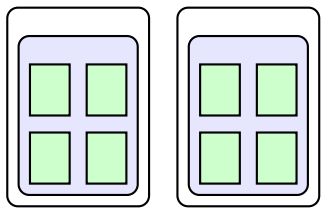
-
-
  -
  -
  -
-
  -
  -
  -
  -
  -

Unil.

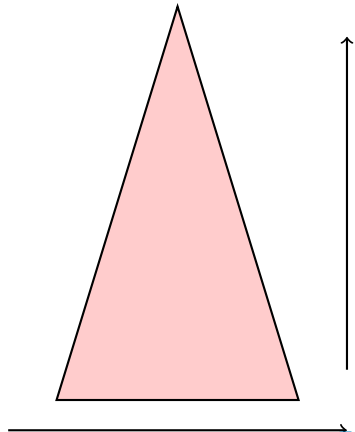
-
-
-
  -
  -
  -

Unil.

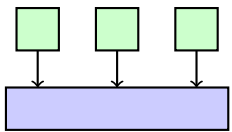
-
-
-
-



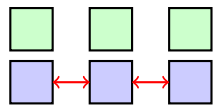
Unil.



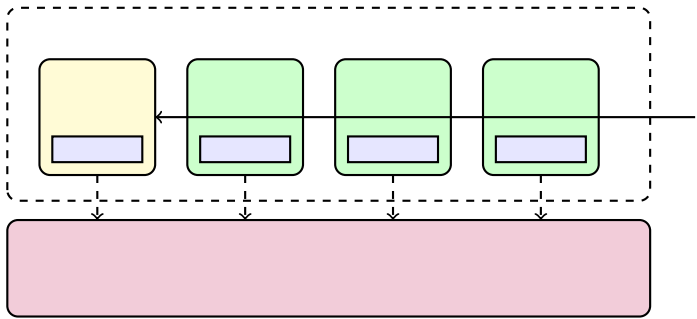
-
-
-
-

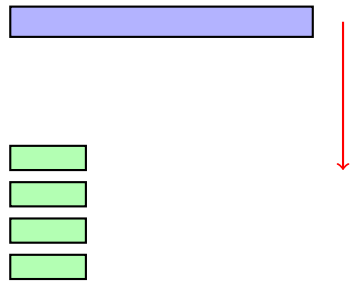


-
-
-
-



Unil.





Unil.

-
-
-

-
-
-

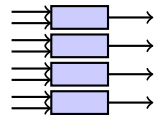
Unil.

-
-
-

-
-
-

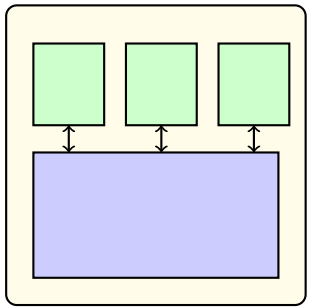
Unil.

$$C[0:3] = A[0:3] + B[0:3]$$



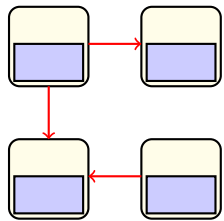
-
-
-
-

```
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
}
```



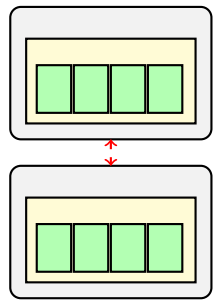
Unil.

-
-
-
-



- `MPI_Send`
- `MPI_Recv`
- `MPI_Bcast`
- `MPI_Reduce`

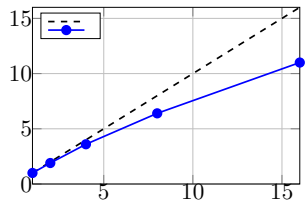
Unil.



Unil.

-
-
$$S = \frac{T}{T}$$
-
-
$$E = \frac{S}{N} = \frac{T}{N \times T}$$
- $E = 1$
- $E = 0.7 - 0.9$

-
-
- $S = 100/30 = 3.33$
- $E = 3.33/4 = 0.83$



Unil.

-
-
-

T

p

$(1-p)$

N

- $(1-p) \times T$
- $\frac{p \times T}{N}$
- $(1-p) \times T + \frac{p \times T}{N}$

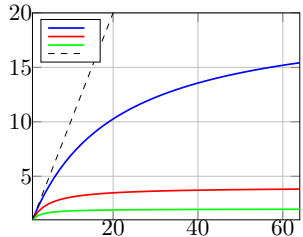
$$S(N) = \frac{T}{(1-p) \times T + \frac{p \times T}{N}}$$

T

$$S(N) = \frac{1}{(1-p) + \frac{p}{N}}$$

Unil.

$$S(n) = \frac{1}{(1-p) + \frac{p}{n}}$$



- $S(n)n$
- p
- $(1-p)$
-
- $n \to \infty S \to \frac{1}{1-p}$
-

Unil.

Unil.

```
sbatch script.sh
squeue
squeue -u $USER
scancel <jobid>
sinfo
srun command
scontrol show job <jobid>
sacct
salloc
```

- `-l`
- `--start`
- `-t RUNNING`

Unil.

```
#!/bin/bash
#SBATCH --job-name=myjob
#SBATCH --output=output_%j.txt
#SBATCH --error=error_%j.txt
#SBATCH --time=01:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=8G

# Load modules
module load python/3.9

# Set environment
export OMP_NUM_THREADS=4

# Run application
python my_script.py
```

- `--job-name`
- `--output/--error`
- `--time`
- `--nodes`
- `--ntasks`
- `--cpus-per-task`
- `--mem`
- `--mem-per-cpu`

Unil.

```
#!/bin/bash
#SBATCH --job-name=serial_job
#SBATCH --output=serial_%j.log
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=2G
#SBATCH --time=00:30:00

echo "Running on host: $(hostname)"
echo "Job ID: $SLURM_JOB_ID"
echo "Number of cores: $SLURM_CPUS_PER_TASK"

python my_serial_script.py
```

Unil.

```
#!/bin/bash
#SBATCH --job-name=openmp_job
#SBATCH --output=openmp_%j.log
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --mem=32G
#SBATCH --time=02:00:00

module load gcc/11.2.0

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
echo "Using $OMP_NUM_THREADS OpenMP threads"

./my_openmp_program
```

Unil.

```
#!/bin/bash
#SBATCH --job-name=mpi_job
#SBATCH --output=mpi_%j.log
#SBATCH --nodes=4
#SBATCH --ntasks=64
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=2G
#SBATCH --time=04:00:00

module load openmpi/4.1.1

echo "Running on $SLURM_NNODES nodes"
echo "Total tasks: $SLURM_NTASKS"

mpirun ./my_mpi_program
```

```
#!/bin/bash
#SBATCH --job-name=hybrid_job
#SBATCH --output=hybrid_%j.log
#SBATCH --nodes=4
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=8
#SBATCH --mem=256G
#SBATCH --time=06:00:00

module load gcc/11.2.0 openmpi/4.1.1

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
echo "MPI tasks: $SLURM_NTASKS"
echo "OpenMP threads per task: $OMP_NUM_THREADS"

mpirun ./my_hybrid_program
```

Unil.

```
--ntasks=1
--cpus-per-task=1
--ntasks=1
--cpus-per-task=N
--ntasks=N
--cpus-per-task=1
--ntasks=M
--cpus-per-task=N
```

-
-
-

Unil.

```
#!/bin/bash
#SBATCH --job-name=array_job
#SBATCH --output=job_%A_%a.log
#SBATCH --array=1-100
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=2G
#SBATCH --time=01:00:00

# SLURM_ARRAY_TASK_ID contains the current array index (1-100)
PARAM=$SLURM_ARRAY_TASK_ID
echo "Processing parameter: $PARAM"
python process.py --id $PARAM
```

Unil.

- %A
- %a
- $SLURM_ARRAY_TASK_ID

- --array=1-100
- --array=1-100:10
- --array=1,5,10,15
- --array=1-100%10

-
-
-

Unil.

```
# Interactive single command
srun --ntasks=1 \
     --cpus-per-task=4 \
     --mem=8G \
     --time=01:00:00 \
     python my_script.py
```

-
-
-
-

Unil.

```
# Allocate resources
salloc --nodes=1 \
       --ntasks=4 \
       --mem=16G \
       --time=02:00:00

# Now on compute node, run multiple commands
hostname
module load python
python  # Interactive Python session
# ... work interactively ...
exit  # Release resources when done
```

-
-

Unil.

threadingmultiprocessingjoblib

mpi4py

numpynumba

Threads.@threads@spawn

MPI.jl

Distributed.jl@distributed

parallelforeach

RmpipbdMPI

```python
from concurrent.futures import ThreadPoolExecutor
import numpy as np

def compute(x):
    # Good for I/O-bound tasks
    return np.sqrt(x)

with ThreadPoolExecutor(max_workers=4) as ex:
    results = ex.map(compute, range(100))
```

```python
from multiprocessing import Pool

def compute(x):
    # Good for CPU-bound tasks
    return x ** 2

with Pool(processes=4) as pool:
    results = pool.map(compute, range(100))
```

-
-

```
# For multiprocessing
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4

# In Python script
import os
from multiprocessing import cpu_count
n_cores = int(os.getenv(
    'SLURM_CPUS_PER_TASK',
    cpu_count()))
```

Unil.

```python
from mpi4py import MPI
import numpy as np

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

# Each process computes part
n_local = 1000
data = np.random.rand(n_local)
local_sum = np.sum(data)

# Reduce to get total sum
total_sum = comm.reduce(
    local_sum,
    op=MPI.SUM,
    root=0)

if rank == 0:
    print(f"Total: {total_sum}")
```

```
#!/bin/bash
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=2G
#SBATCH --time=01:00:00

module load openmpi python/3.9

mpirun python my_mpi_script.py
```

- mpirun
-
-
-

Unil.

```julia
using Base.Threads

# Set threads with environment variable
# export JULIA_NUM_THREADS=4

function parallel_sum(arr)
    result = zeros(nthreads())
    @threads for i in eachindex(arr)
        tid = threadid()
        result[tid] += arr[i]
    end
    return sum(result)
end

# Test
arr = rand(1000000)
total = parallel_sum(arr)
```

```bash
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem=16G
#SBATCH --time=02:00:00

module load julia/1.8

# For threading
export JULIA_NUM_THREADS=$SLURM_CPUS_PER_TASK

julia my_script.jl
```

-
- `@threads`
-

Unil.

```julia
using Distributed

# Add 4 worker processes
addprocs(4)

# Function available on all workers
@everywhere function compute(x)
    return x^2
end

# Parallel reduction
results = @distributed (+) for i = 1:100
    compute(i)
end

println("Sum of squares: $results")
```

```julia
using MPI

MPI.Init()
comm = MPI.COMM_WORLD
rank = MPI.Comm_rank(comm)
size = MPI.Comm_size(comm)

# Each rank does work
local_result = rank * 10

# Gather results
all_results = MPI.Gather(
    local_result, 0, comm)

if rank == 0
    println("Results: $all_results")
end
```

Unil.

```r
library(foreach)
library(doParallel)

# Register parallel backend
cl <- makeCluster(4)
registerDoParallel(cl)

# Parallel loop
results <- foreach(
    i=1:100,
    .combine=c) %dopar% {
    # Computation
    i^2
}

stopCluster(cl)
```

```r
library(parallel)

# Detect cores
ncores <- detectCores()

# Parallel apply
results <- mclapply(
    1:100,
    function(x) x^2,
    mc.cores = ncores)

# Convert to vector
results <- unlist(results)
```

mclapplyforeach

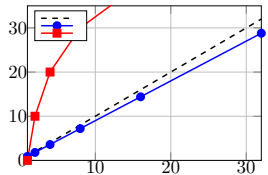
Unil.

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem=16G
#SBATCH --time=01:00:00

module load r/4.2

# Set cores from SLURM
export R_NCORES=$SLURM_CPUS_PER_TASK

Rscript my_script.R
```

```
# Get cores from environment
ncores <- as.integer(
    Sys.getenv("R_NCORES", "1"))

# Use with foreach
cl <- makeCluster(ncores)
registerDoParallel(cl)

# ... rest of code ...

stopCluster(cl)
```

-
-
- .combine

Unil.


30
20
10
0
10
20
30

-
-
-

`sbatch`

`squeue`

Unil.

- $\sum_{i=1}^{N} i^2$
-
-

-
-

Unil.

-
-
-
-
-

Unil.

```
sinfo
sacct--mem--mem-per-cpu
--time
scontrol show partition
```

-
- `sstat`
- `sacct`

Unil.

`module availmodule load`

`$SLURM_SUBMIT_DIR`

-
- `echo`

Unil.

- `stderr`
-
-

- `sallocsrun`
-
-

-
-
-

Unil.

- `/usr/bin/time -v`
-
-
- $= \times \times 1.3$
-
-
-
- $T \approx \frac{T}{N \times}$
-

Unil.

-
-
- %j
-
-
-
-
-
-
- squeue -u $USER
- sacct
- sstat
-
-
-
-
-

Unil.

- https://slurm.schedmd.com/
-
- man sbatchman srun

- https://www.hpc-carpentry.org/
- https://www.xsede.org/for-users/training
- https://www.openmp.org/resources/tutorials-articles/
- https://mpitutorial.com/

- gprofperfIntel VTune