Network Embedding and Amdahl's Law

A. Sahu

Dept of CSE, IIT Guwahati

Outline

- Topology Embedding
- Amdahl's Law
- Introduction to GPU and GPU coding
  - Use of accelerator

Network Embedding

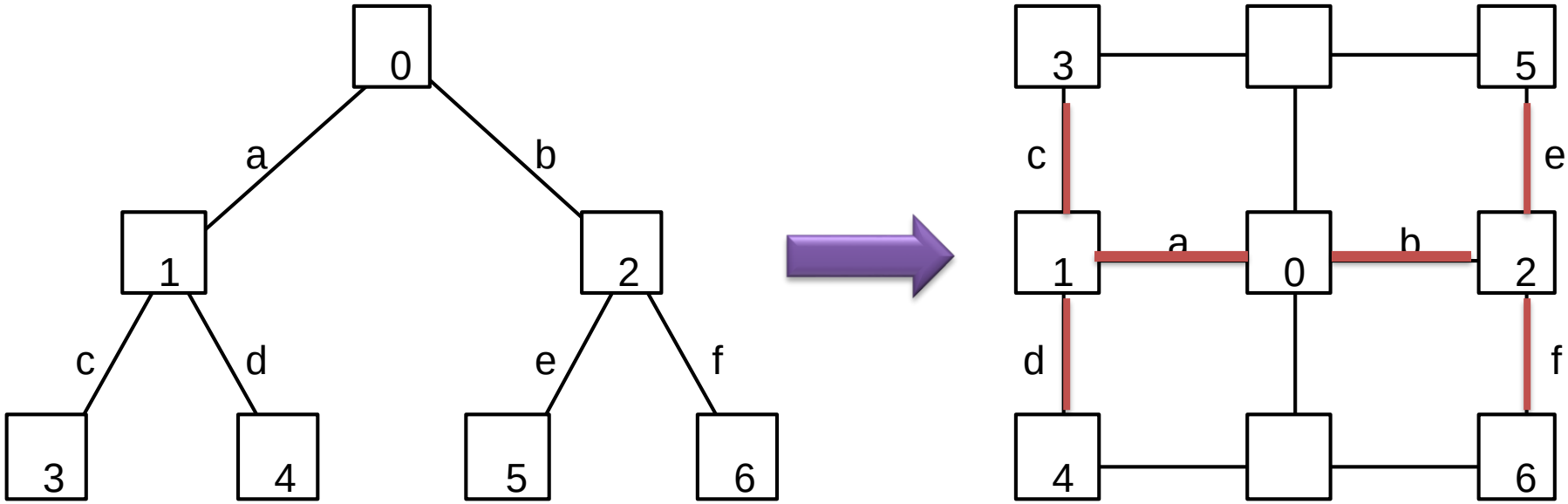
Embeddings and Their Usefulness

- Embedding allow us to
  - Use an algorithm/ program developed for an existing architecture on a new one
  - By simply letting the new one follow the same steps as older one.

Embeddings and Their Usefulness

- Also Embedding Helps in Application Mapping to Architecture
  - Embedding Tree to Mesh : Running Mergesort on a MESH
  - Embedding Tree to HyperCube : Running Mergesort in Mesh
  - Embedding Mesh to Tree : Running Matrix multiplication on TREE
  - And so on ….

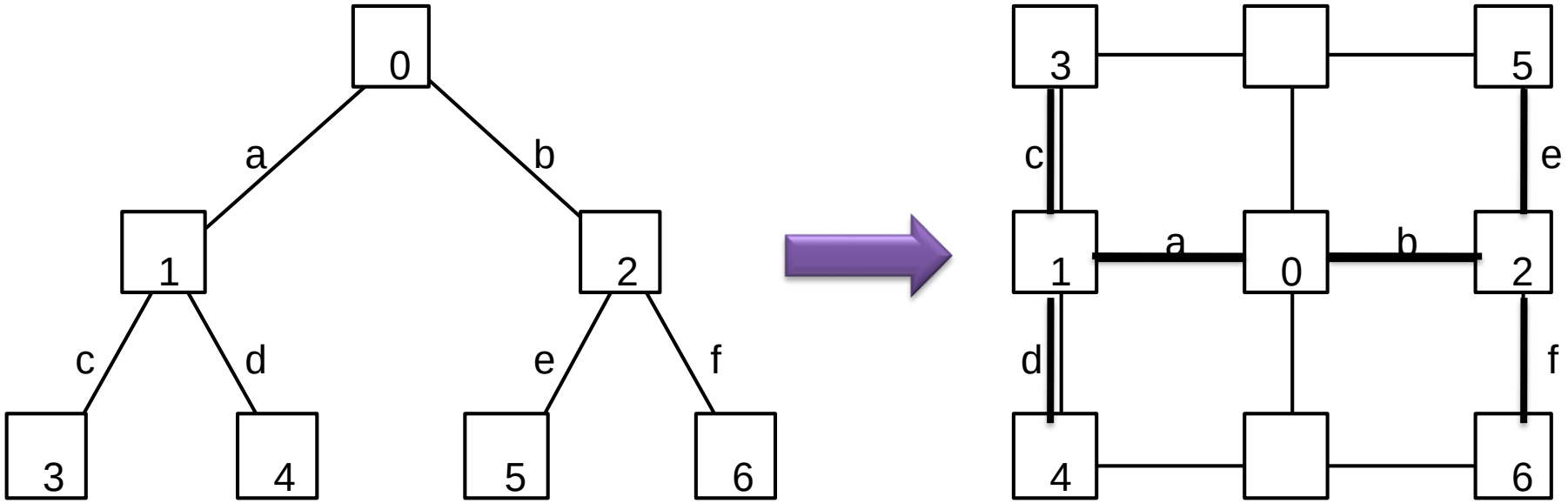
Embeddings 7 Node Tree to MESH



Embeddings and Their Usefulness

- **Dilation:** Longest path onto which an edge is mapped (routing slowdown)
- **Congestion:** Max number of edges mapped onto one edge (contention slowdown)
- **Load factor:** Max number of nodes mapped onto one node (processing slowdown)

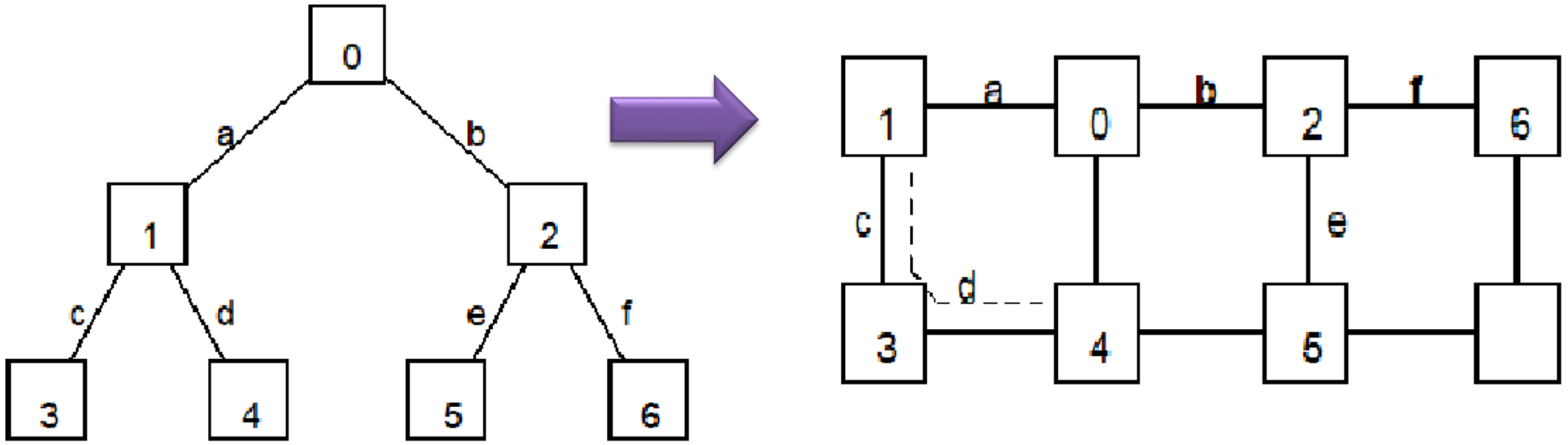
Embeddings and Their Usefulness



Dilation = 1
Congestion = 1
Load factor = 1

Dilation: Longest path onto which an edge is mapped (routing slowdown)
Congestion: Max number of edges mapped onto one edge (contention slowdown)
Load factor: Max number of nodes mapped onto one node (processing slowdown)

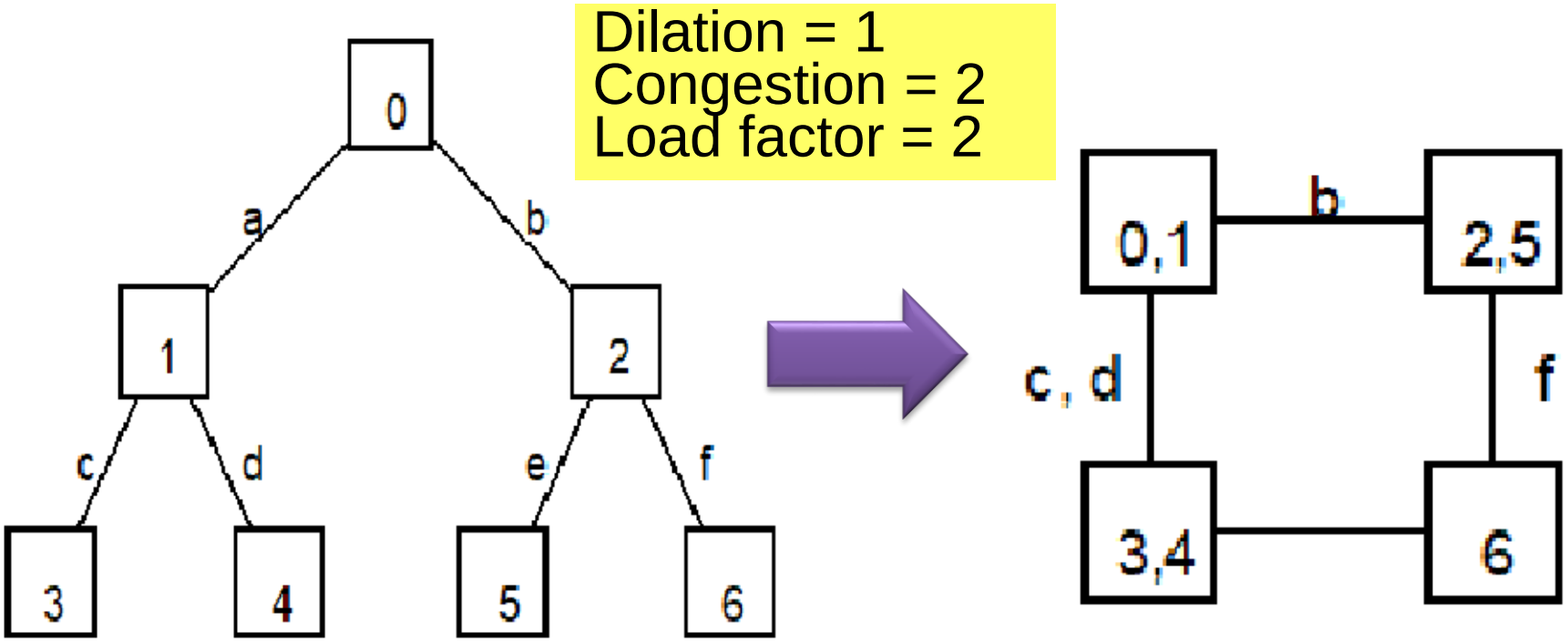
Embeddings and Their Usefulness



Dilation = 2
Congestion = 2
Load factor = 1

Dilation: Longest path onto which an edge is mapped (routing slowdown)
Congestion: Max number of edges mapped onto one edge (contention slowdown)
Load factor: Max number of nodes mapped onto one node (processing slowdown)

Embeddings and Their Usefulness



Dilation: Longest path onto which an edge is mapped (routing slowdown)
Congestion: Max number of edges mapped onto one edge (contention slowdown)
Load factor: Max number of nodes mapped onto one node (processing slowdown)

Performance of Parallel Program (Amdahl's Law)

Example: OpenMP Parallel Program

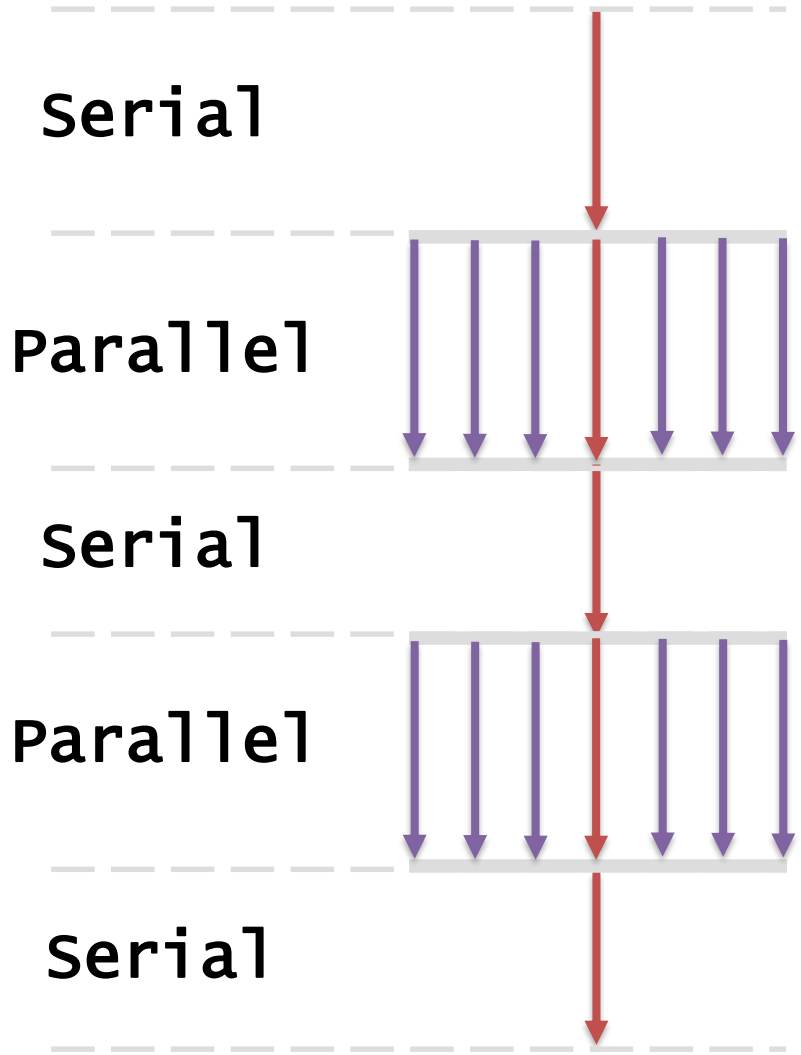
```
printf("begin\n");
N = 1000;

#pragma omp parallel for
for (i=0; i<N; i++)
    A[i] = B[i] + C[i];

M = 500;

#pragma omp parallel for
for (j=0; j<M; j++)
    p[j] = q[j] - r[j];

printf("done\n");
```



Speed up and efficiency

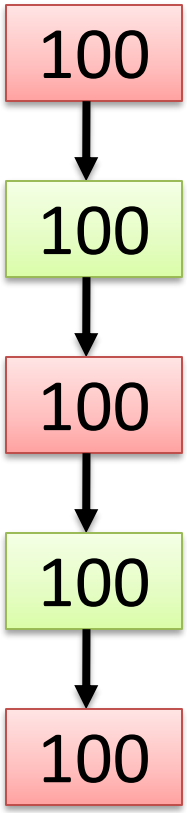
- **Notion :** T_1 =Time on Uni-processor, T_p= Time on p Processors

$$\textbf{Speed up} = S_p = T_1/T_p \leq p$$

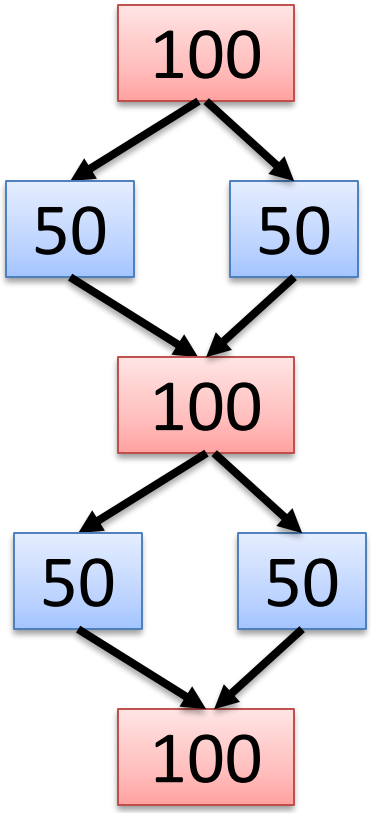
$$\textbf{Efficiency} = E_p = T_1/(p.T_p)$$

- Usually $S_p < p$ or $E_p < 1$ due to overhead
- Some time superliner speed up reported ($S_p > p$ or $E_p > 1$)
  - Failure to use the best sequential algorithm
  - Advantage due to larger memory

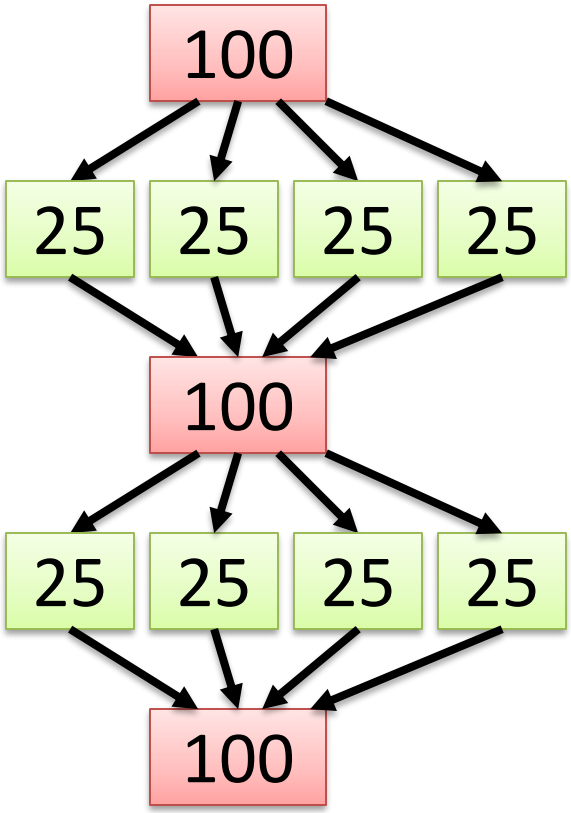
Amdahl's law



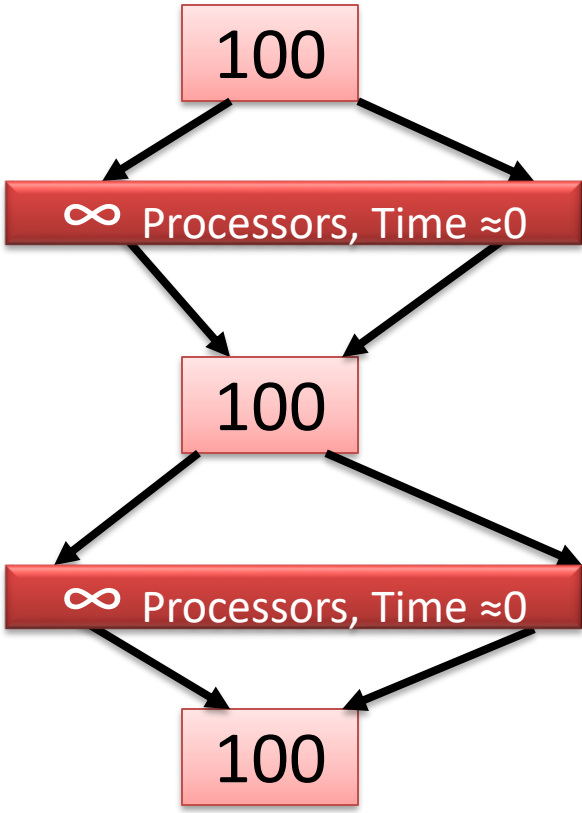
Work 500,
Time 500
Sp=1X



Work 500,
Time 400
Sp=1.25X



Work 500,
Time 350
Sp=1.4X



Work 500,
Time 300
Sp=1.7X

Amdahl's Law

$$\text{Serial fraction} = s = \frac{T_s}{T_1}$$

$$T_p = T_s + \frac{T_1 - T_s}{p}$$

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{T_s + \frac{T_1 - T_s}{p}} = \frac{T_1}{T_s(1 - \frac{1}{p}) + \frac{T_1}{p}}$$

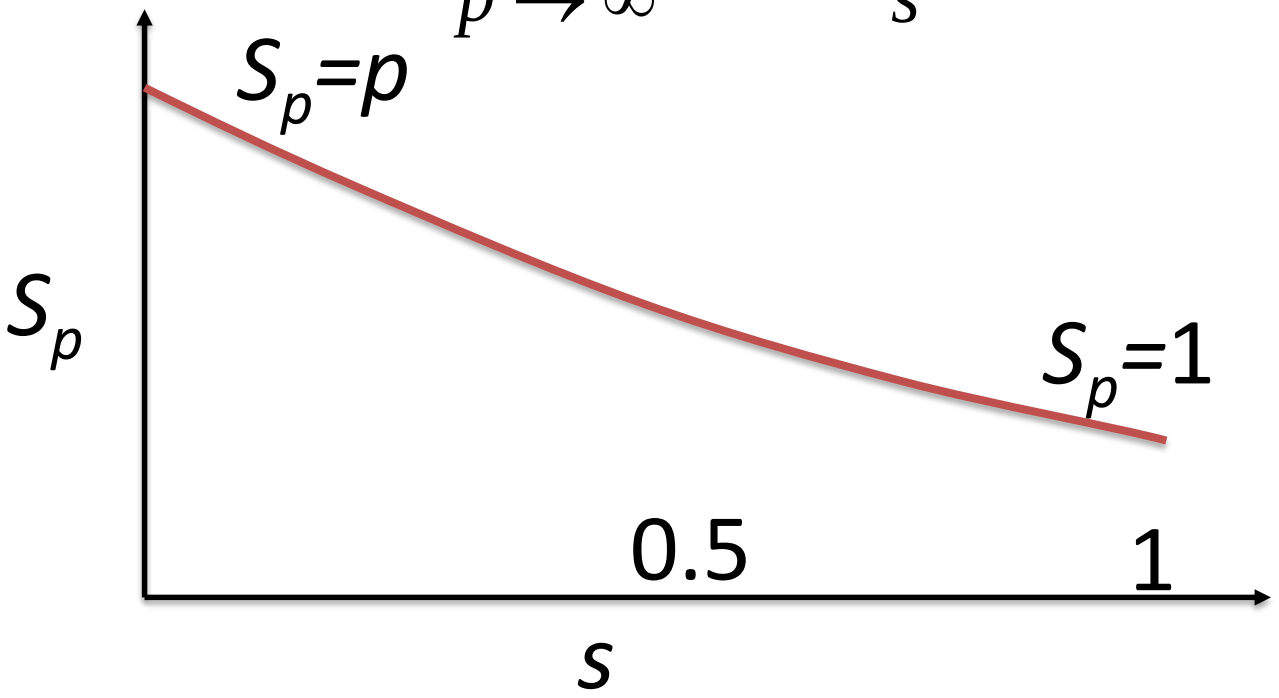
Amdahl's Law

$$S_p = \frac{T_1}{T_s(1-\frac{1}{p})+\frac{T_1}{p}}$$

$$Sp = \frac{1}{s(1-\frac{1}{p})+\frac{1}{p}} = \frac{p}{s(p-1)+1}$$

$$s = \frac{T_s}{T_1}$$

$$\underset{p \to \infty}{Lt} \quad S_p = \frac{1}{s}$$



Assumption behind Amdahl's Law

- All the processors are homogeneous
- All the communication costs are zero
- All the memory accesses takes unit time (PRAM)
- All the parallel section are purely parallel: Divisible load

$$Sp = \frac{1}{s(1-\frac{1}{p}) + \frac{1}{p}} =$$

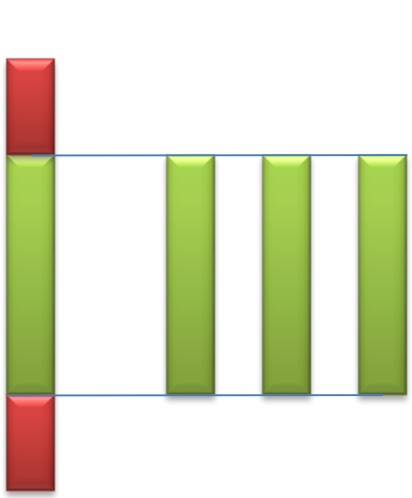
$$\lim_{p \to \infty} S_p = \frac{1}{s}$$

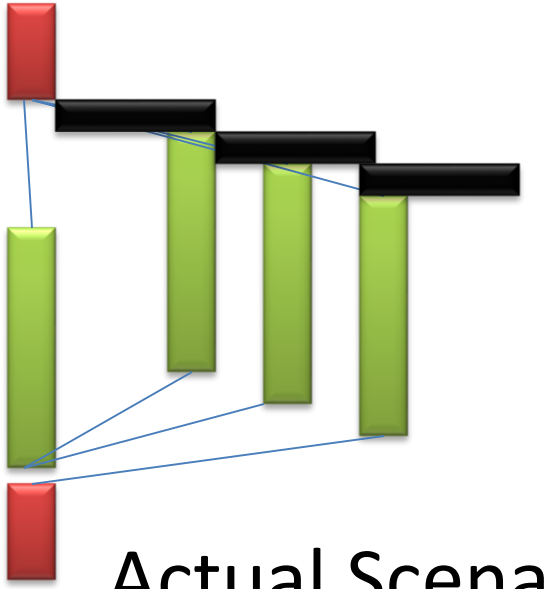
~~All the memory accesses takes unit time (PRAM)~~

- Memory Hierarchy: Cache Memory
- Suppose Application A run on 2Ghz Intel Pentium P4 uni-processor takes 10 minutes
- Same Application A run on 2Ghz Intel i5 Processor (Quad core) **may run much faster than 4X. Super linear Speedup**
  - Earlier cache size was 1MB in P4
  - Now cahce size is 4MB, the whole App A may be fit into Cache. No capacity misses…

~~All the communication cost are zero~~

- Pthread Creation, Fork/Join takes significant amount of time





Ideal Scenario

Actual Scenario

$T = T_{s1} + T_{s2} + T_p$

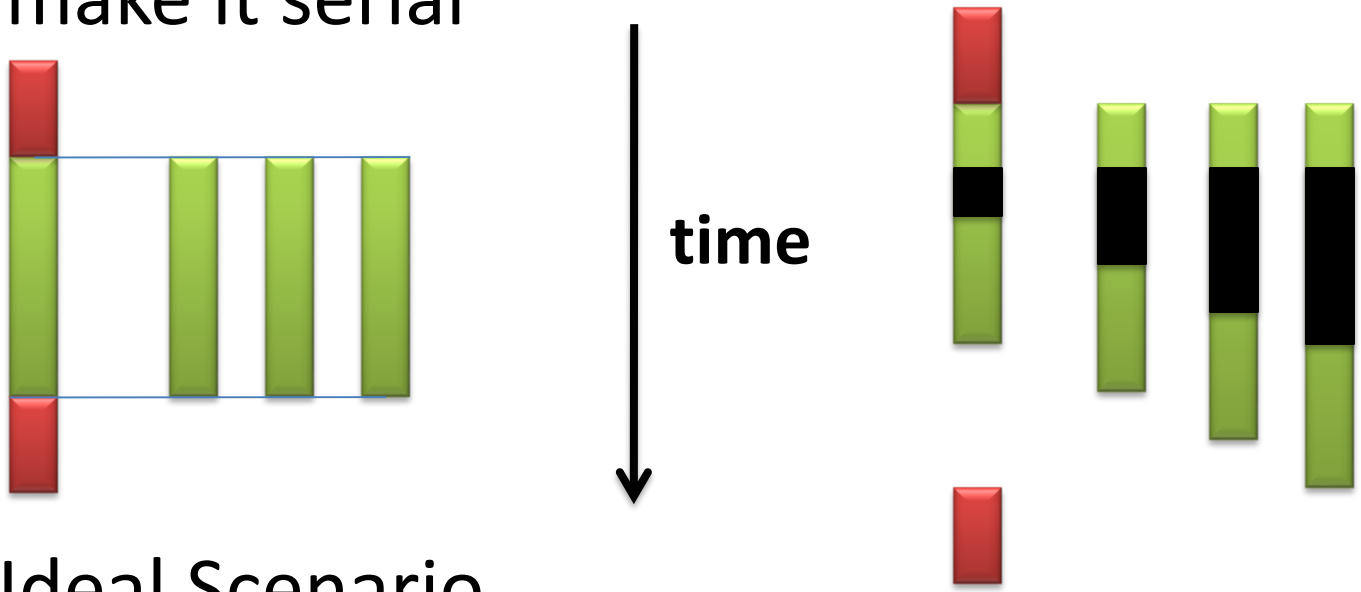
$T = T_{s1} + T_{s2} + T_p + 4(T_f + T_j)$

~~All the parallel section are purely parallel: Divisible load~~

- **Parallel threads** accessing to shared resources make it serial
- Using higher number of processor may need to collaborate and have more communication
- Application parallel section may not be scale up with processor : **Grain size**

~~All the parallel section are purely parallel: Divisible load~~

- **Parallel threads** accessing to shared resources make it serial



Ideal Scenario

Actual Scenario with share Resource

$T = T_{s1} + T_{s2} + T_p$

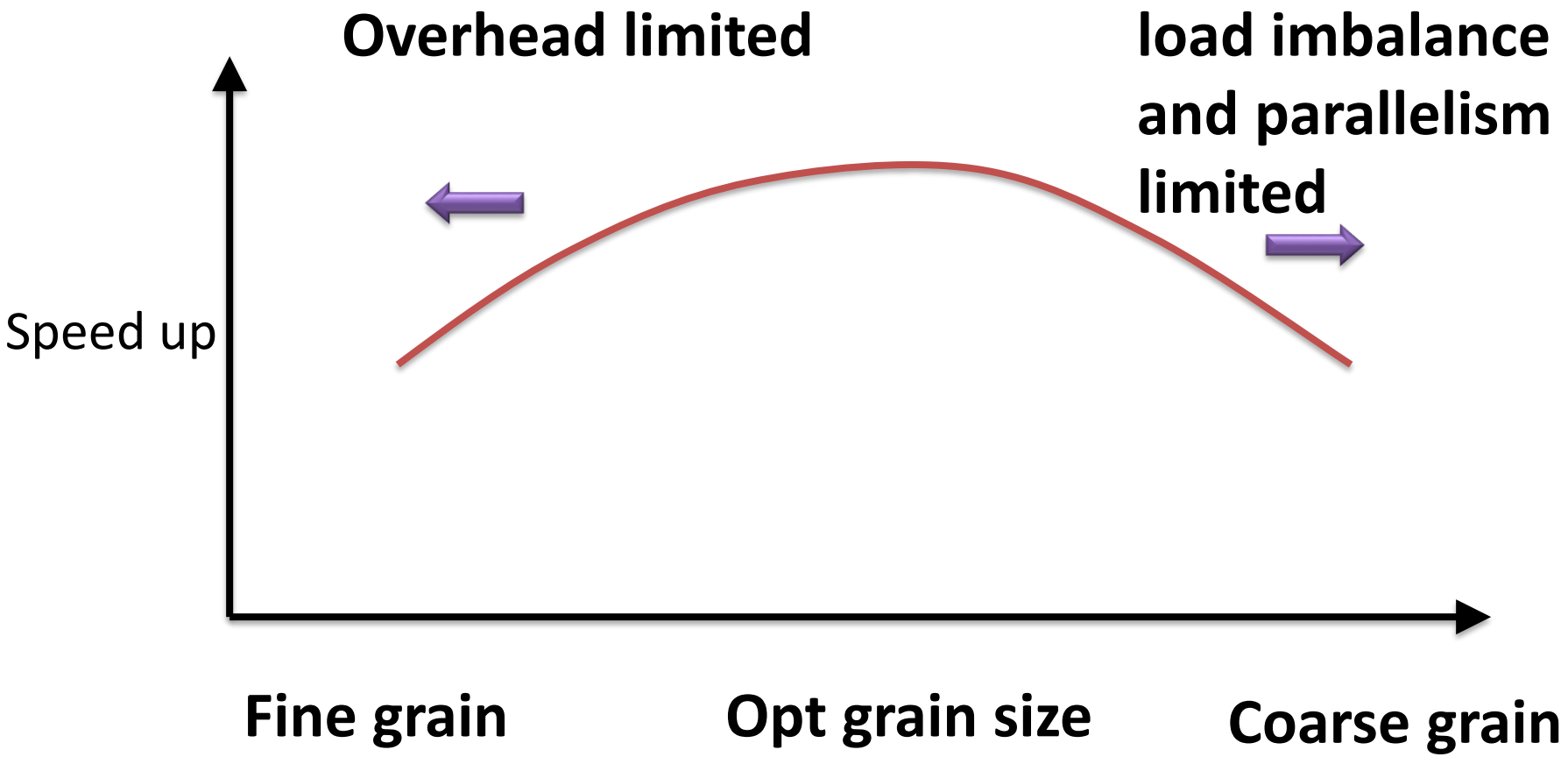
$T = T_{s1} + T_{s2} + T_p + 4(T_{cs})$

~~All the processors are homogeneous~~

- Asymmetric Processing Environment
- One **big** core and many **small or tiny** cores
- Intel Xeon : 8 big cores

- GPU : 4/8 **big** cores+ 2000 **tiny** cores
- Intel Phi : 4/8 **big** cores(host)+ 250 **small** cores

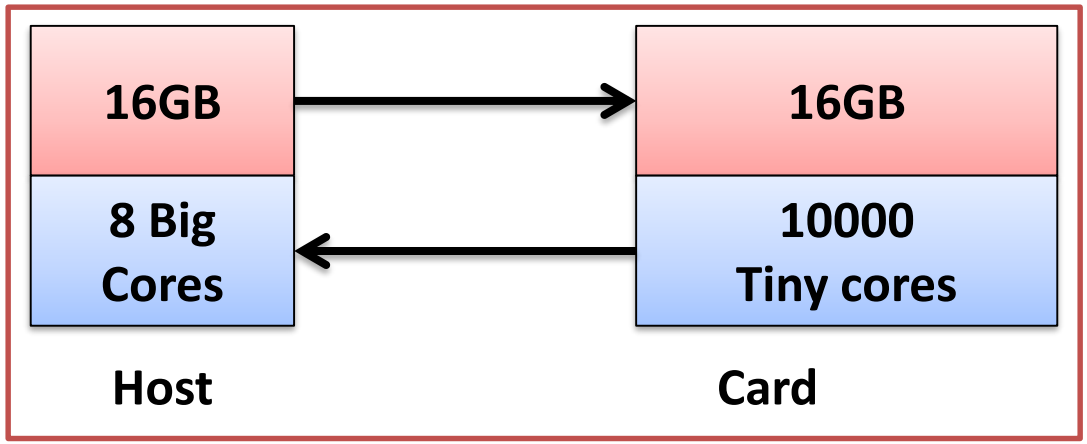
Grain size and performance



GPU
&
Application Analysis for GPGPU

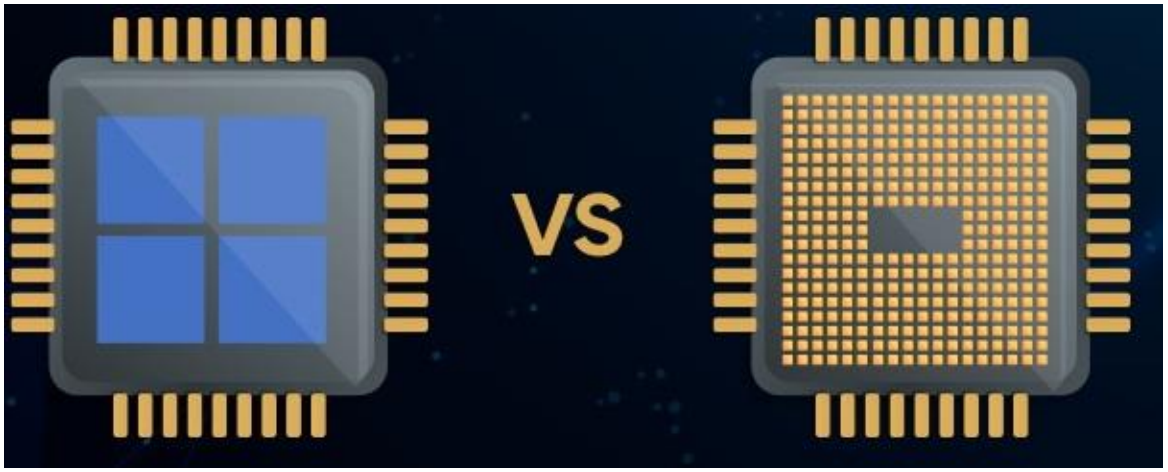
GPU

- Graphics Cards to Motherboard PCI Slot
- To accelerate Graphics computation
- Earlier day : It was fixed purpose
- Now a days, it is programmable, configurable
  - Why not to use them for general purpose?
  - For what kind of application



GPU

- GPU vs CPU



- GPU Cards
  - GTX3090: **10496** cores, 24GB DDR6-384bit interface, Rs 1.3L

GPU Philosophy

- **Small independent function/code executed huge number of time**
- **Number of cores in thousands, tiny cores**
- Cores are organized in cluster
  - Kepler SMX: 14 SM, 192 SP Cuda cores/SM, 64 DP units, 32 SFU, 32 (LD/ST) U
  - TU102-RTX 2080Ti: 72 SM, 4608 Cuda Cores, 576 Tensor core, 72 RayTrace cores
- Explicit memory hierarchy, programmer controlled
- Also implicit memory hierarchy : cache