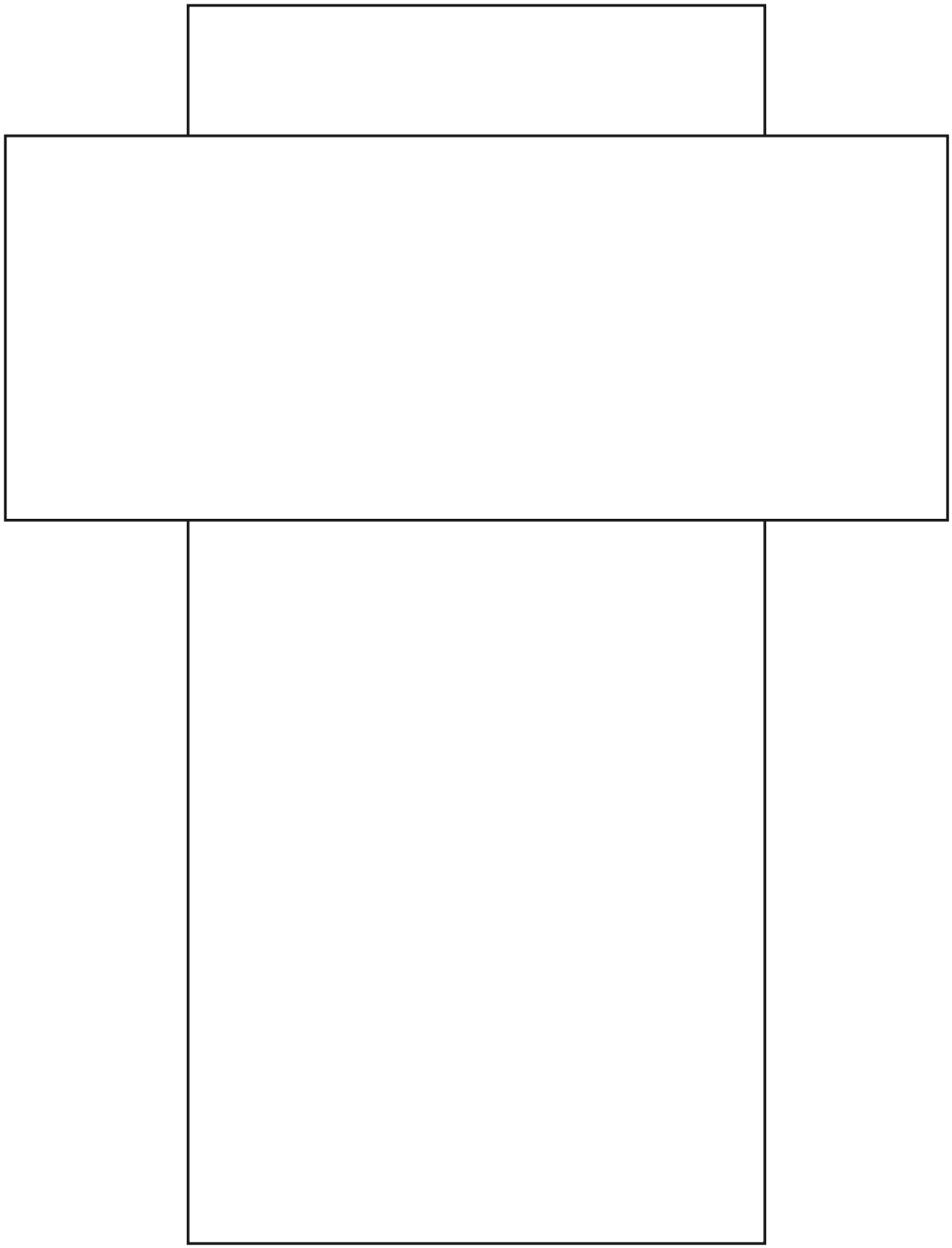
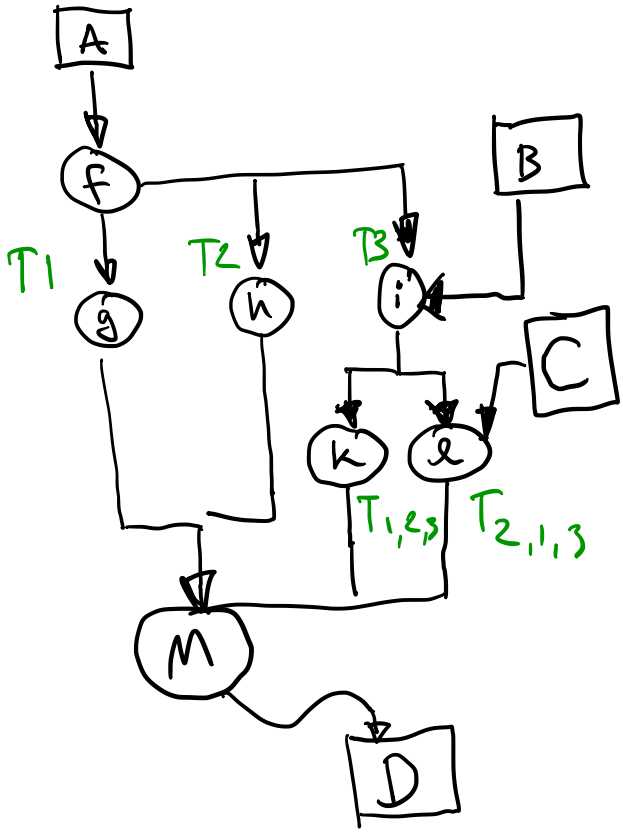


Parallel programming Next few weeks

- Shared memory programming with openMP.
- Distributed memory programming.
- Today Some basic notation and introduction to concepts in parallel programing.
- Note: We will not cover everything but focus on the things we need for projects.
- We will not talk very much about parallelism at the hardware level.

Example:



Squares are DATA

Circles are TASKS

Tasks operate on Data

Tasks have

- Data dependencies
- Control dependencies

Q: Where can parallelism be exploited? What is the critical path?

Data Parallelism

- Any parallelism that grows with the size of data set.
- The more data the more tasks can be used.

  Example?
- Operations on the data may be same or different.
- This is a great kind of parallelism as it scales
- More data per tasks granularity goes up

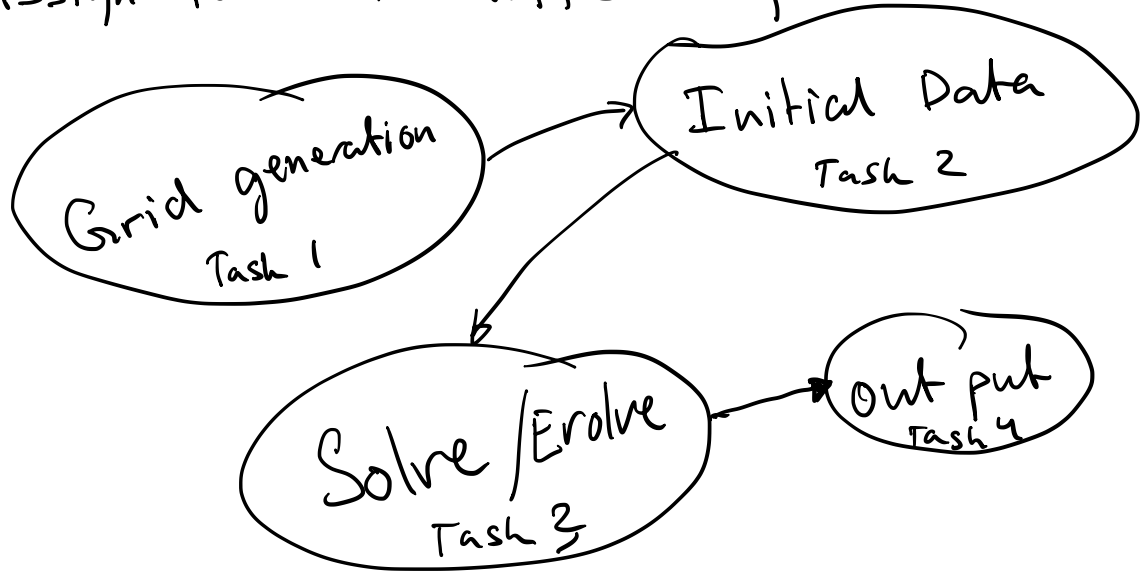
```
do i = 1, N
    A(i) = α * B(i)
end
```

```
do i=1, N/2          do i=N/2+1, N
   ...                  ...
end do               end do
```

Functional parallelism (decomposition)

Examples?

Assign tasks to different functions



Can only scale up to a constant factor.

Data parallelism is often regular parallelism

Ex matrix-vector multiply for dense matrix

It can be irregular like for sparse matrices

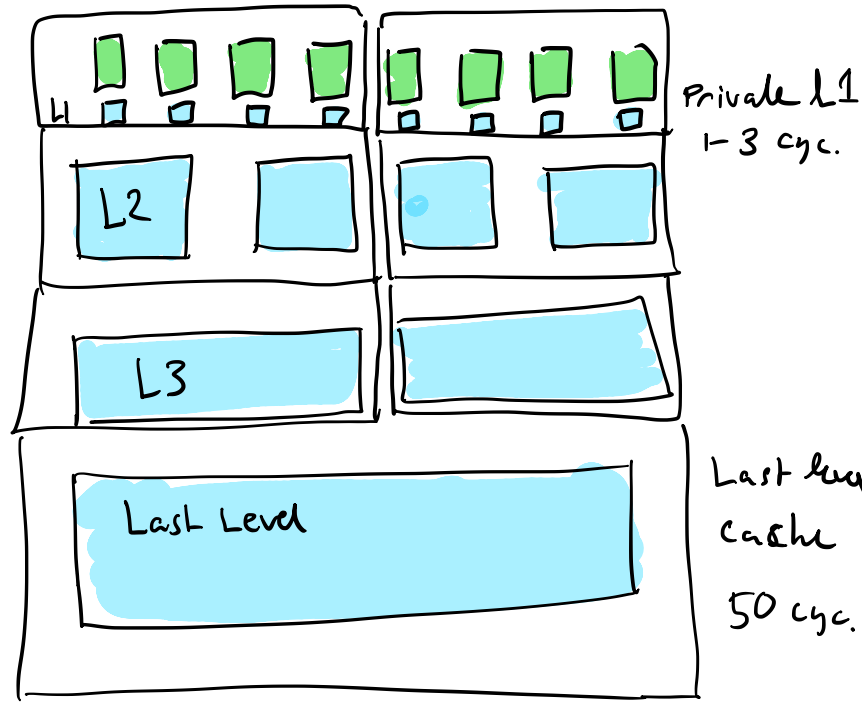
- Regular: Easy to predict data dependence
  easy to load balance

  Irregular: Harder to load balance.

Machine models (Model of how we think the computer looks like)

- Multi core architecture
  8-core here
- Instruction level parallelism
  ILP
- Thread parallelism. TPL
- Vector parallelism DLP

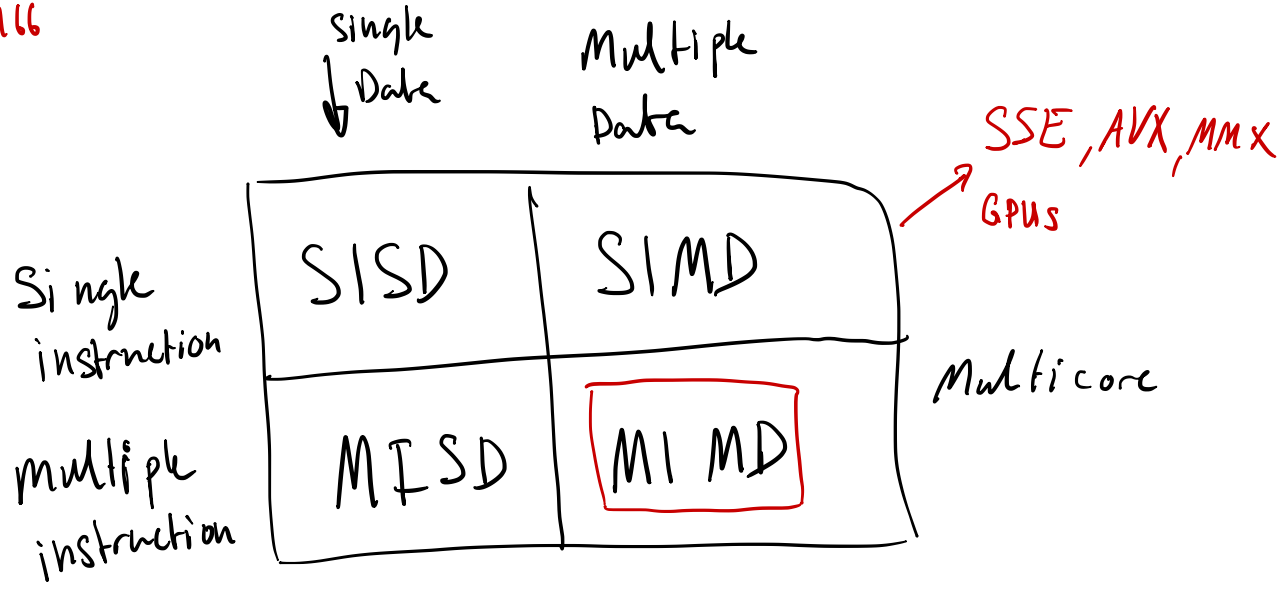
Shared memory



Flynn's Taxonomy

Classification of parallel comp.

1966



Performance metric

Execution time: Time to complete task T_E

Should be reduced by parallelism. (Example)

Throughput: Work completed per time unit

Might not decrease the time for a single work unit.

(example?)

Speedup How much faster your code is in parallel compared to the fastest serial code.

$$S(p) = T_{serial} / T(p), \quad \text{Execution on } p \text{ cores}$$

$$\text{Throughput}(p) / \text{Throughput in serial}$$

What can limit speedup?

- Load imbalance
- Serial work
- Resources

Efficiency : $T_{serial} / (pT(p)) = E(p)$

$S/p = E(p)$

Scalability

Strong (Fixed size) Scaling: Fix problem size

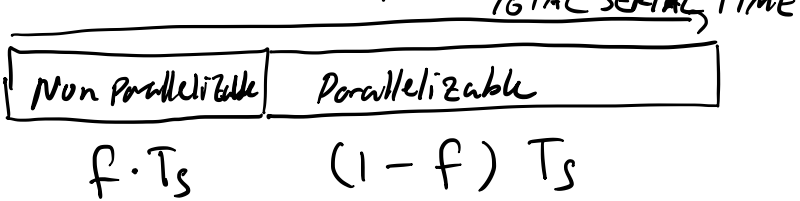
$T(p)$ vs p

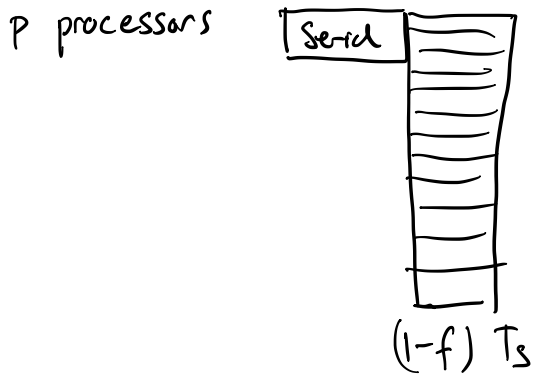
Weak (Iso granular) Scaling : Fixed work per proc.

$T(p)$ vs p

Amdahl's Law: Speedup is limited by serial fraction

$$S \leq \frac{T_{serial}}{f T_s + (1-f)\frac{T_s}{p}} = \frac{p}{1 + (p-1)f} \approx \frac{1}{f} \quad p \to \infty$$





f	Speedup
20%	5
10%	10
1%	100

Looks dire

Gustafson's Law

Scaling up the work load many problems can make use of parallelism with much larger speedup than predicted by Amdahl's law

$$S(P) = P - \alpha(P-1)$$

α — non-parallelizable

This is typically true for DNNs. (Data parallelism)