- 이산 푸리에 변환(영어: Discrete Fourier Transform, DFT)과 그 역변환을 빠르게 수행하는 알고리즘.
- FFT는 디지털 신호 처리에서 편미분 방정식의 근을 구하는 알고리즘에 이르기까지 많은 분야에서 사용.
  - 스펙트럼 분석기
  - OFDM 변복조기
  - CT 스캐너, MRI
  - MP3 압축방식
  - 영상 필터

6.2. 병렬화

1. 순차코드

- numpy library 이용
- 2D FFT 수행

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

img = plt.imread('images/lenna.png')
imgr = img[:,:,0]
H = np.fft.fft2(imgr)
Hs = np.fft.fftshift(H)

ly, lx = imgr.shape

kY, kX = np.ogrid[-ly/2:ly/2, -lx/2:lx/2]

sig1, sigh, sigm = 30, 20, 50
Flow = np.exp( -(kX**2 + kY**2) / (2*sig1**2))
Fhgh = 1 - np.exp( -(kX**2 + kY**2) / (2*sigh**2))
Fmix = 1 - 0.7 * np.exp( -(kX**2 + kY**2) / (2*sigm**2))

Glow = Hs.copy() * Flow
Ghgh = Hs.copy() * Fhgh
Gmix = Hs.copy() * Fmix
iGlow = np.fft.ifftshift(Glow)
iGhgh = np.fft.ifftshift(Ghgh)
iGmix = np.fft.ifftshift(Gmix)
glow = np.fft.ifft2(iGlow)
ghgh = np.fft.ifft2(iGhgh)
gmix = np.fft.ifft2(iGmix)

fig = plt.figure(1, figsize = (10, 7))
plt.subplot(221)
plt.imshow(imgr, cmap = 'gray')
plt.axis('off')
plt.text(430,40,r'$(a)$', color = 'k')

plt.subplot(222)
plt.imshow(np.abs(glow), cmap = 'gray')
plt.axis('off')
plt.text(430,40,r'$(b)$', color = 'k')

ax = plt.subplot(223)
plt.imshow(np.abs(ghgh), cmap = 'gray')
plt.axis('off')
plt.text(430,40,r'$(c)$', color = 'w')

ax = plt.subplot(224)
plt.imshow(np.abs(gmix), cmap = 'gray')
plt.axis('off')
plt.text(430,40,r'$(d)$', color = 'w')

plt.tight_layout()
plt.show()
```

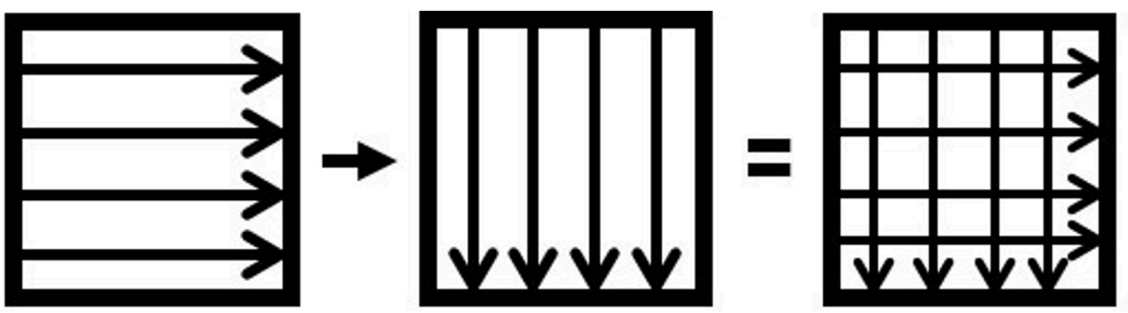
In [ ]:
```
fig = plt.figure(2, figsize = (10, 7))
ax1 = plt.subplot(131)
pos = plt.imshow(Flow, cmap = 'gray')
plt.axis('off')
fig.colorbar(pos, ax=ax1, fraction=0.046, pad=0.04)
plt.text(40,40,r'$(a)$ Low-pass filter', color = 'w')

ax2 = plt.subplot(132)
```

```
pos = plt.imshow(Fhgh, cmap = 'gray')
plt.axis('off')
fig.colorbar(pos, ax=ax2, fraction=0.046, pad=0.04)
plt.text(40,40,r'$(b)$ High-pass filter', color = 'k')

ax3 = plt.subplot(133)
pos = plt.imshow(Fmix, cmap = 'gray')
plt.axis('off')
fig.colorbar(pos, ax=ax3, fraction=0.046, pad=0.04)
plt.text(40,40,r'$(b)$ Mixed filter', color = 'k')
```

2. High-pass 필터 순차코드

- 2D FFT를 x, y 방향 1D FFT로 수행

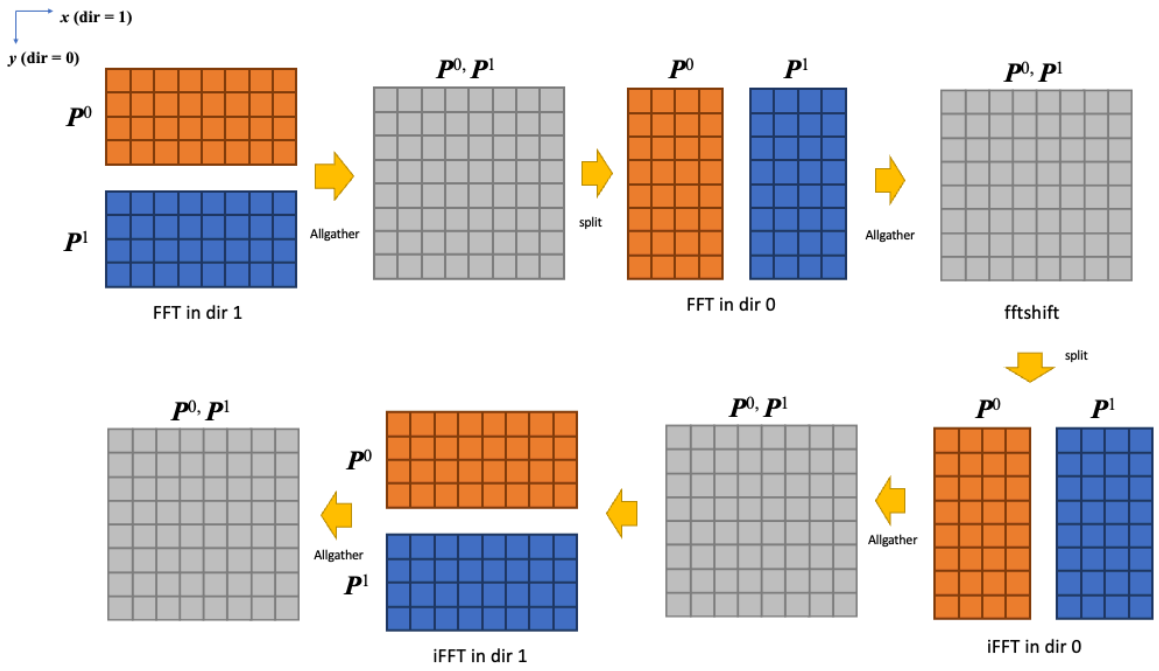
$$H(n_1, n_2) \equiv \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} \exp(2\pi i k_2 n_2 / N_2)\ \exp(2\pi i k_1 n_1 / N_1)\ h(k_1, k_2)$$



In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

img = plt.imread('images/lenna.png')
imgr = img[:,:,0]
H = np.fft.fft2(imgr)
Hs = np.fft.fftshift(H)

ly, lx = imgr.shape

sigh = 20

kY, kX = np.ogrid[-ly/2:ly/2, -lx/2:lx/2]
Fhgh = 1 - np.exp( -(kX**2 + kY**2) / (2*sigh**2))

Ghgh = Hs.copy() * Fhgh
iGhgh = np.fft.ifftshift(Ghgh)
ghgh = np.fft.ifft2(iGhgh)

fig = plt.figure(1, figsize = (10, 7))
plt.subplot(121)
plt.imshow(imgr, cmap = 'gray')
plt.axis('off')
plt.text(430,40,r'$(a)$', color = 'k')

plt.subplot(122)
plt.imshow(np.abs(ghgh), cmap = 'gray')
plt.axis('off')
plt.text(430,40,r'$(b)$', color = 'k')

plt.tight_layout()
plt.show()
```

3. High-pass 필터 병렬화

- 2D FFT를 x, y 방향 1D FFT로 수행
- 각 방향별로 병렬화
- FFT는 한 방향으로 모든 데이터가 필요
- 방향에 따라 데이터를 분할



In [ ]:

```
%%writefile examples/FFT2D.py
import matplotlib.pyplot as plt
import numpy as np
from mpi4py import MPI
from tools import para_range

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

img = plt.imread('images/lenna.png')
imgr = img[:,:,0]

ly, lx = imgr.shape
kY, kX = np.ogrid[-ly/2:ly/2, -lx/2:lx/2]

sigh = 20
Fhgh = 1 - np.exp( -(kX**2 + kY**2) / (2*sigh**2))

##### 1D FFT 2개로 분할
##### 계산 범위 지정 (y 방향)
jsta, jend = para_range(ly, size, rank) # FIX ME
H_local_y = np.fft.fft2(imgr[jsta:jend+1,:], axes = [1]).copy()
H_all = np.empty(imgr.shape, dtype = np.complex128)

##### 계산된 범위 취합
comm.Allgather(H_local_y, H_all)

##### 계산 범위 지정 (x 방향)
ista, iend = para_range(lx, size, rank) #FIX ME
H_local_x = np.fft.fft2(H_all[:,ista:iend+1], axes = [0]).copy()

##### 계산된 범위 취합 : 메모리 불연속으로 행별로 Allgather 수행
for i in range (ly) :
    comm.Allgather(H_local_x[i], H_all[i]) #FIX ME

Hs = np.fft.fftshift(H_all)
Ghgh = Hs.copy() * Fhgh
iGhgh = np.fft.ifftshift(Ghgh)

##### y방향 inverse FFT의 계산 범위 분할
G_local_y = np.fft.ifft2(iGhgh[jsta:jend+1,:], axes = [1]).copy()
ghgh = np.empty(imgr.shape, dtype = np.complex128)

##### 계산된 범위 취합
comm.Allgather(G_local_y, ghgh) # FIX ME

##### 계산 범위 지정 (x 방향)
G_local_x = np.fft.ifft2(ghgh[:,ista:iend+1], axes = [0]).copy()

##### 계산된 범위 취합 : 메모리 불연속
for i in range (ly) :
    comm.Allgather( G_local_x[i], ghgh[i])

if rank == 0 :
    fig = plt.figure(1, figsize = (10, 7))
    plt.subplot(121)
    plt.imshow(imgr, cmap = 'gray')
    plt.axis('off')
    plt.text(430,40,r'$(a)$', color = 'k')

    plt.subplot(122)
    plt.imshow(np.abs(ghgh), cmap = 'gray')
    plt.axis('off')
    plt.text(430,40,r'$(b)$', color = 'k')
```

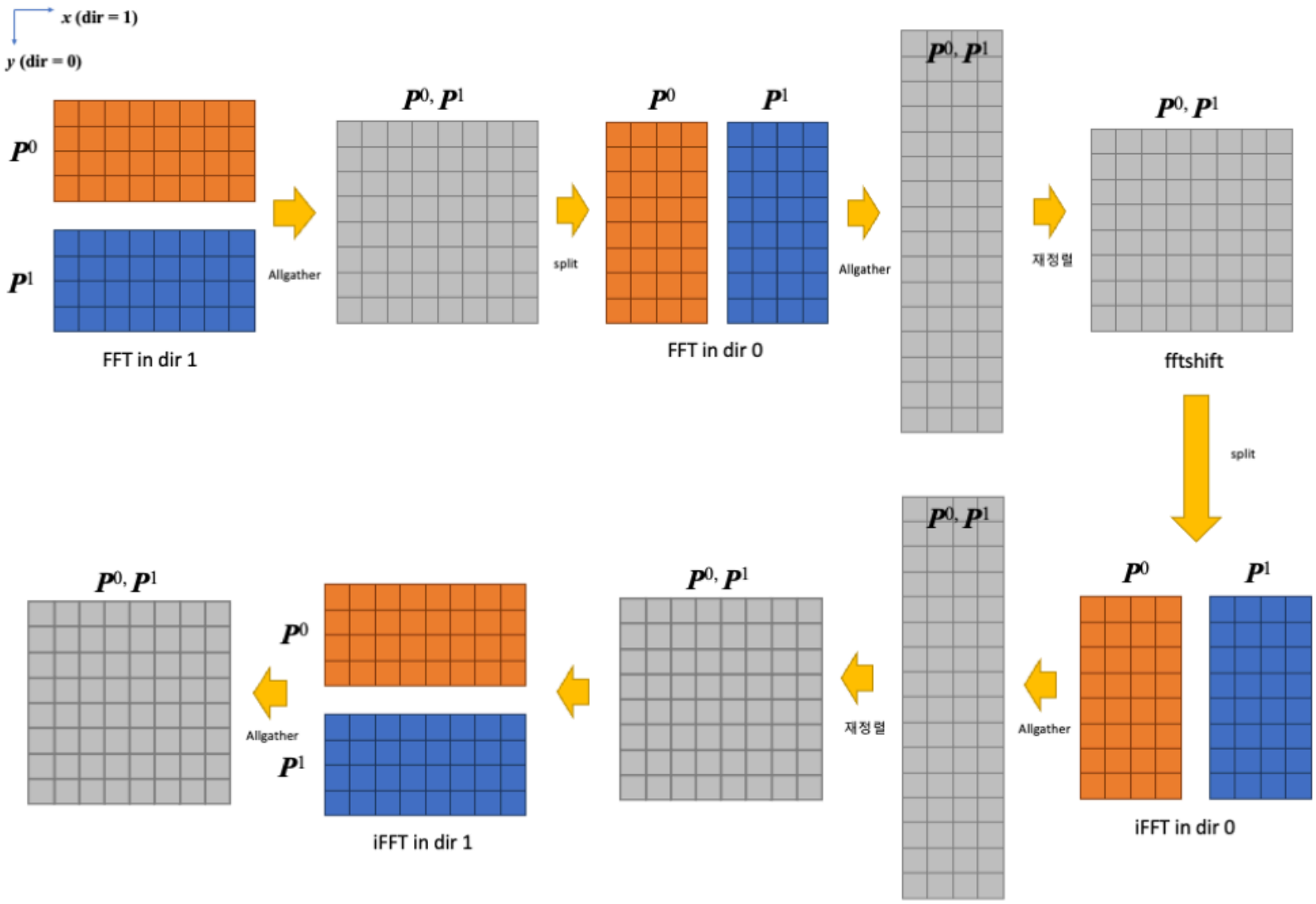
```
    plt.tight_layout()
    plt.savefig('examples/FFT2D.png')
    plt.show()
```

In [ ]:
```
!mpiexec -np 2 python examples/FFT2D.py
```

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('examples/FFT2D.png')
plt.imshow(img)
plt.show()
```

4. Data packing/unpacking

- 여러번 호출되는 Allgather 함수를 한 번의 호출로 변경
- 데이터 재배열 필요



In [ ]:
```
%%writefile examples/FFT2D_2.py
import matplotlib.pyplot as plt
import numpy as np
from mpi4py import MPI
from tools import para_range

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

img = plt.imread('images/lenna.png')
imgr = img[:,:,0]

ly, lx = imgr.shape
kY, kX = np.ogrid[-ly/2:ly/2, -lx/2:lx/2]

sigh = 20
Fhgh = 1 - np.exp( -(kX**2 + kY**2) / (2*sigh**2))

##### 1D FFT 2개로 분할
##### 계산 범위 지정 (y 방향)
jsta, jend = para_range(ly, size, rank)
H_local_y = np.fft.fft2(imgr[jsta:jend+1,:], axes = [1]).copy()
H_all = np.empty(imgr.shape, dtype = np.complex128)

##### 계산된 범위 취합
comm.Allgather(H_local_y, H_all)
```

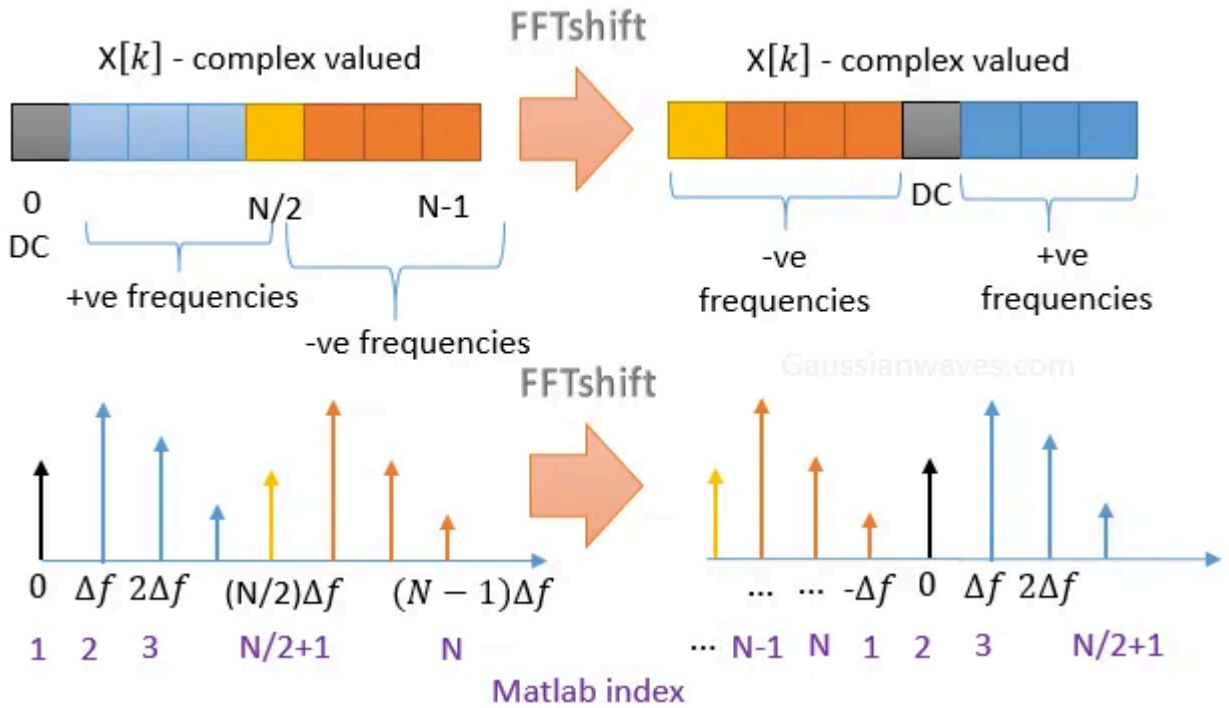
```
##### 계산 범위 지정 (x 방향)
ista, iend = para_range(lx, size, rank)
H_local_x = np.fft.fft2(H_all[:,ista:iend+1], axes = [0]).copy()

chunk = int(lx/size)
H_all2 = np.empty((size * ly, chunk), dtype = np.complex128)

##### 계산 범위 취합 방식 수정
comm.Allgather(H_local_x, H_all2) # FIX ME

##### 데이터의 랭크간 전치 : 인덱스에 주의
for i in range (ly) :
    for p in range (size) :
        H_all[i][p*chunk:p*chunk+chunk] = H_all2[i+p*ly][0:chunk]

Hs = np.fft.fftshift(H_all)
Ghgh = Hs.copy() * Fhgh
iGhgh = np.fft.ifftshift(Ghgh)

##### y방향 inverse FFT의 계산 범위 분할 및 FFT
G_local_y = np.fft.ifft2(iGhgh[jsta:jend+1,:], axes = [1]).copy()
ghgh = np.empty(imgr.shape, dtype = np.complex128)

##### 계산된 범위 취합
comm.Allgather(G_local_y, ghgh)

##### 계산 범위 지정 (x 방향)
G_local_x = np.fft.ifft2(ghgh[:,ista:iend+1], axes = [0]).copy()

chunk = int(lx/size)
ghgh_2 = np.empty((size * ly, chunk), dtype = np.complex128)

##### 계산 범위 취합 방식 수정
comm.Allgather(G_local_x, ghgh_2)

##### 데이터의 랭크간 전치 : 인덱스에 주의
for i in range (ly) :
    for p in range (size) :
        ghgh[i][p*chunk:p*chunk+chunk] = ghgh_2[i+p*ly][0:chunk]

if rank == 0 :
    fig = plt.figure(1, figsize = (10, 7))
    plt.subplot(121)
    plt.imshow(imgr, cmap = 'gray')
    plt.axis('off')
    plt.text(430,40,r'$(a)$', color = 'k')

    plt.subplot(122)
    plt.imshow(np.abs(ghgh), cmap = 'gray')
    plt.axis('off')
    plt.text(430,40,r'$(b)$', color = 'w')

    plt.tight_layout()
    plt.savefig('examples/FFT2D_2.png')
    plt.show()
```

In [ ]:
```
!mpiexec -np 2 python examples/FFT2D_2.py
```

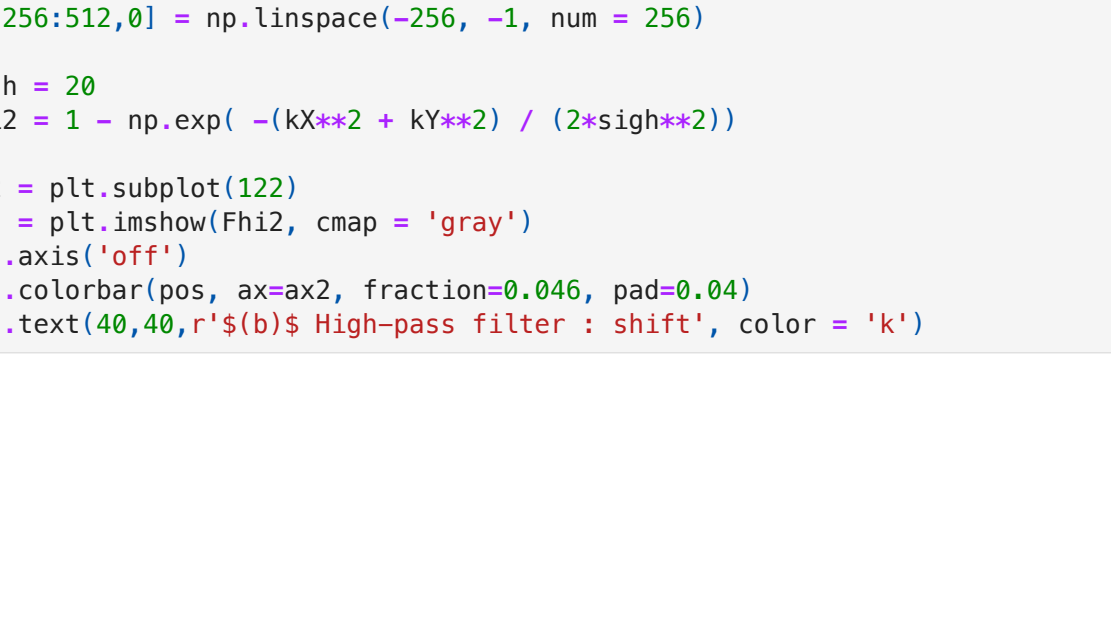
In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('examples/FFT2D_2.png')
plt.imshow(img)
plt.show()
```

5. Filter shape 변경

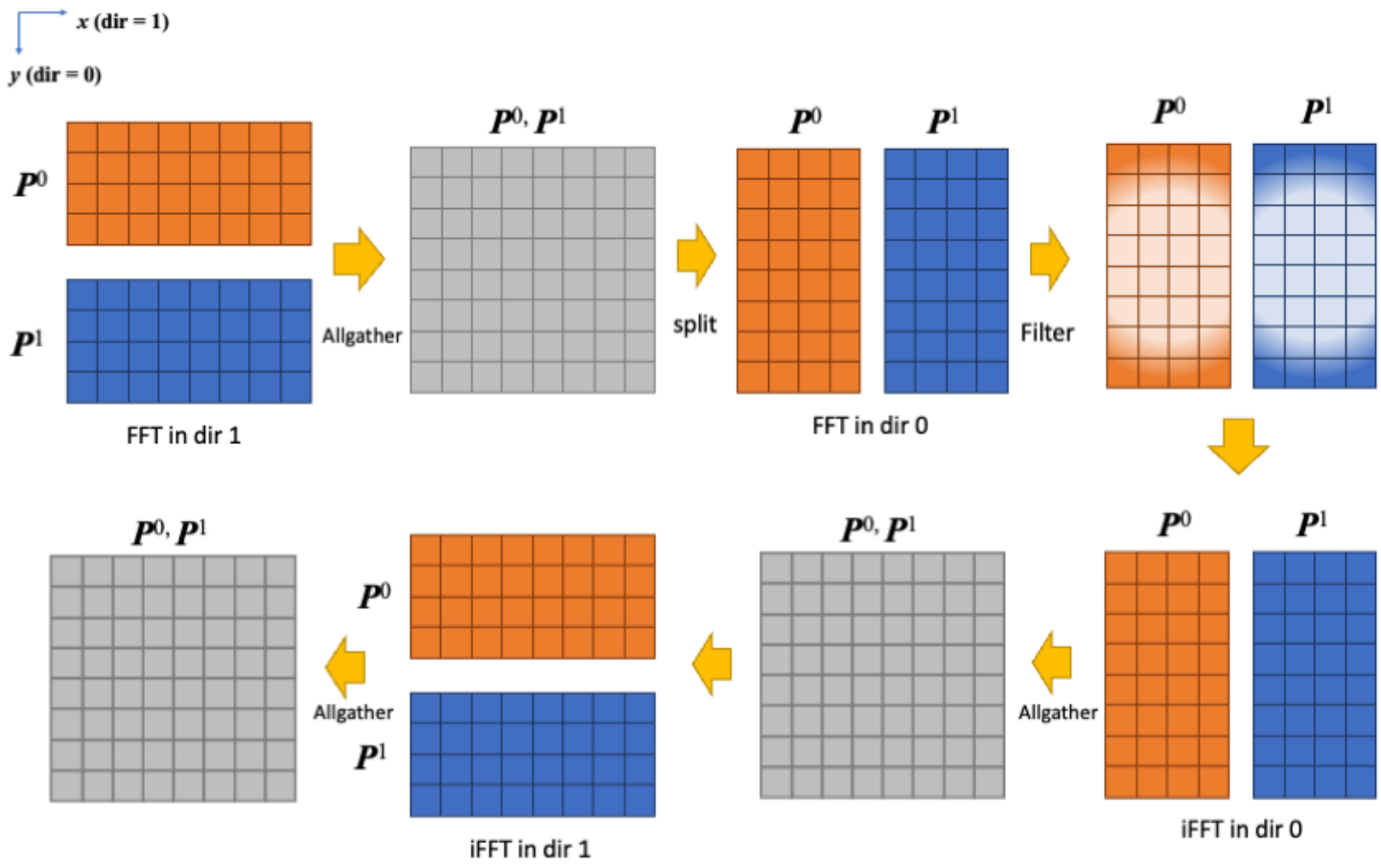
- FFT 후 index의 순서가 변경되어 filter 적용을 위해서는 shift 연산이 필요



- 2차원 FFT에 대해서도 filter shape과 FFT shift의 특성을 이용하면 분할된 상태에서 필터 적용 가능



In [ ]:
```
kY, kX = np.ogrid[-ly/2:ly/2, -lx/2:lx/2]

sigh = 20
Fhgh = 1 - np.exp( -(kX**2 + kY**2) / (2*sigh**2))

fig = plt.figure(2, figsize = (10, 7))
ax1 = plt.subplot(121)
pos = plt.imshow(Fhgh, cmap = 'gray')
plt.axis('off')
fig.colorbar(pos, ax=ax1, fraction=0.046, pad=0.04)
plt.text(40,40,r'$(a)$ High-pass filter : original', color = 'k')

kY, kX = np.ogrid[0:ly, 0:lx]
kX[0,256:512] = np.linspace(-256, -1, num = 256)
kY[256:512,0] = np.linspace(-256, -1, num = 256)

sigh = 20
Fhi2 = 1 - np.exp( -(kX**2 + kY**2) / (2*sigh**2))

ax2 = plt.subplot(122)
pos = plt.imshow(Fhi2, cmap = 'gray')
plt.axis('off')
fig.colorbar(pos, ax=ax2, fraction=0.046, pad=0.04)
plt.text(40,40,r'$(b)$ High-pass filter : shift', color = 'k')
```



In [ ]:

```
%%writefile examples/FFT2D_3.py
import matplotlib.pyplot as plt
import numpy as np
from mpi4py import MPI
from tools import para_range

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

img = plt.imread('images/lenna.png')
imgr = img[:,:,0]

ly, lx = imgr.shape

##### 필터 수정
kY, kX = np.ogrid[0:ly, 0:lx]
kX[0,256:512] = np.linspace(-256, -1, num = 256)
kY[256:512,0] = np.linspace(-256, -1, num = 256)

sigh = 20
Fhi2 = 1 - np.exp( -(kX**2 + kY**2) / (2*sigh**2))

##### 1D FFT 2개로 분할
##### 계산 범위 지정 (y 방향)
jsta, jend = para_range(ly, size, rank)
H_local_y = np.fft.fft2(imgr[jsta:jend+1,:], axes = [1]).copy()
H_all = np.empty(imgr.shape, dtype = np.complex128)

##### 계산된 범위 취합
comm.Allgather(H_local_y, H_all)

##### 계산 범위 지정 (x 방향)
ista, iend = para_range(lx, size, rank)
H_local_x = np.fft.fft2(H_all[:,ista:iend+1], axes = [0]).copy()

##### 데이터 취합 및 Shift 불필요, 분할된 데이터에 수정된 Filter영역 곱
iGhgh = H_local_x * Fhi2[:,ista:iend+1]#FIX ME

##### 분할된 상태에서 inverse FFT
G_local_x = np.fft.ifft2(iGhgh, axes = [0]).copy()

##### y방향 FFT를 위해 데이터 취합
chunk = int(lx/size)
ghgh_2 = np.empty((size * ly, chunk), dtype = np.complex128)

##### 계산 범위 취합 방식 수정
comm.Allgather(G_local_x, ghgh_2)

##### 데이터의 랭크간 전치 : 인덱스에 주의
ghgh = np.empty(imgr.shape, dtype = np.complex128)
for i in range (ly) :
    for p in range (size) :
        ghgh[i][p*chunk:p*chunk+chunk] = ghgh_2[i+p*ly][0:chunk]

##### y방향 inverse FFT의 계산 범위 분할 및 FFT
G_local_y = np.fft.ifft2(ghgh[jsta:jend+1,:], axes = [1]).copy()
```

```
##### 계산된 범위 취합
comm.Allgather(G_local_y, ghgh)

if rank == 0 :

    fig = plt.figure(1, figsize = (10, 7))
    plt.subplot(121)
    plt.imshow(imgr, cmap = 'gray')
    plt.axis('off')
    plt.text(430,40,r'$(a)$', color = 'k')

    plt.subplot(122)
    plt.imshow(np.abs(ghgh), cmap = 'gray')
    plt.axis('off')
    plt.text(430,40,r'$(b)$', color = 'w')

    plt.tight_layout()
    plt.savefig('examples/FFT2D_3.png')
    plt.show()
```

In [ ]:
```
!mpiexec -np 2 python examples/FFT2D_3.py
```

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('examples/FFT2D_3.png')
plt.imshow(img)
plt.show()
```

In [ ]:
```

```