Task Fusion in Distributed Runtimes

Shiv Sundram
Wonchan Lee
Alex Aiken

Task Based Programming Models

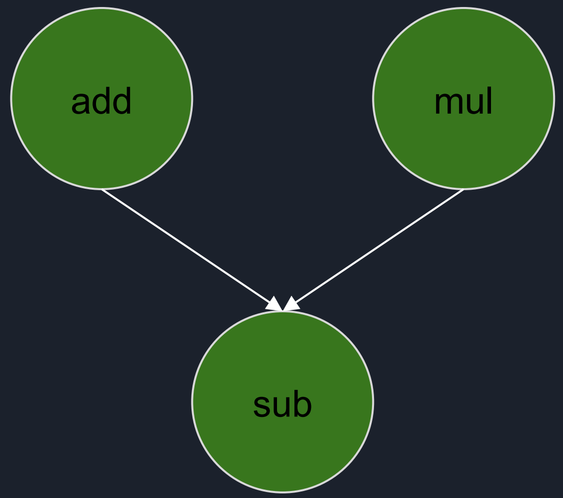
Legion

PaRSEC

dask

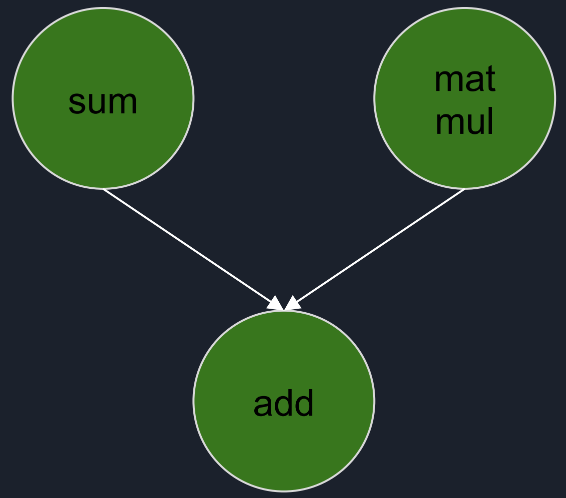
StarPU

cuNumeric



Task

- Functions
- Communicates with other tasks solely via inputs and output
- Can be sharded across processors for data parallel operations

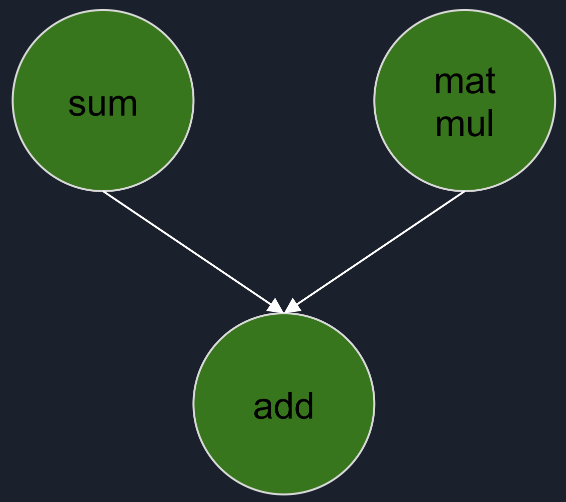


Automation in Task Based Runtimes

- Scheduling
- Resource allocation
- Memory movement
- Synchronization

Automation → Overheads

Problematic when tasks are small



Goal: Eliminate Overheads

Dynamic DAG: tasks & task sizes not known ahead of time
Small tasks: overheads consume 40% of runtime in real code

Best case scenario:

- Eliminate overheads
- Correctness: don’t introduce bugs/race conditions
- Dynamic: optimization should occur at runtime
- Zero changes to user/application code

CuNumeric

```
def black_scholes(S, X, T, R, V):
    sqrt_t = np.sqrt(T)
    d1 = np.log(S / X) + (R + 0.5 * V * V) * T / (V * sqrt_t)
    d2 = d1 - V * sqrt_t
    cnd_d1 = cnd(d1)
    cnd_d2 = cnd(d2)
    exp_rt = np.exp(-R * T)
    call_result = S * cnd_d1 - X * exp_rt * cnd_d2
    put_result = X * exp_rt * (1.0 - cnd_d2) - S * (1.0 - cnd_d1)
    return call_result, put_result
```

CuNumeric

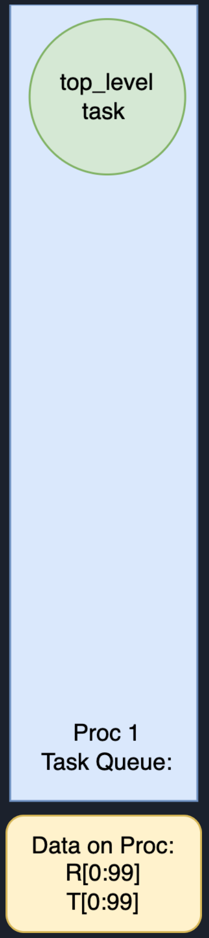
```python
import cunumeric as np
```

```python
def black_scholes(S, X, T, R, V):
    sqrt_t = np.sqrt(T)
    d1 = np.log(S / X) + (R + 0.5 * V * V) * T / (V * sqrt_t)
    d2 = d1 - V * sqrt_t
    cnd_d1 = cnd(d1)
    cnd_d2 = cnd(d2)
    exp_rt = np.exp(-R * T)
    call_result = S * cnd_d1 - X * exp_rt * cnd_d2
    put_result = X * exp_rt * (1.0 - cnd_d2) - S * (1.0 - cnd_d1)
    return call_result, put_result
```

CuNumeric & Task Launches

```python
import cunumeric as np
```

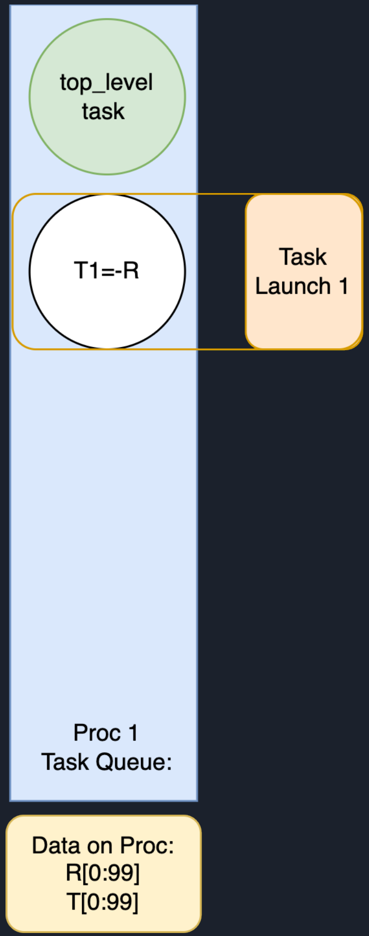
```python
def black_scholes(S, X, T, R, V):
    sqrt_t = np.sqrt(T)
    d1 = np.log(S / X) + (R + 0.5 * V * V) * T / (V * sqrt_t)
    d2 = d1 - V * sqrt_t
    cnd_d1 = cnd(d1)
    cnd_d2 = cnd(d2)
    exp_rt = np.exp(-R * T)
    call_result = S * cnd_d1 - X * exp_rt * cnd_d2
    put_result = X * exp_rt * (1.0 - cnd_d2) - S * (1.0 - cnd_d1)
    return call_result, put_result
```



CuNumeric & Task Launches

```python
import cunumeric as np
```

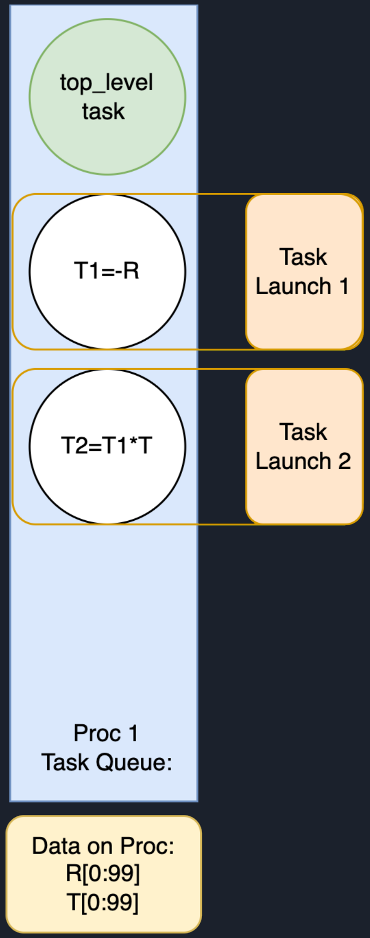
```python
def black_scholes(S, X, T, R, V):
    sqrt_t = np.sqrt(T)
    d1 = np.log(S / X) + (R + 0.5 * V * V) * T / (V * sqrt_t)
    d2 = d1 - V * sqrt_t
    cnd_d1 = cnd(d1)
    cnd_d2 = cnd(d2)
    exp_rt = np.exp(-R * T)
    call_result = S * cnd_d1 - X * exp_rt * cnd_d2
    put_result = X * exp_rt * (1.0 - cnd_d2) - S * (1.0 - cnd_d1)
    return call_result, put_result
```



CuNumeric & Task Launches

```python
import cunumeric as np
```

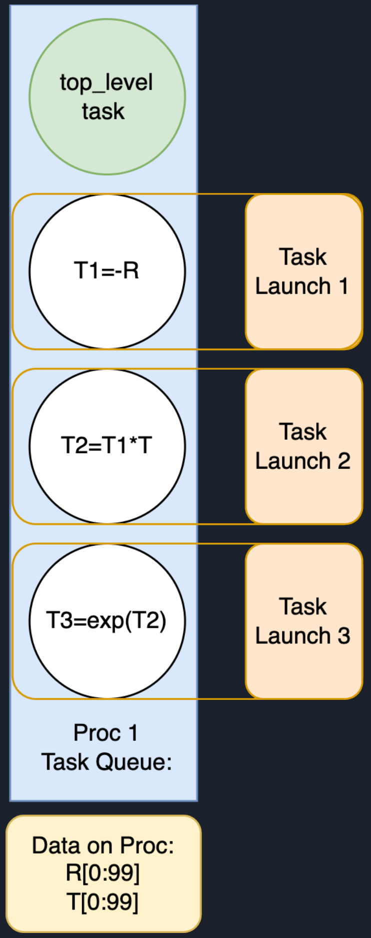
```python
def black_scholes(S, X, T, R, V):
    sqrt_t = np.sqrt(T)
    d1 = np.log(S / X) + (R + 0.5 * V * V) * T / (V * sqrt_t)
    d2 = d1 - V * sqrt_t
    cnd_d1 = cnd(d1)
    cnd_d2 = cnd(d2)
    exp_rt = np.exp(-R * T)
    call_result = S * cnd_d1 - X * exp_rt * cnd_d2
    put_result = X * exp_rt * (1.0 - cnd_d2) - S * (1.0 - cnd_d1)
    return call_result, put_result
```



CuNumeric & Task Launches

```
import cunumeric as np
```

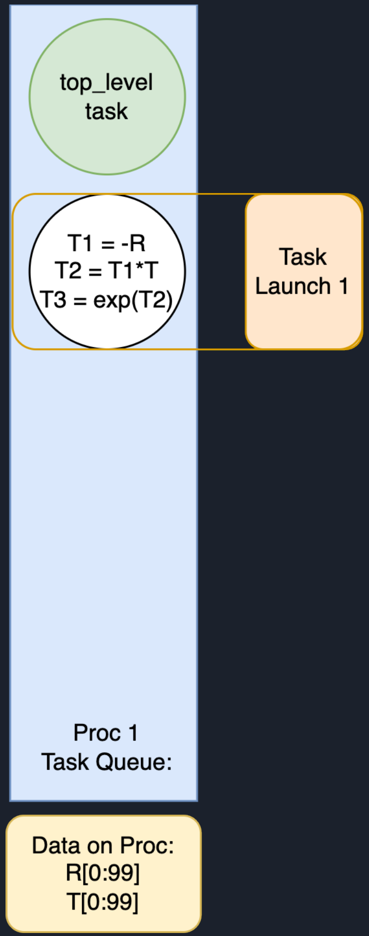
```
def black_scholes(S, X, T, R, V):
    sqrt_t = np.sqrt(T)
    d1 = np.log(S / X) + (R + 0.5 * V * V) * T / (V * sqrt_t)
    d2 = d1 - V * sqrt_t
    cnd_d1 = cnd(d1)
    cnd_d2 = cnd(d2)
    exp_rt = np.exp(-R * T)
    call_result = S * cnd_d1 - X * exp_rt * cnd_d2
    put_result = X * exp_rt * (1.0 - cnd_d2) - S * (1.0 - cnd_d1)
    return call_result, put_result
```



CuNumeric & Task Launches - Fusion

```python
import cunumeric as np
```

```python
def black_scholes(S, X, T, R, V):
    sqrt_t = np.sqrt(T)
    d1 = np.log(S / X) + (R + 0.5 * V * V) * T / (V * sqrt_t)
    d2 = d1 - V * sqrt_t
    cnd_d1 = cnd(d1)
    cnd_d2 = cnd(d2)
    exp_rt = np.exp(-R * T)
    call_result = S * cnd_d1 - X * exp_rt * cnd_d2
    put_result = X * exp_rt * (1.0 - cnd_d2) - S * (1.0 - cnd_d1)
    return call_result, put_result
```

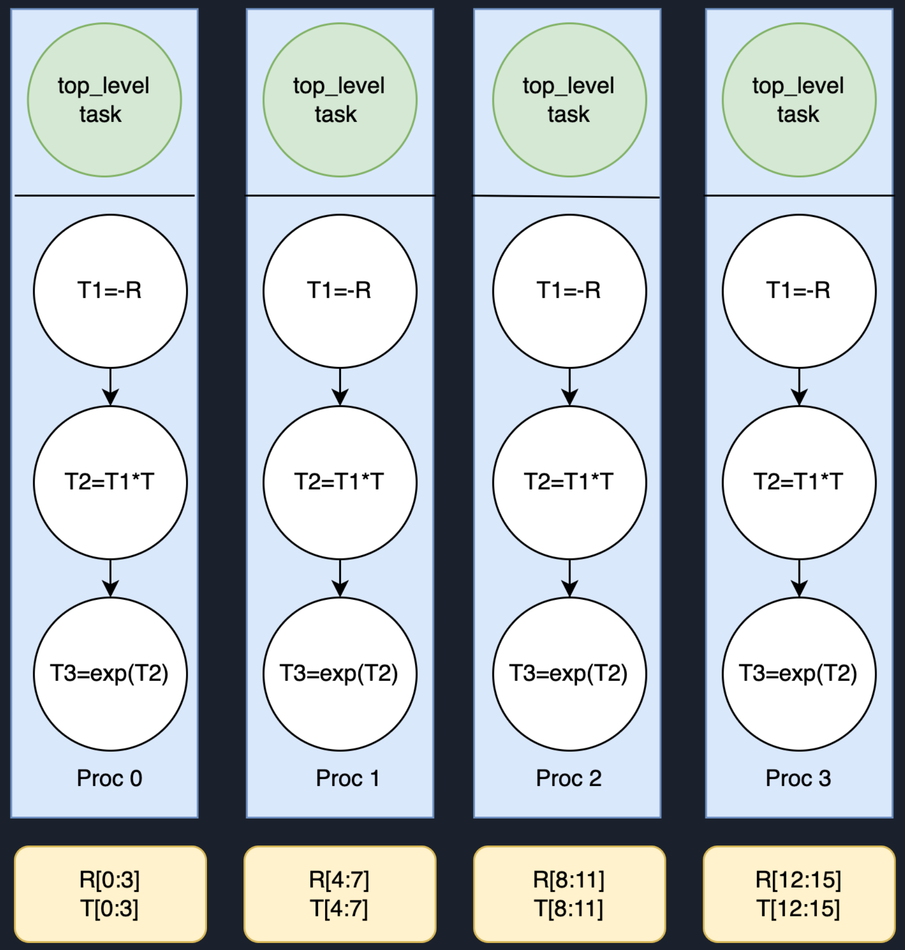


Parallel Tasks

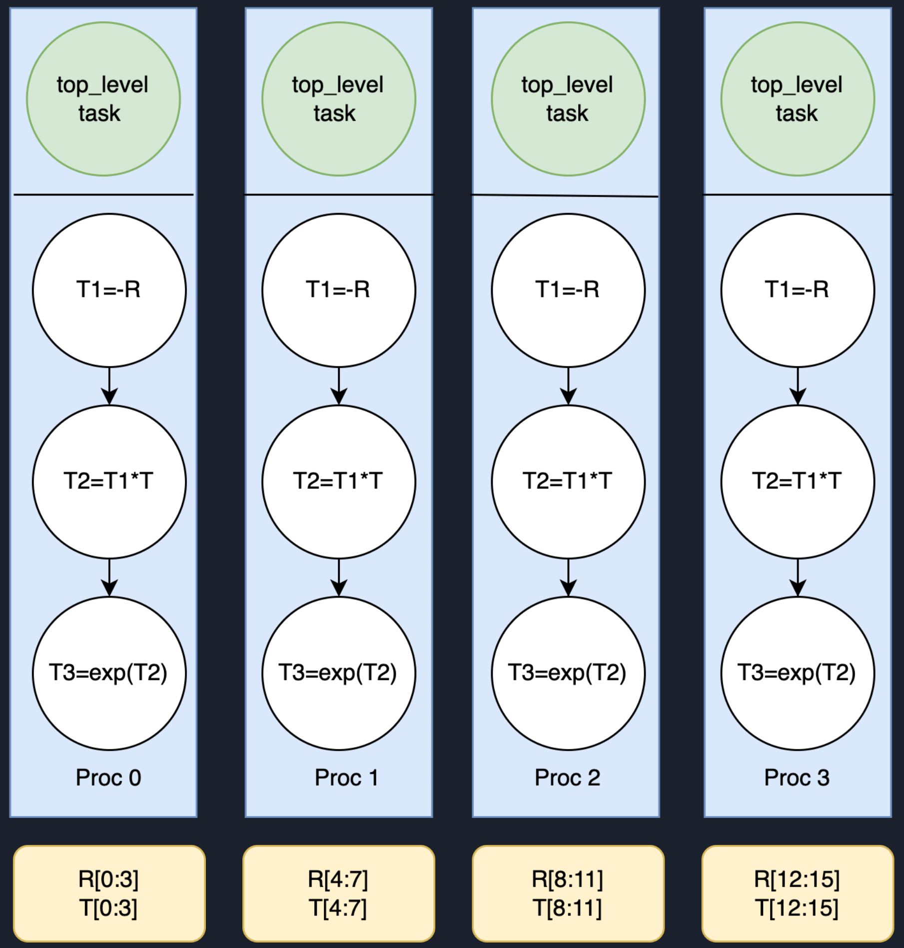
Each task: sharded across processors

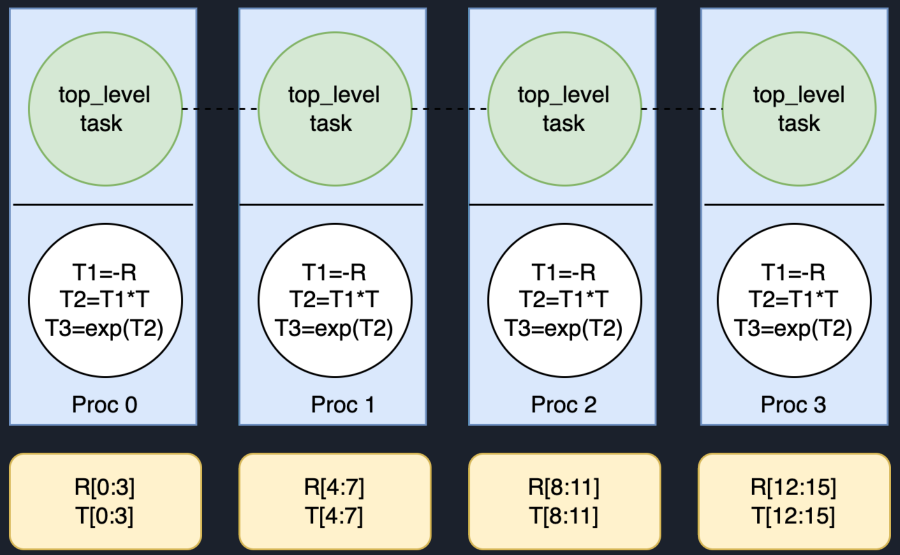
Data partitioned amongst processors

Computation sharded amongst processors



Parallel Tasks





Desired Task Fusion

Want task fusion to:

- Eliminate overheads of runtime via fusion
- Handle dynamic DAGs
- No changes to user/application code
- Correctness: Do not fuse if this resulting execution is unsafe

Stencil

```
input    = np.array(N=18)
central = input[1 : -1]
west     = input[0 : -2]
east      = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

input																	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Stencil

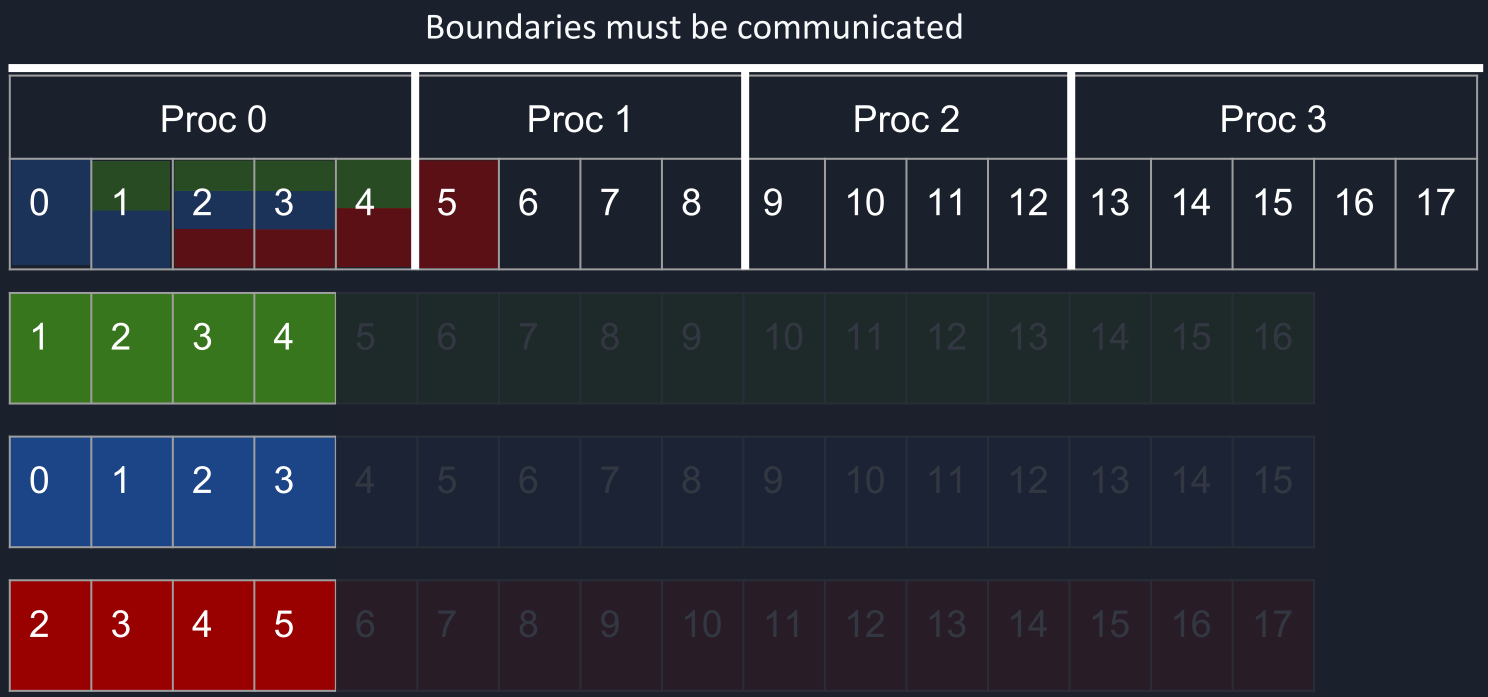
```
input    = np.array(N=18)
central = input[1 : -1]
west      = input[0 : -2]
east       = input[2 : N]
for i in range(num_iters):
    output      = east + west
    central[:] = output
```

input																	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Stencil

```
input    = np.array(N=18)
central = input[1 : -1]
west      = input[0 : -2]
east       = input[2 : N]
for i in range(num_iters):
    output      = east + west
    central[:] = output
```

```
input    = np.array(N=18)
central = input[1 : -1]
west     = input[0 : -2]
east      = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```



- Processor 0 contains input[0:5]
- Processor 0 requires input[0:6]
- Input [5] is communicated to Proc 0

Stencil

Stencil

```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

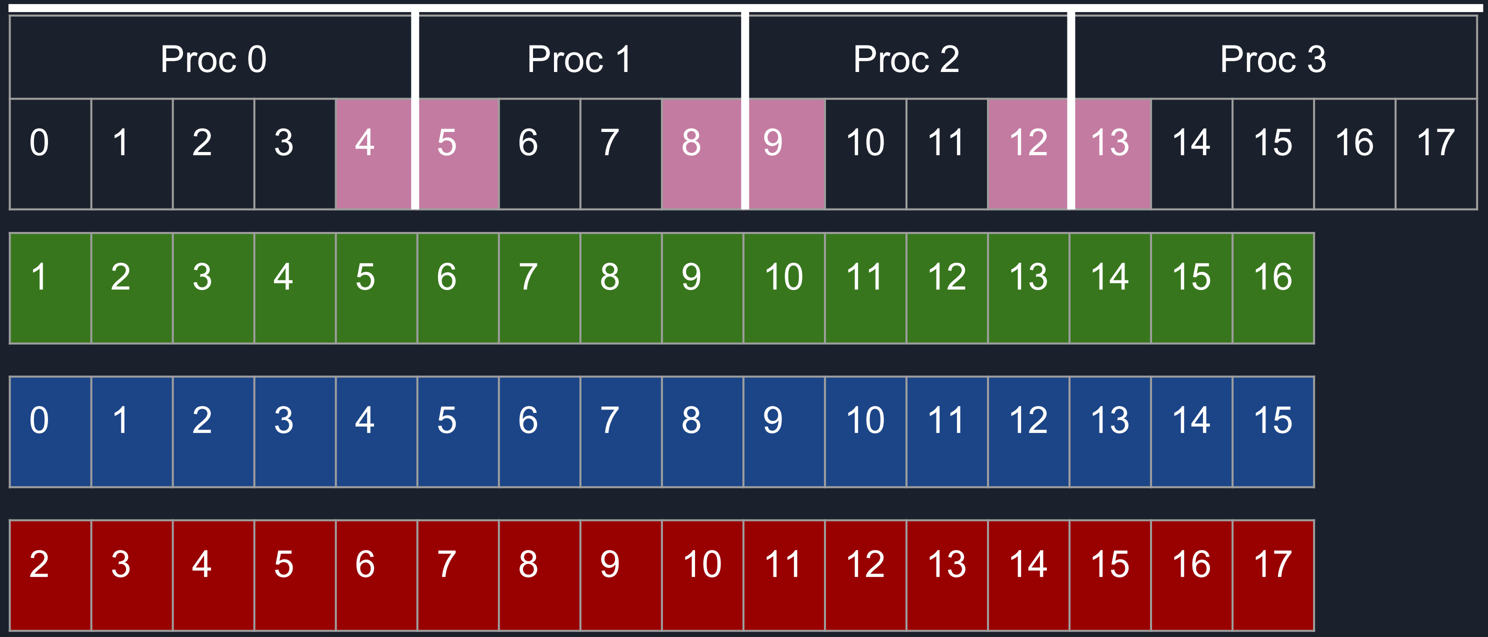
- Processor 0 contains input[0:5]
- Processor 0 requires input[0:6]
- Input [5] is communicated to Proc 0

Stencil

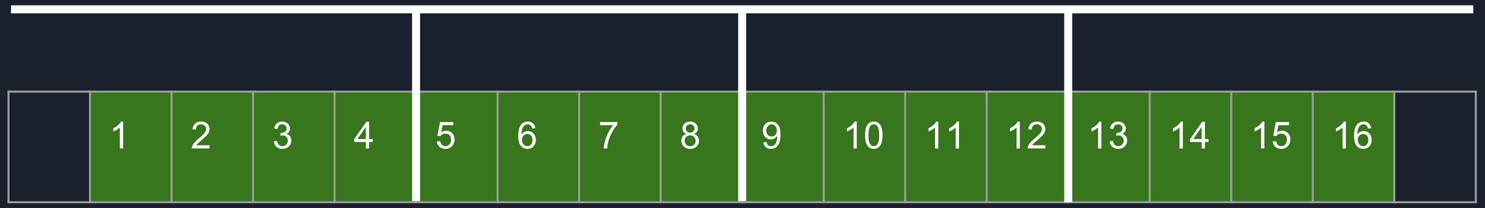
```
input    = np.array(N=18)
central = input[1 : -1]
west     = input[0 : -2]
east      = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

- Processor 0 contains input[0:5]
- Processor 0 requires input[0:6]
- Input [5] is communicated to Proc 0

Must communicate pink boundary cells



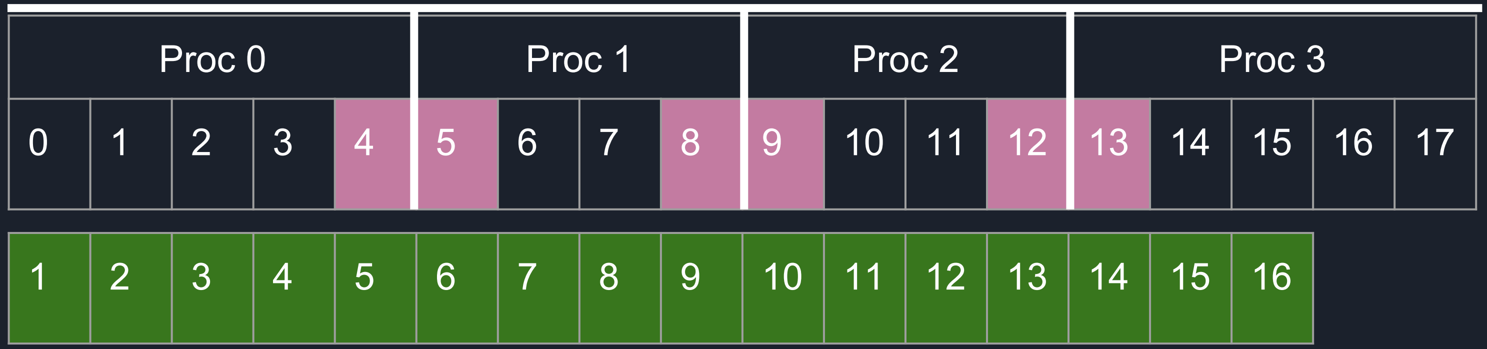
Computation Decomposition



Stencil

```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

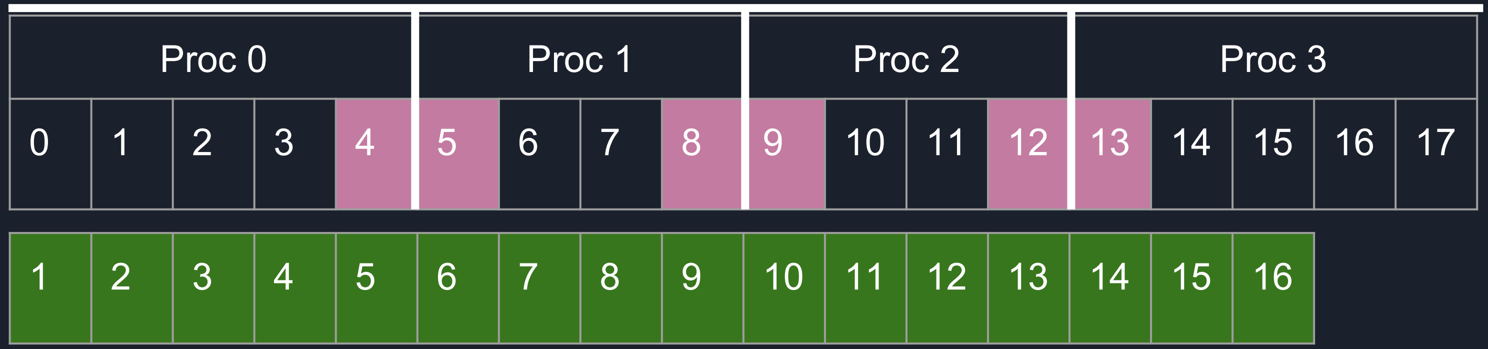
Must communicate pink boundary cells



Stencil

```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

Must communicate pink boundary cells



Task 1: ADD (east, west) → out

Task 2: WRITE (out) → central

Stencil

```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

Must communicate pink boundary cells

Proc 0					Proc 1				Proc 2				Proc 3				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

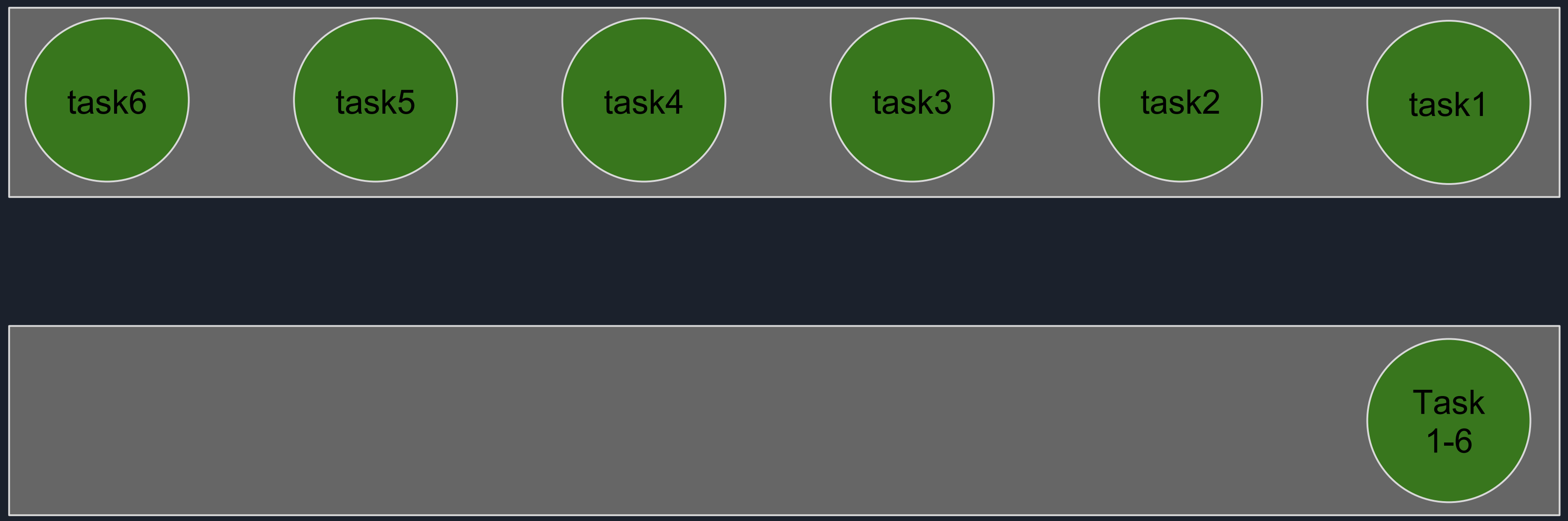
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

COMMUNICATE

Task 1: ADD (east, west) → out

Task 2: WRITE (out) → central

Task Execution

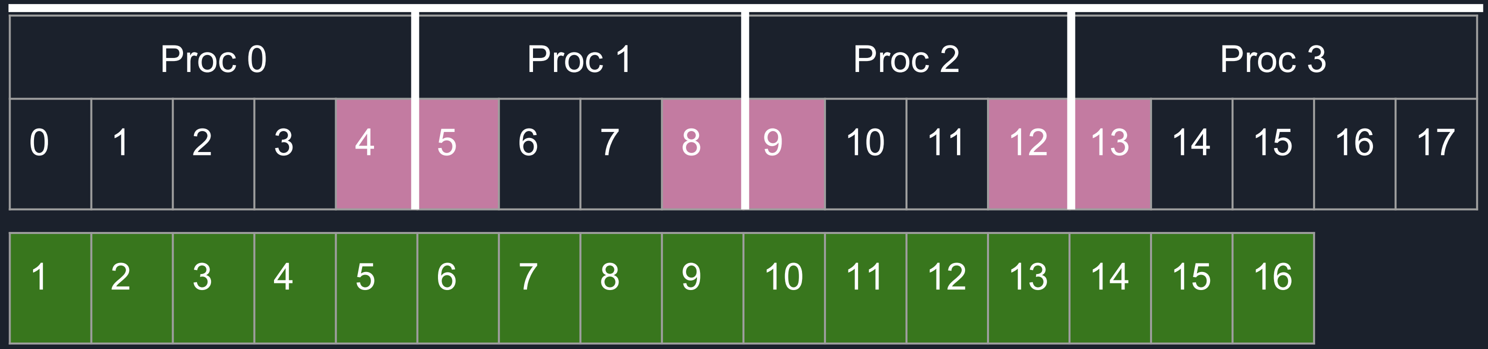


Stencil

```
input    = np.array(N=18)
central = input[1 : -1]
west      = input[0 : -2]
east       = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

- Cannot introduce inter-task synchronization within a task

Must communicate pink boundary cells



COMMUNICATE

Task 1: ADD (east, west) → out

Task 2: WRITE (out) → central

COMMUNICATE

Task 3: ADD (east, west) → out

Task 4: WRITE (out) → central

When is it legal to fuse?

Execution Model

Each task associated with

1. Shard on each processor
2. Launch Space : arrangement of processors (e.g. 2x2 grid of CPUs)
3. Data Partitions : partition array across processors
4. Projection Functions : subset of array needed by task

Execution Model

Each fused task:

1. A collection (>1) of subtasks
2. All subtasks' launch spaces, partitions, and projection functions

Execution Model

Each fused task:

1. A collection (>1) of subtasks
2. All subtasks' launch spaces, partitions, and projection functions

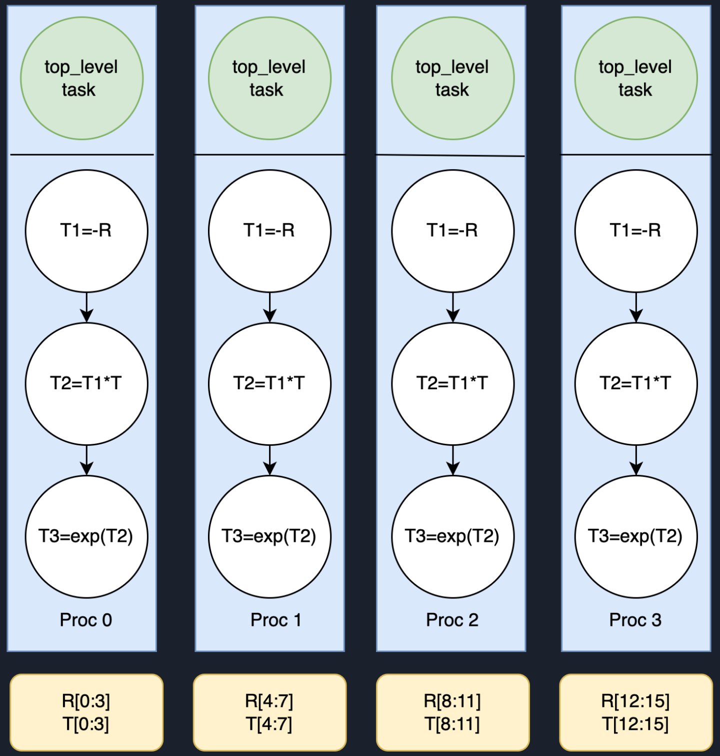
Define 4 fusion constraints that determine barrier placement

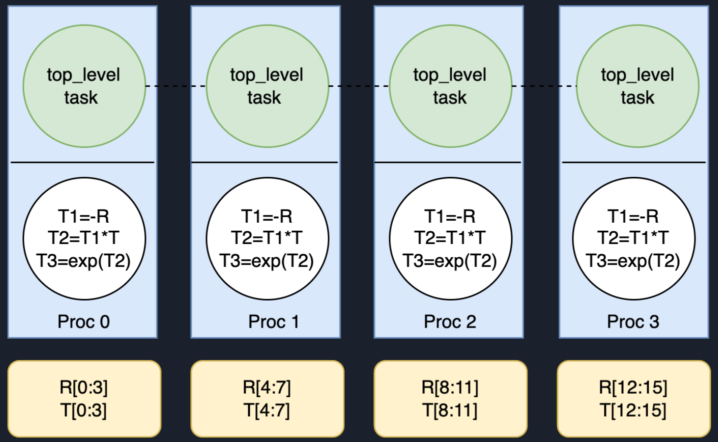
Constraint 1: Communication Absence

Tasks that, during execution, require synchronization with other tasks, cannot be fused

Example: array reductions → partial reductions communicated amongst procs

All ops embarrassingly data parallel → not an issue





Constraint 2: Launch Space Equivalence

A launch space : geometry of processor grid

Jacobi Iteration

```
d = np.diag(A)
R = A - np.diag(d)
for i in range(iters):
    x = (b - np.dot(R, x)) / d
```

Inner loop tasks:

```
temp1 = DOT(R, x, launch_space=(2,2))
temp2 = SUB(b, temp1,launch_space=(4,1))
x     = DIV(temp2, d, launch_space=(4,1))
```

1. Tasks target different launch shapes → cannot fuse

2. Dot product → requires reduction → inter task communication → cannot fuse

Projection

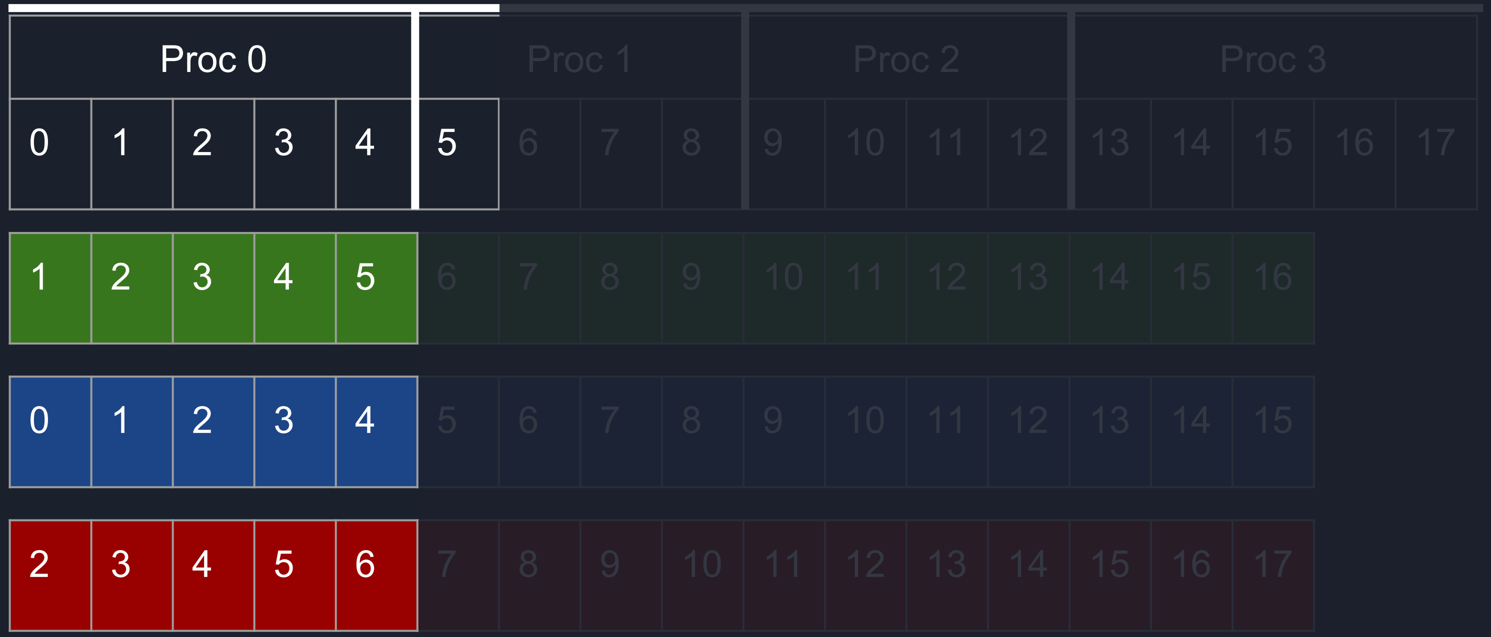
```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

Proc 0					Proc 1				Proc 2				Proc 3				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Projection

```
input    = np.array(N=18)
central = input[1 : -1]
west     = input[0 : -2]
east      = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```



Constraint 3: Projection (Domain Decomposition)

Task 1: divides up array view 1 way
 (according to what each task shard needs for the computation)

Proc 0				Proc 1				Proc 2				Proc 3					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Task 2: divides up same array view, but a different way

Proc 0					Proc 1				Proc 2				Proc 3				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Not allowed

Constraint 4: Producer Consumer Restriction

Proc 0				Proc 1				Proc 2				Proc 3					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

Constraint 4: Producer Consumer Restriction

Task 1: read from Blue (and Red)

Proc 0				Proc 1				Proc 2				Proc 3					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

Constraint 4: Producer Consumer Restriction

Task 1: read from Blue

Task 2: writes to Green (Produce)

Proc 0				Proc 1				Proc 2				Proc 3					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

Constraint 4: Producer Consumer Restriction

Task 1: read from Blue

Task 2: writes to Green (Produce)

Task 3: read from Blue (Consume)

Proc 0				Proc 1				Proc 2				Proc 3					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

Task A writes to one view of array

Updates not guaranteed to be seen when Task B reads from different view of same array

Constraint 4: Producer Consumer Restriction

Task 1: read from Blue

Task 2: writes to Green (Produce)

Task 3: read from Blue (Consume)

Proc 0				Proc 1				Proc 2				Proc 3					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

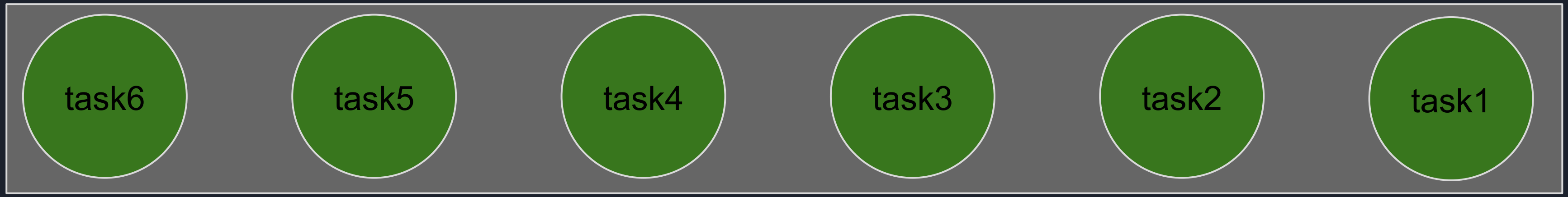
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

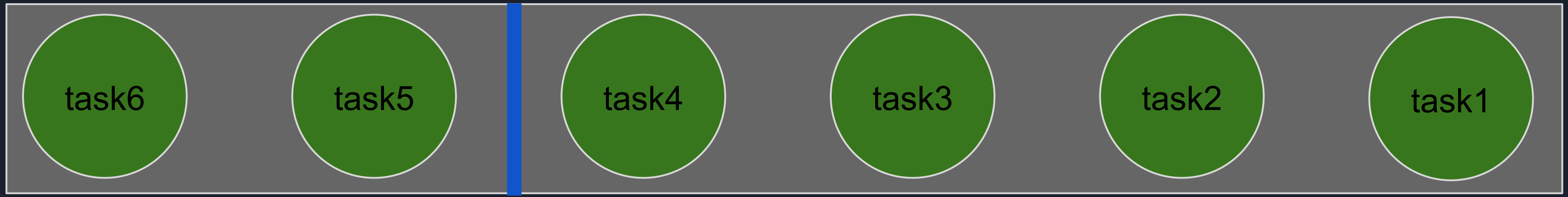
```
input   = np.array(N=18)
central = input[1 : -1]
west    = input[0 : -2]
east    = input[2 : N]
for i in range(num_iters):
    output     = east + west
    central[:] = output
```

Tasks 2-3 are not fusable.

Greedy Fusion



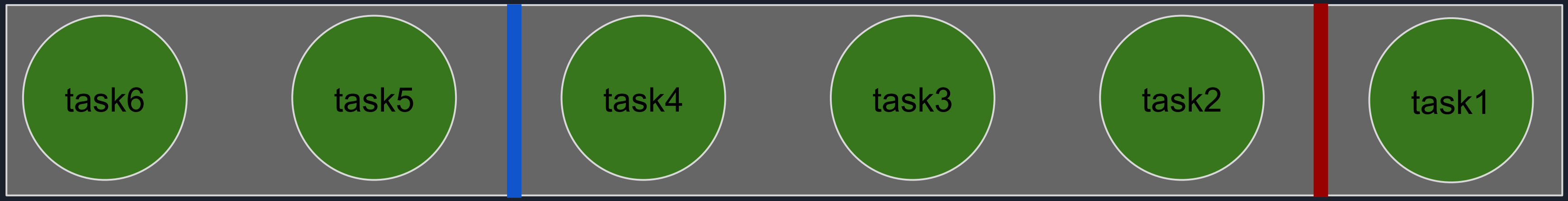
Greedy Fusion



Constraints

1. Communication Absence

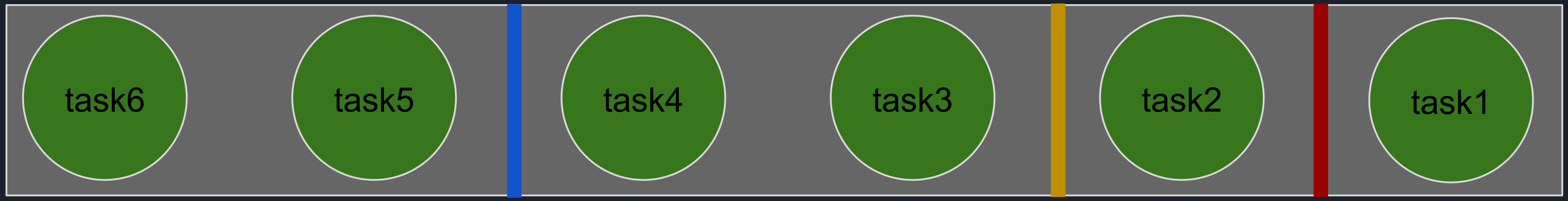
Greedy Fusion



Constraints

1. Communication Absence
2. Launch Space Equivalence

Greedy Fusion



Constraints

1. Communication Absence
2. Launch Space Equivalence
3. Projection (Domain Decomposition) Equivalence

Greedy Fusion

Constraints
1. Communication Absence
2. Launch Space Equivalence
3. Projection (Domain Decomposition) Equivalence
4. Producer Consumer Restrictions

Greedy Fusion

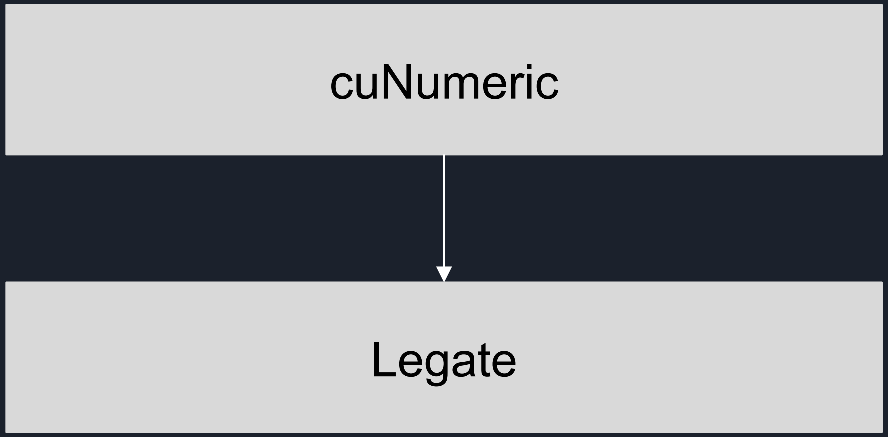
Constraints

1. Communication Absence
2. Launch Space Equivalence
3. Projection (Domain Decomposition) Equivalence
4. Producer Consumer Restrictions

Greedy Fusion

Constraints
1. Communication Absence
2. Launch Space Equivalence
3. Projection (Domain Decomposition) Equivalence
4. Producer Consumer Restrictions

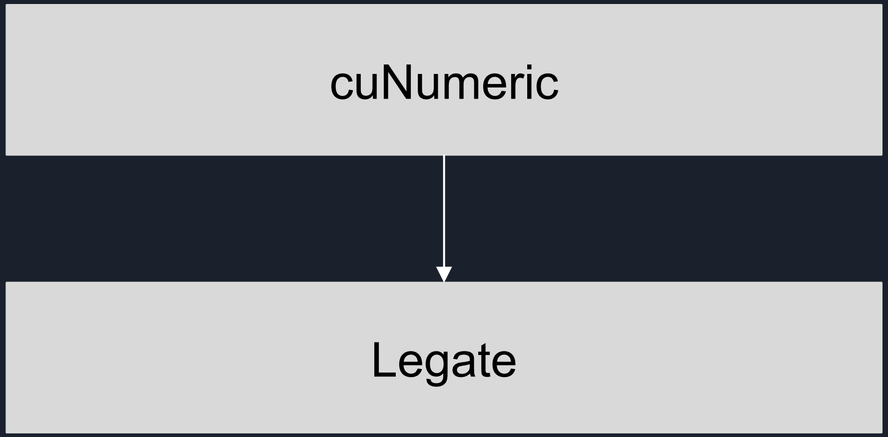
Fusion Implementation



cuNumeric
- numpy code

Legate:
1. Auto-partitions data
2. Determines projection funcs
3. Determines launch grid
4. Launches Tasks

Fusion Implementation



cuNumeric
- numpy code

Legate:
1. Auto-partitions data
2. Determines projection funcs
3. Determines launch grid
4. **Task Fusion**
5. Launches Tasks

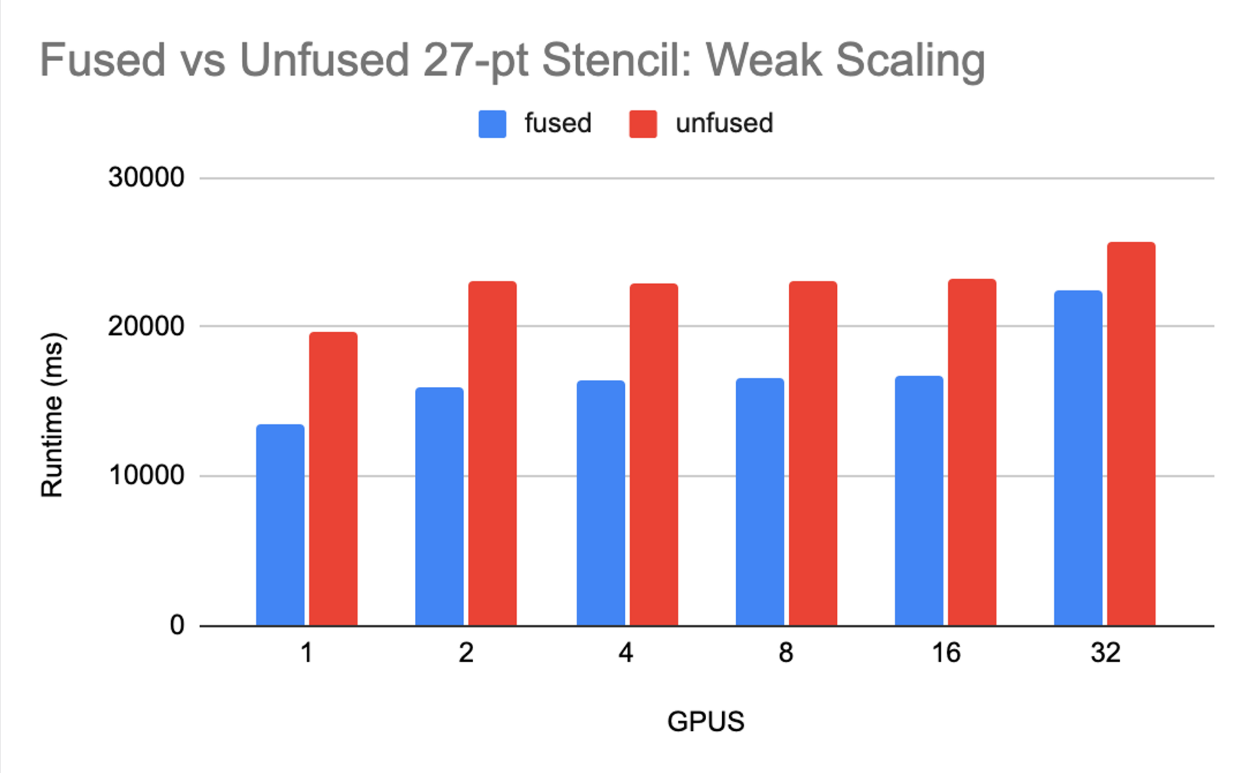
Constraint Application Frequency

5 Benchmarks

Barriers Generated by Each Constraint				
Benchmark	CommunicationAbsence	LaunchSpace	ProducerConsumer	Projection
27-pt (3D) stencil	0%	.1%	49.5%	50.4%
Black-Scholes	100%	0%	0%	0%
Mandelbrot	0%	0%	0%	0%
Logistic Regression	75%	25%	0%	0%
Jacobi Iteration	97.8%	2.2%	0%	0%

Percentage of All Tasks Fused and Metrics on Length of Fused Tasks				
Benchmark	min length	avg length	max length	% tasks fused
27-pt (3D) stencil	2	18	47	98%
Black-Scholes	3	31.4	49	98%
Mandelbrot	2	49.8	50	100%
Logistic Regression	3	3	3	75%
Jacobi Iteration	2	2.75	3	63%

Speedup: Weak Scaling



Speedup

Speedup from Task Fusion on 1-32 GPUs					
Benchmark	1 GPU	4 GPUs	8 GPUs	16 GPUs	32 GPUs
27-pt (3D) stencil	1.43x	1.40x	1.39x	1.38x	1.23x
Black-Scholes	1.55x	1.43x	1.45x	1.42x	1.44x
Mandelbrot	1.17x	1.13x	1.12x	1.12x	1.11x
Logistic Regression	1.18x	1.13x	1.11x	1.16x	1.15x
Jacobi Iteration	1.04x	1.00x	.98x	1.02x	.96x

Jacobi iteration: few fusable tasks

Percentage of All Tasks Fused and Metrics on Length of Fused Tasks				
Benchmark	min length	avg length	max length	% tasks fused
27-pt (3D) stencil	2	18	47	98%
Black-Scholes	3	31.4	49	98%
Mandelbrot	2	49.8	50	100%
Logistic Regression	3	3	3	75%
Jacobi Iteration	2	2.75	3	63%

Further Work

Different (non-greedy) fusion algorithms

Kernel fusion in leaf tasks

Different languages

Further Work

Different (non-greedy) fusion algorithms

Kernel fusion in leaf tasks

Different languages

Thank you