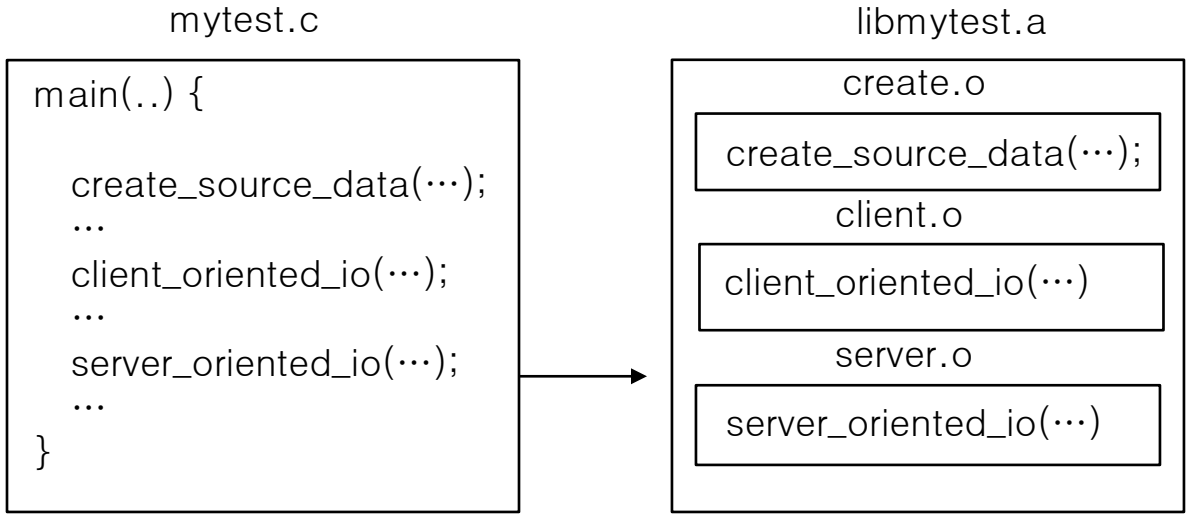
Term Project(1)

- 계산 및 I/O 속도를 높이기 위한 병렬작업(parallel operation)을 구현하기 위한 목적임
- Static library를 사용해서 구현할 것 (template 참조, 시간 계산을 위한 compiler directive 사용)
- 4 processes, 4개의 node file을 사용하며, 전체 data크기는 4MB임 (1M of integer)
- 전체 데이터가 (*, cyclic)으로 분산되어 있으며, 이에 대한 client_oriented_io, server_oriented_io를 구현 (io는 write임)
- 두 기법의 통신 기법이 달라야 함 (pipe, fifo, message queue, shared memory, socke중 두 개를 선택)
- 구현 및 점검을 쉽게 하도록 A[k] = k로 함

Term Project Template




```
$gcc –c create.c
$gcc –c –DTIMES client.c
$gcc –c –DTIMES server.c
$ar rc libmytest.a create.o  // option r로 object.o를 삽입하거나 치환, c로 libmytest.a를 생성
$ar rc libmytest.a client.o
$ar rc libmytest.a server.o   // $ar tv libmytest.a 로 모든 object file이 삽입되었는지 확인
$gcc –o mytest mytest.c –L./ –lmytest

/* client.o 수정 시
$gcc –c–DTIMES client.c
$ar rv libmytest.a client.o
$gcc –o mytest mytest.c –L./ –lmytest.o

/* ar dv 필요시 해당 object 삭제
```

Term Project(2)

- fork를 통해 compute node와 I/O node를 구성할 것
- compute node 간의 데이터 분산을 (*, cyclic) 으로 구성할 것
- (example) 64integer, 4procs, 4 io node

2D input data to be distributed on 4 processes

P1	P2	P3	P4	P1	P2	P3	P4
D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)	D(8)
D(9)	D(10)	D(11)	D(12)	D(13)	D(14)	D(15)	D(16)
D(17)	D(18)	D(19)	D(20)	D(21)	D(22)	D(23)	D(24)
D(25)	D(26)	D(27)	D(28)	D(29)	D(30)	D(31)	D(32)
D(33)	D(34)	D(35)	D(36)	D(37)	D(38)	D(39)	D(40)
D(41)	D(42)	D(43)	D(44)	D(45)	D(46)	D(47)	D(48)
D(49)	D(50)	D(51)	D(52)	D(53)	D(54)	D(55)	D(56)
D(57)	D(58)	D(59)	D(60)	D(61)	D(62)	D(63)	D(64)

Term Project(3)

- 2D 입력데이터는 아래와 같이 메모리에서 4개의 procs상에 분산됨
- 각 분산된 데이터를 파일로 저장해서 데모 시 화면 출력을 해야 함
  ($od -i p1.dat | more)

P1 분산데이터

D(1)	D(5)	D(9)	D(13)	D(17)	D(21)	D(25)	D(29)
D(33)	D(37)	D(41)	D(45)	D(49)	D(53)	D(57)	D(61)

P2 분산데이터

D(2)	D(6)	D(10)	D(14)	D(18)	D(22)	D(26)	D(30)
D(34)	D(38)	D(42)	D(46)	D(50)	D(54)	D(58)	D(62)

P3 분산데이터

D(3)	D(7)	D(11)	D(15)	D(19)	D(23)	D(27)	D(31)
D(35)	D(39)	D(43)	D(47)	D(51)	D(55)	D(59)	D(63)

P4 분산데이터

D(4)	D(8)	D(12)	D(16)	D(20)	D(24)	D(28)	D(32)
D(36)	D(40)	D(44)	D(48)	D(52)	D(56)	D(60)	D(64)

Term Project(4)

- 두 I/O 기법으로 4개의 node file에 연속적으로 저장하며, 두 기법의 통신이 반드시 달라야 함
- 각 node file에 저장되는 io chunk 크기는 8integer (32B)
- 각 node file의 내용을 데모 시 화면 출력을 해야 함

I/O node #1

D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)	D(8)	chunk 1
D(33)	D(34)	D(35)	D(36)	D(37)	D(38)	D(39)	D(40)	chunk 5

I/O node #2

D(9)	D(10)	D(11)	D(12)	D(13)	D(14)	D(15)	D(16)	chunk 2
D(41)	D(42)	D(43)	D(44)	D(45)	D(46)	D(47)	D(48)	chunk 6

I/O node #3

D(17)	D(18)	D(19)	D(20)	D(21)	D(22)	D(23)	D(24)	chunk 3
D(49)	D(50)	D(51)	D(52)	D(53)	D(54)	D(55)	D(56)	chunk 7

I/O node $4

D(25)	D(26)	D(27)	D(28)	D(29)	D(30)	D(31)	D(32)	chunk 4
D(57)	D(58)	D(59)	D(60)	D(61)	D(62)	D(63)	D(64)	chunk 8

Term Project(5)

- Compute node에서 연산된 데이터들을 I/O node에서 모으는 두 가지 병렬 I/O (parallel I/O) 방식을 각각 다른 통신 수단을 사용하여 실험하고 성능을 비교함
  - Client-oriented collective I/O
    - 각 client(compute node)가 주어진 영역의 데이터들을 다른 client로 부터 모아서 연속적인 데이터들을 보내는 방법
  - Server-oriented collective I/O
    - 각 server(I/O node)가 주어진 영역의 데이터들을 client로 부터 모아서 저장하는 방법

Term Project(6)

Client-oriented collective I/O

- 각 client의 I/O domain = 데이터 크기/client 수 임.
- 각 client는 주어진 영역의 데이터를 받아서 해당 server에게 전달. I/O node는 받은 데이터를 저장함
- Example)

  P1 : D(1)~D(16)

  : D(1)는 보유, D(2)은 P2, D(3)는 P3, D(4)은 P4으로 부터 받음

  : 모아진 데이터를 chunk 1은 I/O node #1, chunk 2는 I/O node #2에 전달해서 저장함

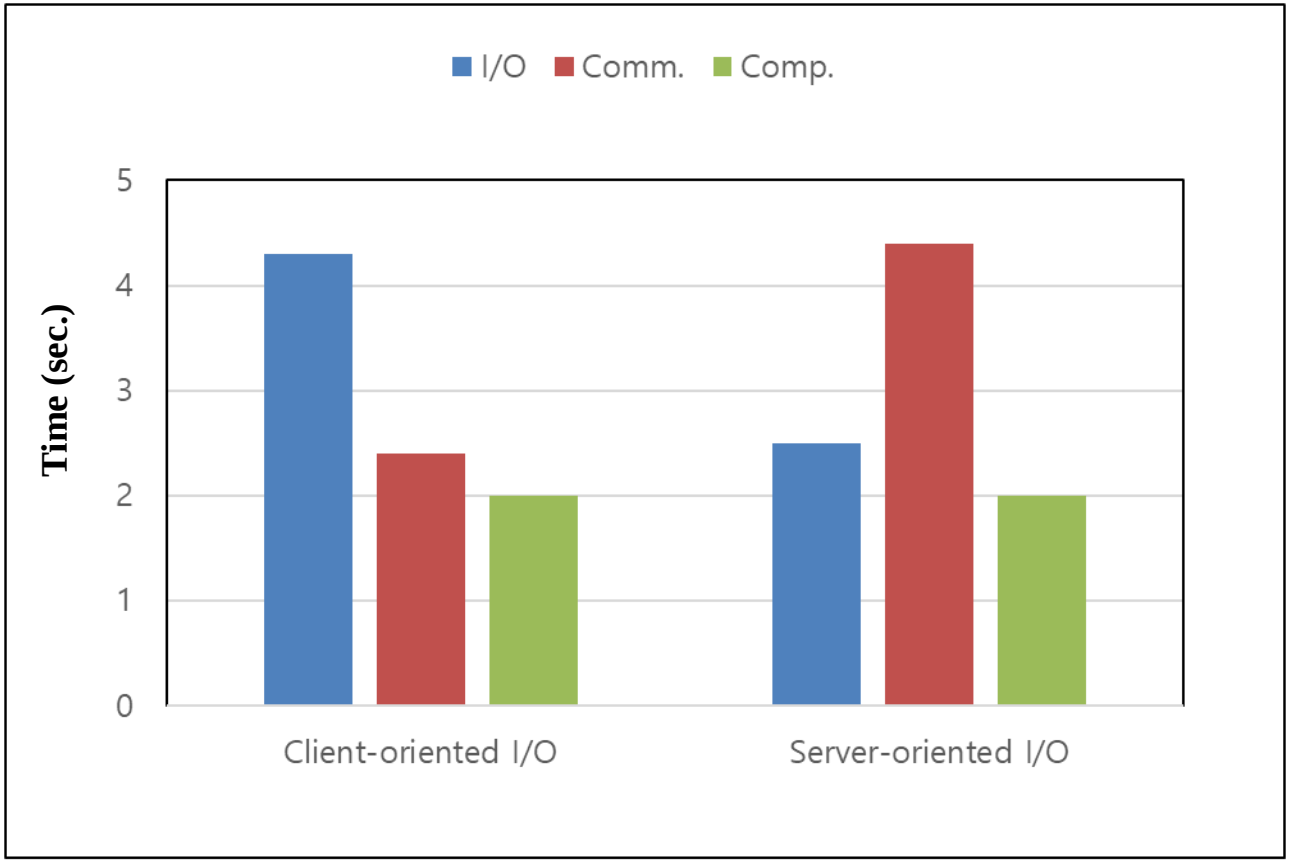
Server-oriented collective /O

- 각 server의 I/O domain = 데이터 크기/server 수 임.
- 각 server는 주어진 영역의 데이터를 client로 부터 받아서 모아서 저장함
- Example)

  I/O node #1 : chunk 1, chunk 5

  : 모아진 데이터를 한번에 저장함

Term Project(7)



Term Project(8)

추가사항

- 각 I/O node에 분산되는 chunk 크기는 4KB (1K of integer)
- 각 I/O node와 compute node에 분산된 data를 파일로 저장할 것
- Demo
  - 각각을 구현한 방법을 설명할 것
  - 각 compute node에 분산된 data를 od command로 dump 시킬 것
  - 실행
  - 각 I/O node에 분산된 data를 od command로 dump 시킬 것
  - 결과 graph및 결론
  - 팀 별 10분씩 발표하고 ppt 제출
  - 12/25(월)이 공휴일이므로 데모는 12/18(월)과 12/20(수)로 나누어서 수행함