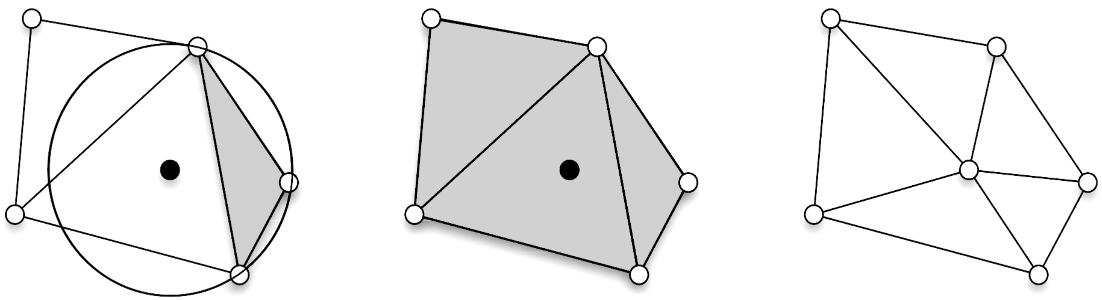
Parallel Graph Algorithms

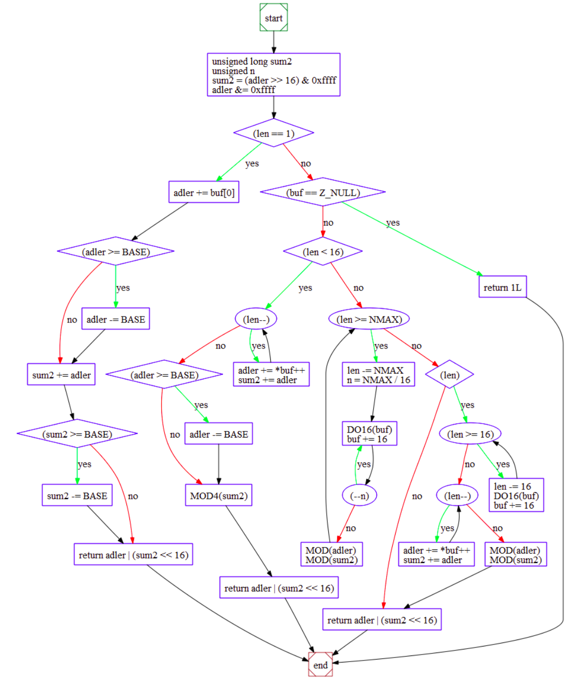


Rupesh Nasre.

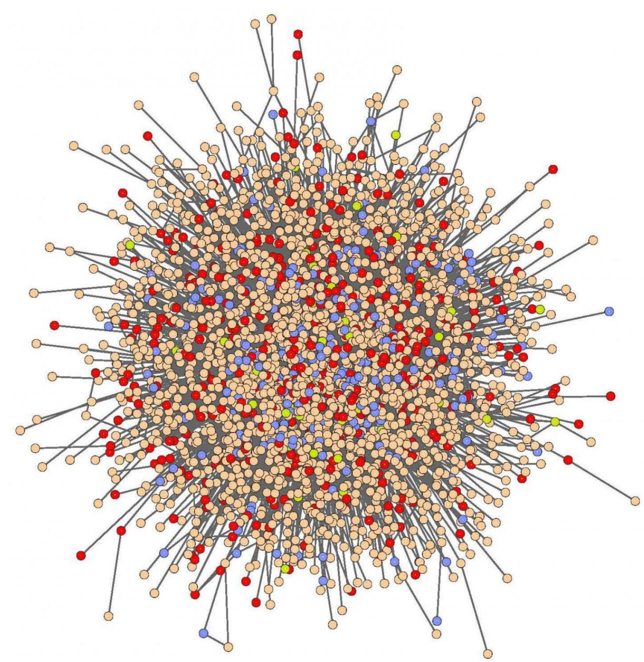
IIT Madras

PACE
Programming Languages, Architecture
and Compilers Education Laboratory

January 2025



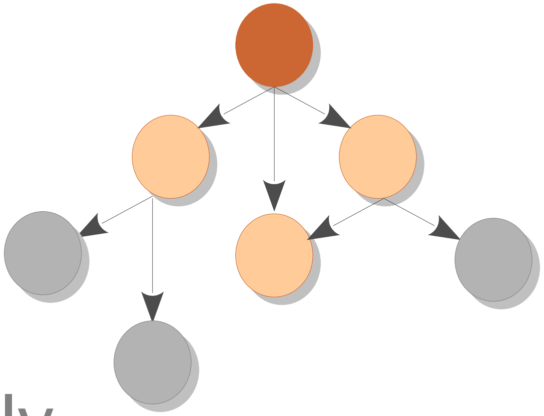
Graphs are Everywhere!



Source: Google images

Graphs

- Where do we encounter graphs?
  - Social networks, road connections, molecular interactions, planetary forces, …
  - snap, florida, dimacs, konect, ...
- Why treat them separately?
  - They provide structural information.
  - They can be processed more efficiently.
- What challenges do they pose?
  - Load imbalance, poor locality, …
  - Irregularity



What is IrRegularity?

- Data-access or control patterns are unpredictable at compile time.

Irregular data-access

```
int a[N], b[N], c[N];
readinput(a);

c[5] = b[a[4]];
```

Irregular control-flow

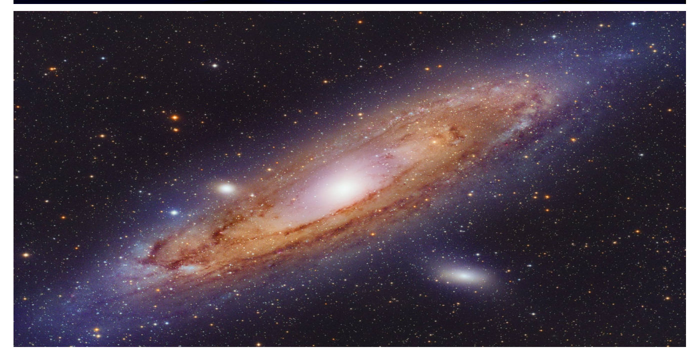
```
int a[N];
readinput(a);

if (a[4] > 30) {
      ...
}
```

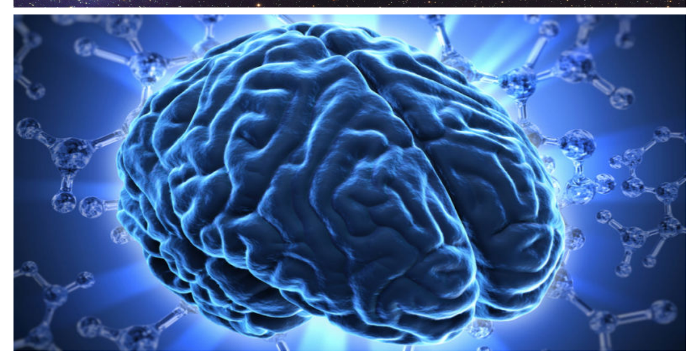
Needs dynamic techniques

Pointer-based data structures often contribute to irregularity.

Scalability

- **Meta / Facebook**
  - 2.2 billion active users
  - 1.3 billion is India's population
  - e.g. top people in the world
- **Milky Way**
  - over 100 billion stars
  - e.g. finding possibility of life
- **Human Brain**
  - 100 billion neurons
  - Artificial intelligence





Source: google images

`Finding betweenness centrality on a million node graph (in a sequential manner) takes several weeks!`

Handling Large Graphs

Storage

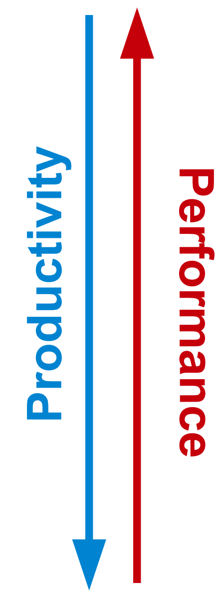
- Distributed setup
  - Graph is partitioned across a cluster.
- External memory algorithms
  - Graph partitions are processed sequentially.
- Algorithms on compressed data
  - Compression needs to maintain retrieval ability.
- Maintaining graph core
  - Removal of unnecessary subgraphs.

Time

- Parallelism
  - Multi-core, distributed, GPUs
- Approximations
  - Approximate computing

Parallelism Approaches

- Manual
  - OpenMP, MPI, CUDA
- Libraries
  - Galois, Ligra, LonestarGPU, Gunrock, ...
- Domain-Specific Languages
  - Green-Marl, Elixir, Falcon, ...



Specifying Parallelism

- Do not specify.
  - Sequential input, completely automated, currently very challenging in general
- Implicit parallelism
  - aggregates, aggregate functions, primitive-based processing, ...
- Explicit parallelism
  - pthreads, MPI, OpenCL, ...

Identifying Dependence

```
for (ii = 0; ii < 10; ++ii) {
    a[2 * ii] = ... a[2 * ii + 1] ...
}
```

Is there a flow dependence between different iterations?

Flow dependence is read-after-write (to the same memory location).
w →→→→ r

Dependence equations

$0 <= ii_w < ii_r < 10$

$2 * ii_w = 2 * ii_r + 1$

which can be written as

$0 <= ii_w$

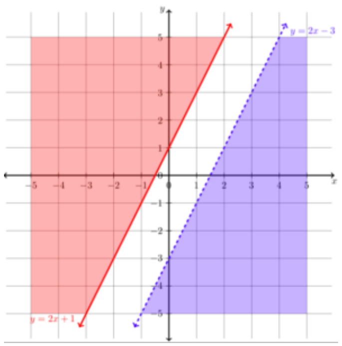
$ii_w <= ii_r - 1$

$ii_r <= 9$

$2 * ii_w <= 2 * ii_r + 1$

$2 * ii_r + 1 <= 2 * ii_w$

$$\begin{pmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \\ 2 & -2 \\ -2 & 2 \end{pmatrix} \begin{pmatrix} ii_w \\ ii_r \end{pmatrix} <= \begin{pmatrix} 0 \\ -1 \\ 9 \\ 1 \\ -1 \end{pmatrix}$$



Dependence exists if the system has a solution.

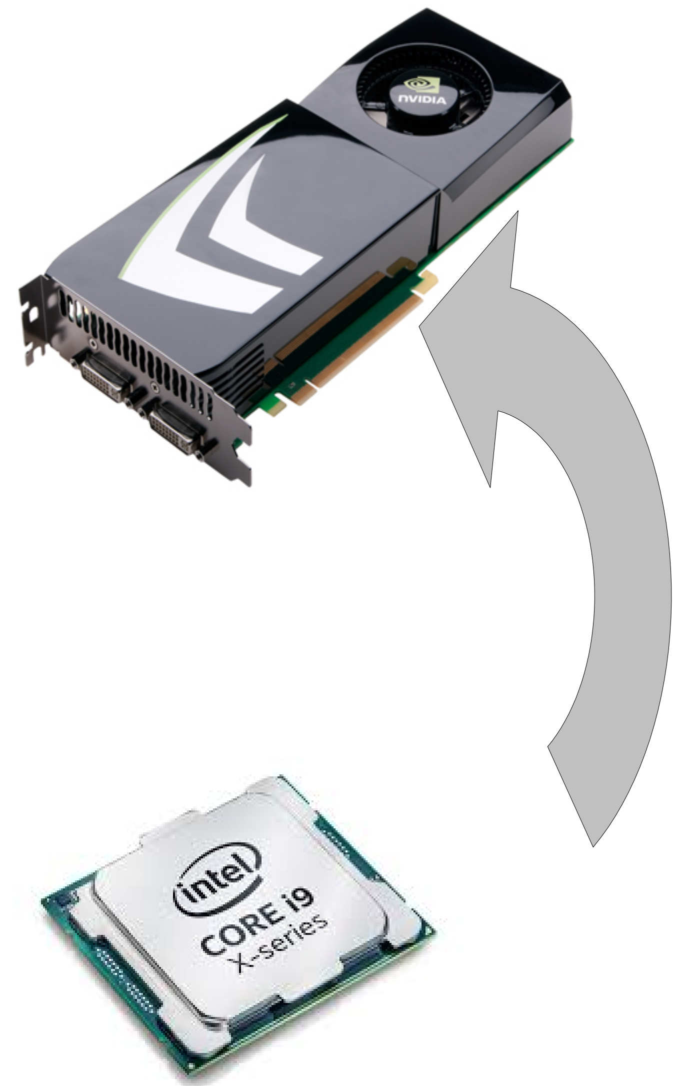
Parallel Architectures

- **Multicore CPUs**
  - Intel, ARM, …
  - pthreads, OpenMP, ...
- **Distributed systems**
  - CPUs with interconnects
  - MPI
- **Manycore GPUs**
  - NVIDIA, AMD, …
  - CUDA, OpenCL, ...

CPU-GPU processing concepts have similarity with those in distributed systems.

What is a GPU?

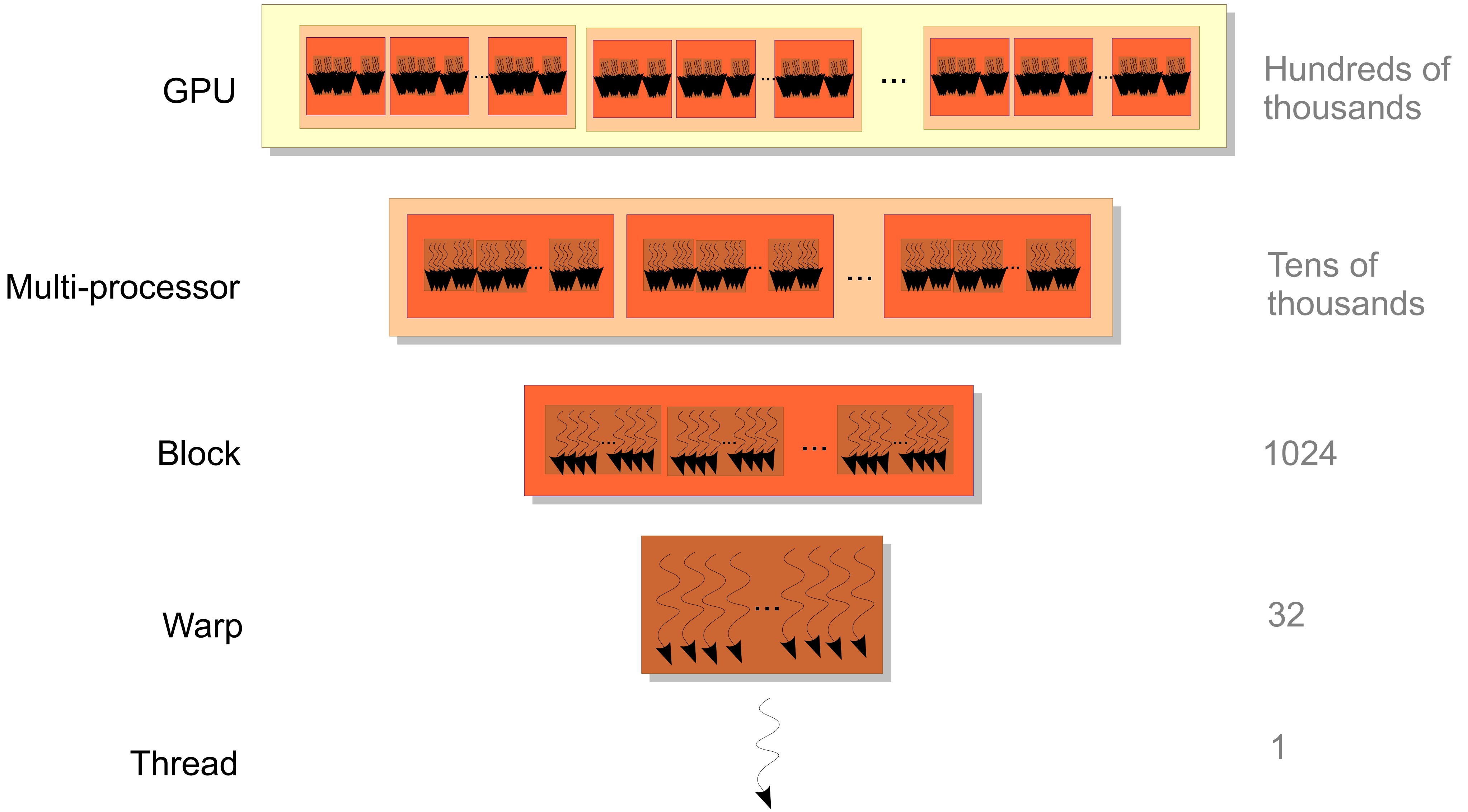
- Graphics Processing Unit
- Separate piece of hardware connected using a bus
- Separate address space than that of the CPU
- Massive multithreading
- Warp-based execution



What is a Warp?

Source: Wikipedia

GPU Computation Hierarchy

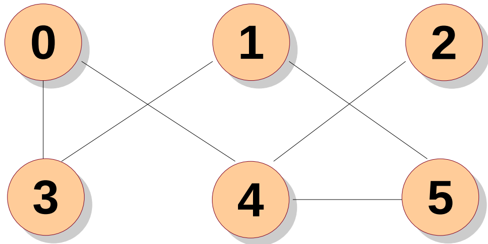


Challenges with GPUs

- Warp-based execution (pre-Volta)
- Locking is expensive
- Dynamic memory allocation is costly
- Limited data-cache
- Programmability issues
  - separate address space
  - low recursion support
  - complex computation hierarchy
  - exposed memory hierarchy
  - ...

Challenges in Graph Algorithms

- Synchronization
  - locks are prohibitively expensive on GPUs
  - atomic instructions quickly become expensive
- Memory latency
  - locality is difficult to exploit
  - low caching support
- Thread-divergence (pre-Volta)
  - work done per node varies with graph structure
- Uncoalesced memory accesses
  - warp-threads access arbitrary graph elements



Graph Representation

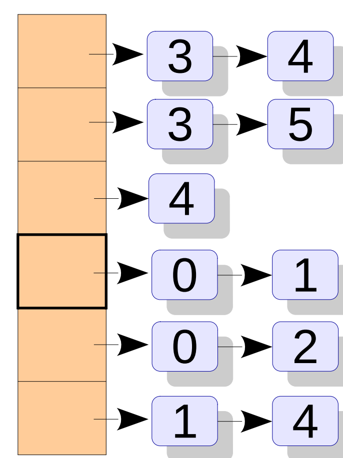
1. Adjacency matrix

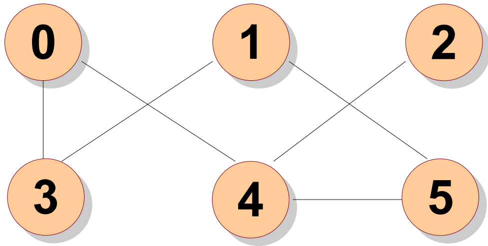
- |V| x |V| matrix
- Each entry [i, j] denotes if edge (i,j) is present in G
- Useful for **dense** graph
- Finding neighbors is O(|V|)

			1	1	
			1		1
				1	
1	1				
1		1			
	1			1	

2. Adjacency list

- |V| + |E| size
- Each vertex i has a list of its neighbors
- Useful for **sparse** graphs
- Finding neighbors is O(max. degree)





Graph Representation

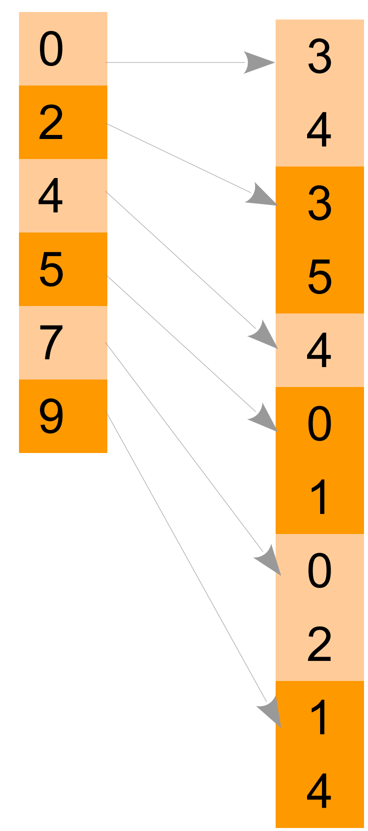
0	3
0	4
1	3
1	5
2	4
3	0
3	1
4	2
5	1
5	4

3. Edge list / Coordinate list (COO)

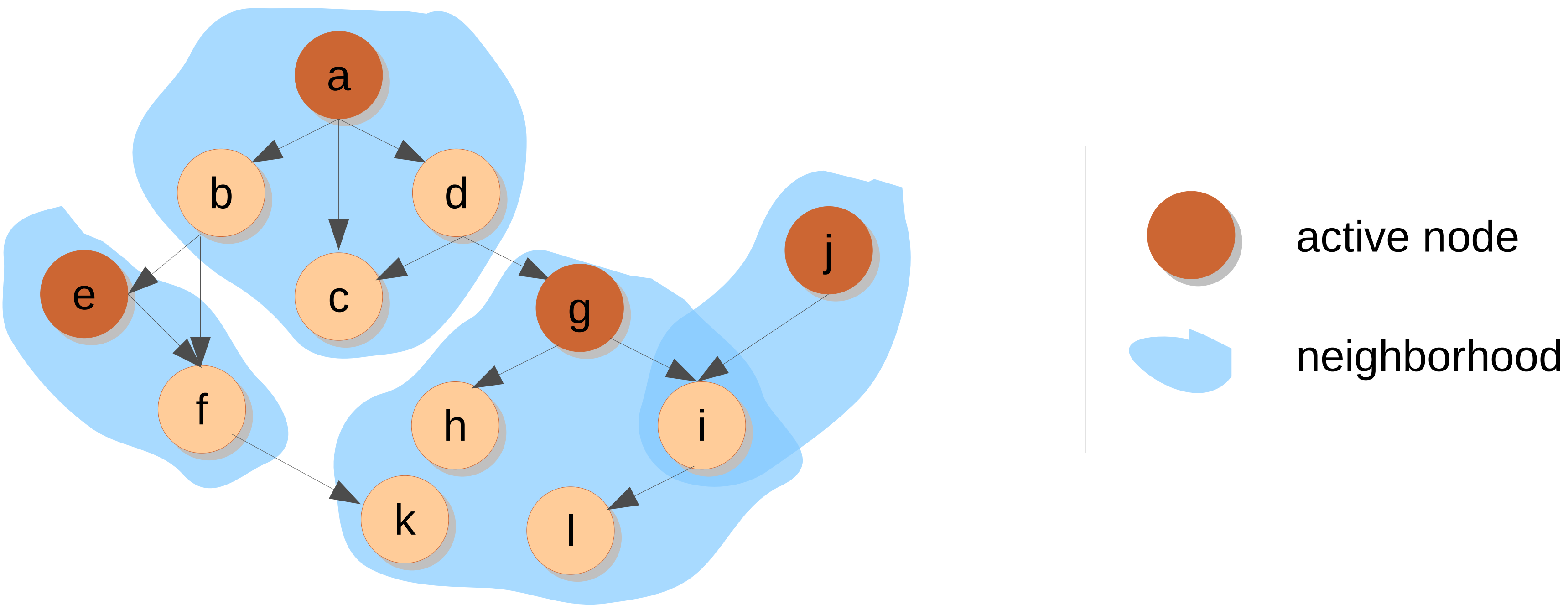
- |E| pairs
- Useful for edge-based algorithms
- Typically sorted on vertex id

4. Compressed sparse row (CSR)

- Concatenated adjacency lists
- Useful for **sparse** graphs
- Useful for data transfer



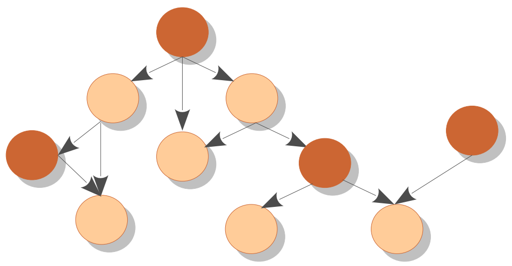
TAO Classification



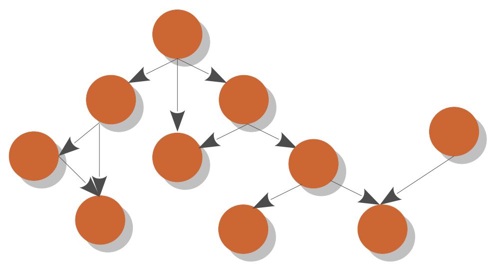
- **Operator formulation**: Computation as an iterated application of operator
- **Topology-driven processing**: operator is applied at all the nodes even if there is no work to do at some nodes (e.g., Bellman-Ford SSSP)
- **Data-driven processing**: operator is applied only at the nodes where there might be work to be done (e.g., SSSP with delta-stepping)

The TAO of Parallelism in Algorithms, Pingali *et al*, PLDI 2011

Data-driven vs. Topology-driven



data-driven



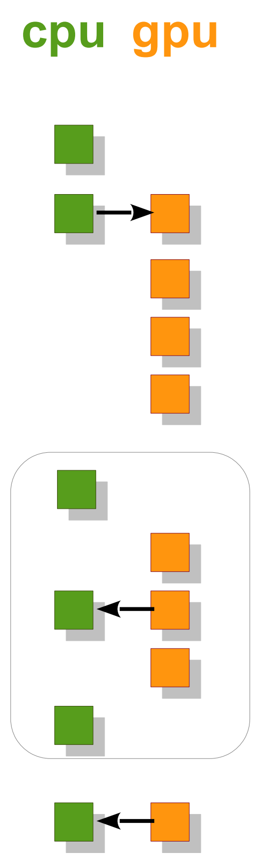
topology-driven

- work-efficient
- centralized worklist
- fine-grained synchronization using atomics
- complicates implementation

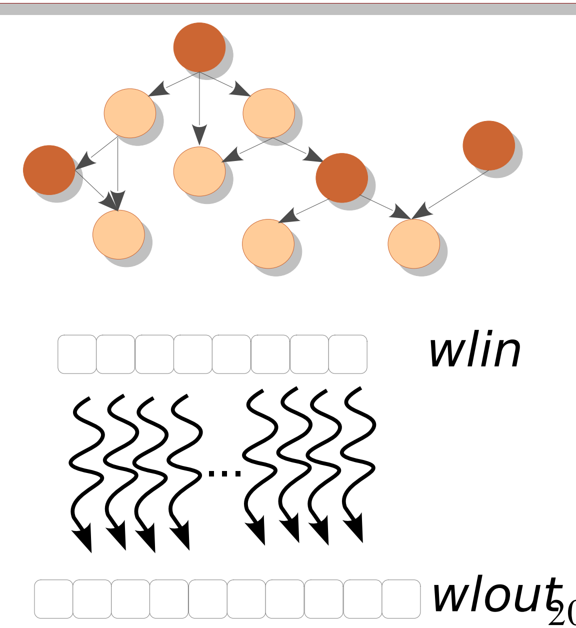
- performs extra work
- no worklists
- coarse-grained synchronization using barriers
- easier to implement

Data-driven: Base Version

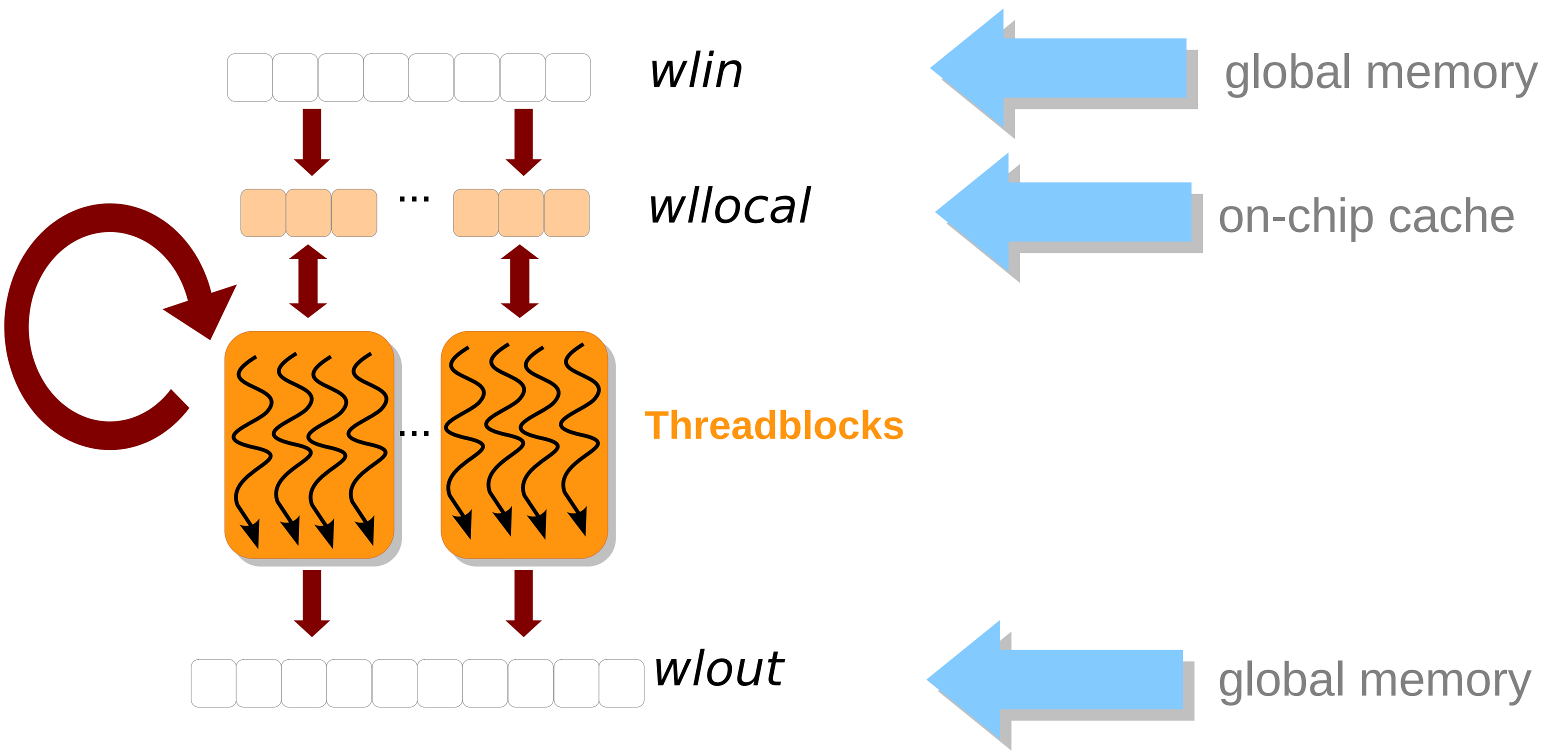
```
main {
    read input
    transfer input
    initialize_kernel
    initialize_worklist(wlin)
    clear wlout

    while wlin not empty {
        operator(wlin, wlout, ...)
        transfer wlout size
        clear wlin
        swap(wlin, wlout)
    }
    transfer results
}
```




```
sssp_operator(wlin, wlout, ...) {
    src = wlin[...]
    dsrc = distance[src]
    forall edges (src, dst, wt) {
        ddst = distance[dst]
        altdist = dsrc + wt
        if altdist < ddst {
            distance[dst] = altdist
            wlout.push(dst)
} } }
```

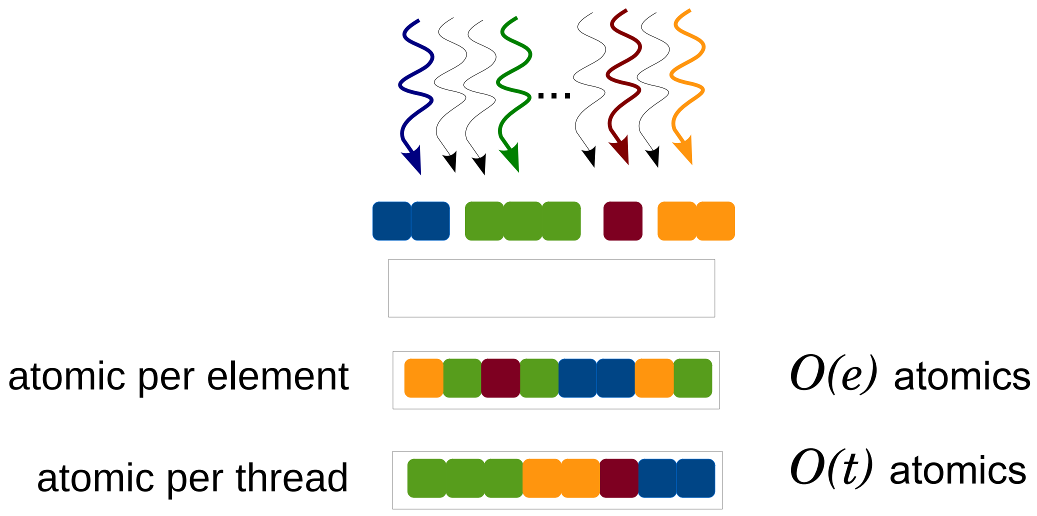


Data-driven: Hierarchical Worklist



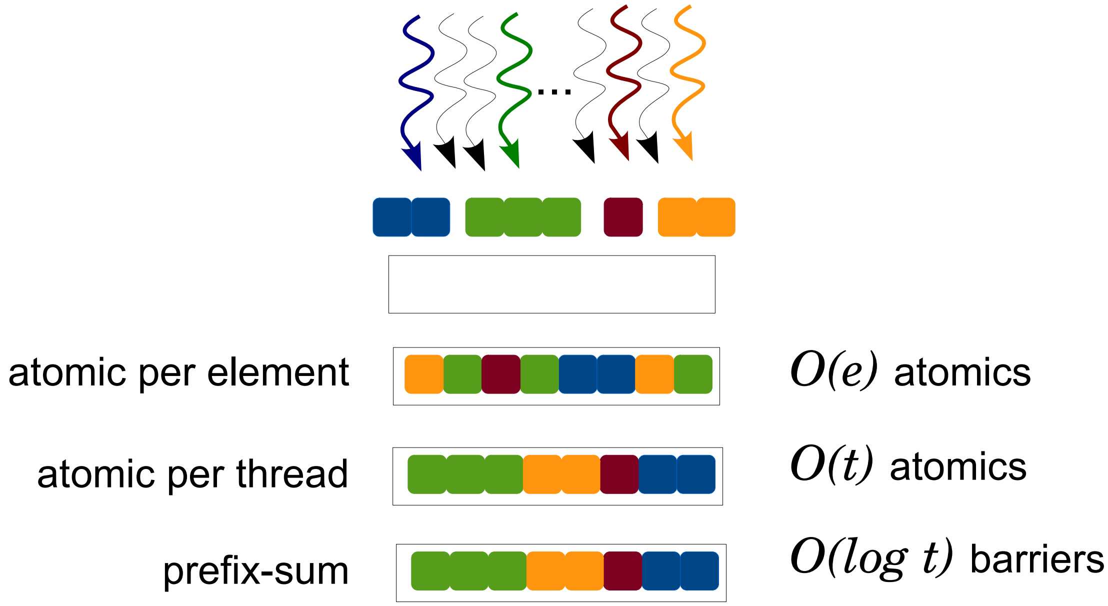
- Worklist exploits memory hierarchy
- Makes judicious use of limited on-chip cache

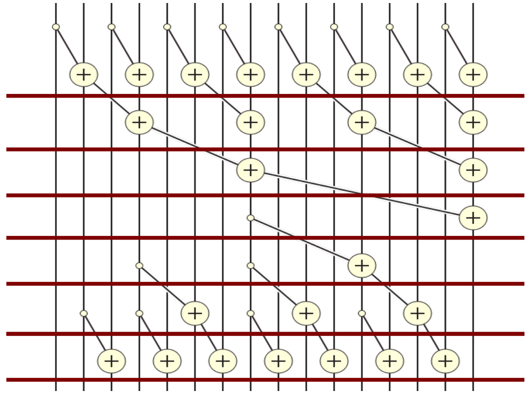
Data-driven: Work Chunking



- Reserves space for multiple work-items in a single atomic
- May reduce overall synchronization

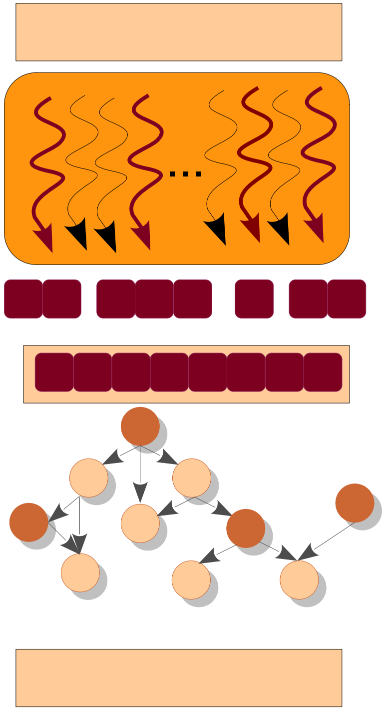
Data-driven: Atomic-free Worklist Update





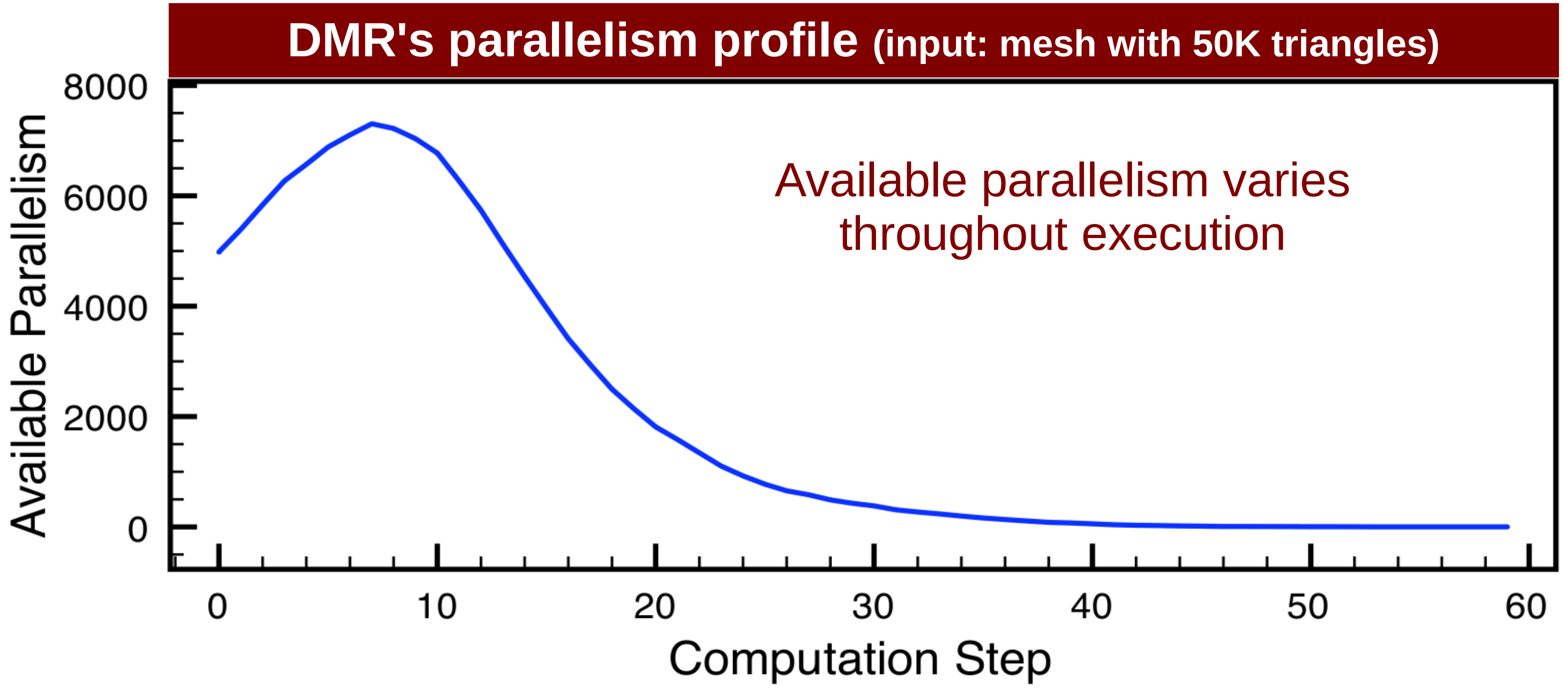
Data-driven: Work Donation

```
donate_kernel {
    shared donationbox[...];
    // determine if I should donate
    --barrier--

    // donate
    --barrier--

    // operator execution

    // empty donation box
}
```



- Work-donation improves load balance

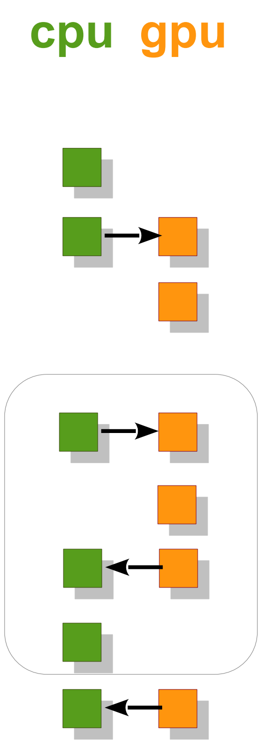
Data-driven: Variable Kernel Configuration

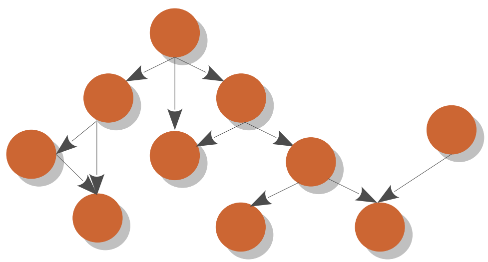


- Varying configuration improves work-efficiency
- It also reduces conflicts and may improve performance

Topology-driven: Base Version

main {
    read input
    transfer input
    initialize_kernel
    do {
     transfer **false** to *changed*
     operator(...)
     transfer *changed*
    } **while** *changed*
    transfer results
}

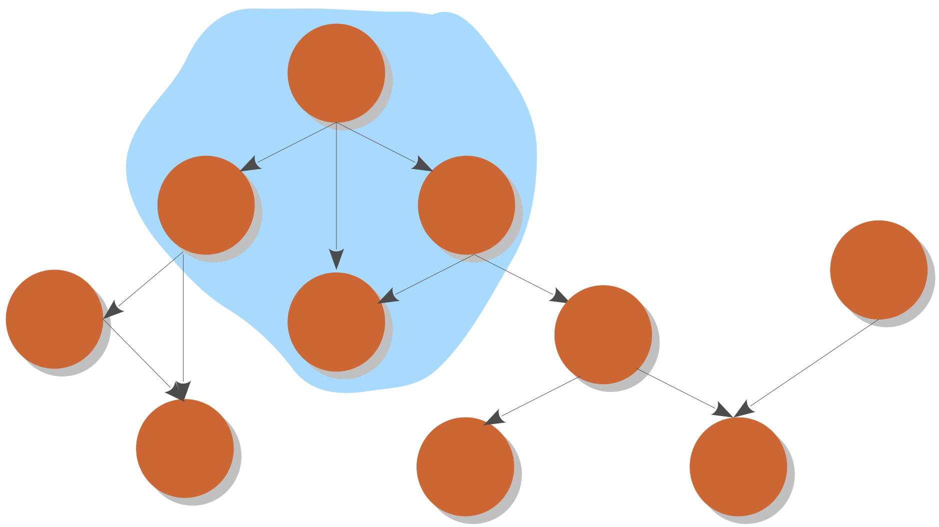




Topology-driven: Kernel Unrolling

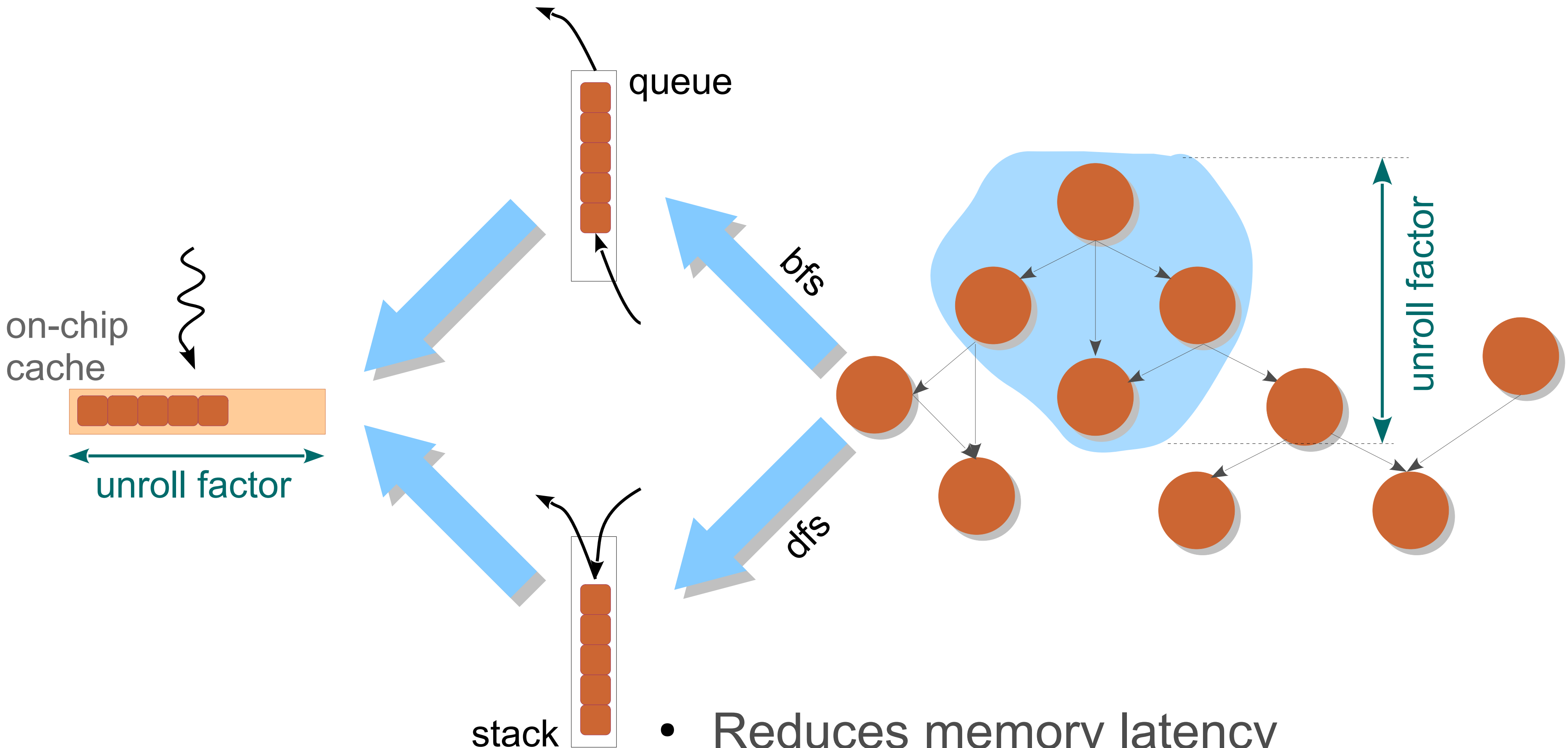
```
sssp_operator(src) {
    dsrc = distance[src]
    forall edges (src, dst, wt) {
        ddst = distance[dst]
        altdist = dsrc + wt

        if altdist < ddst
          distance[dst] = altdist
    }
}
```

Memory-bound kernel



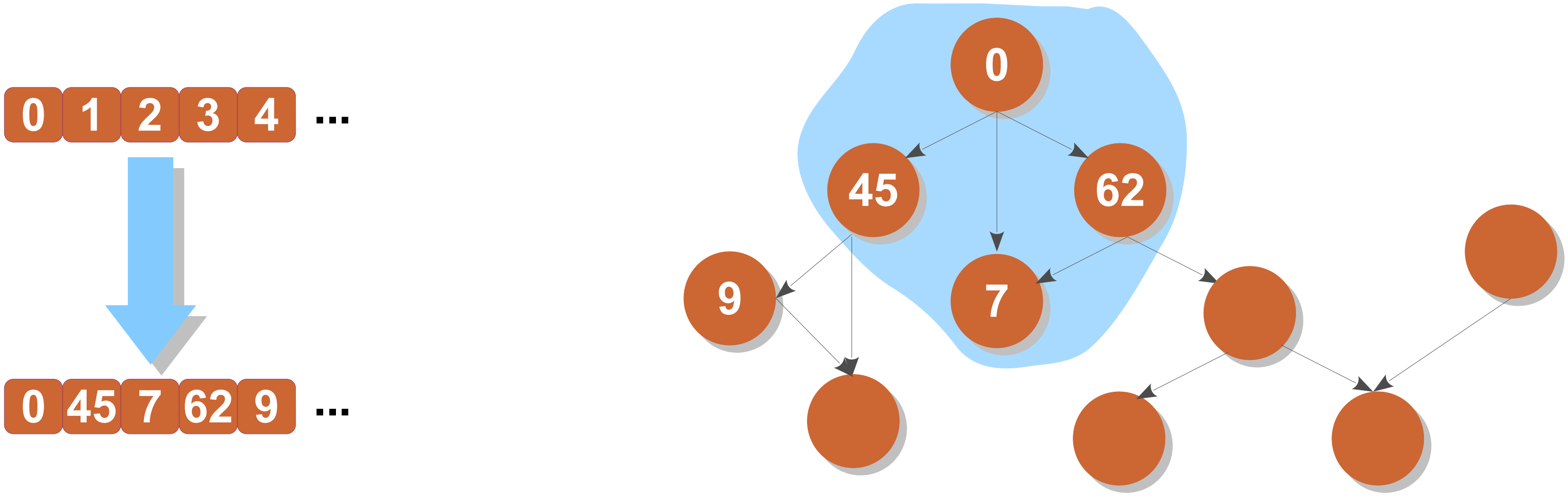
- Improves amount of computation per thread invocation
- Need to ensure absence of races
- Propagates information faster

Topology-driven: Exploiting Memory Hierarchy



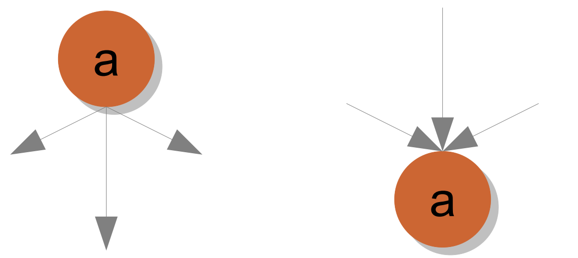
- Reduces memory latency
- Requires careful selection of unroll factor

Topology-driven: Improved Memory Layout

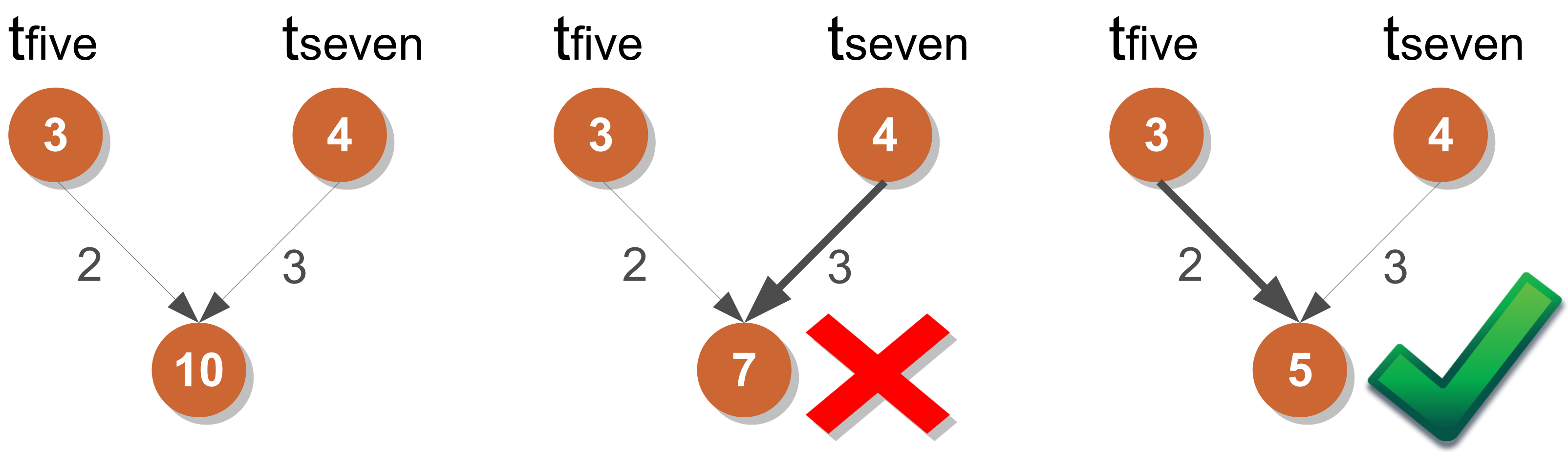


- Bring logically close graph nodes also physically close in memory
- Improves spatial locality

Improving Synchronization



push-based pull-based

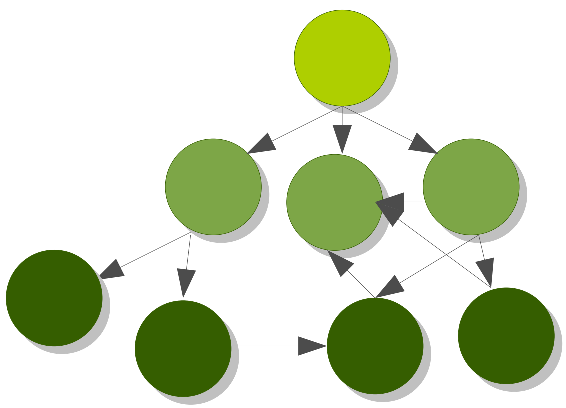


Atomic-free update

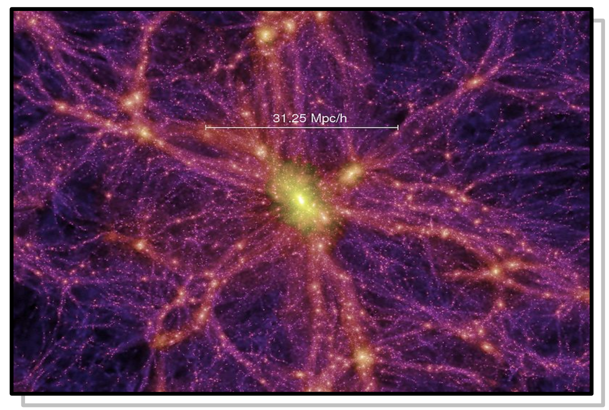
Lost-update problem

Correction by topology-driven processing, exploiting monotonicity

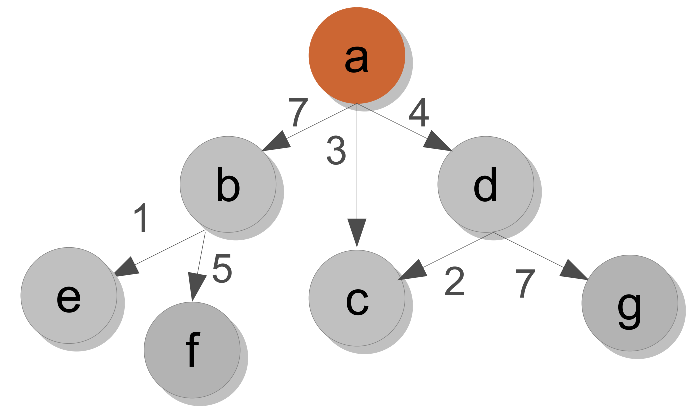
Irregular Algorithms on GPUs



Breadth-first search



Barnes-Hut n-body simulation

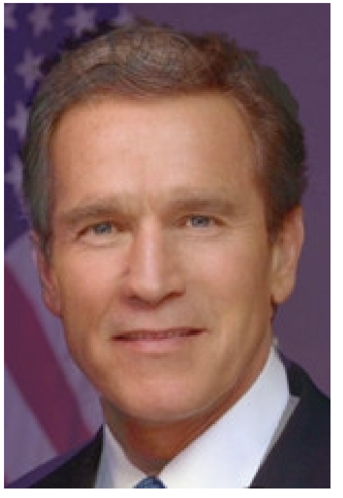


Single-source shortest paths

- Better memory layout
- Kernel unrolling
- Local worklists
- Improved synchronization

Application	Speedup
BFS	48
BH	90
SSSP	45

Identify the Celebrity

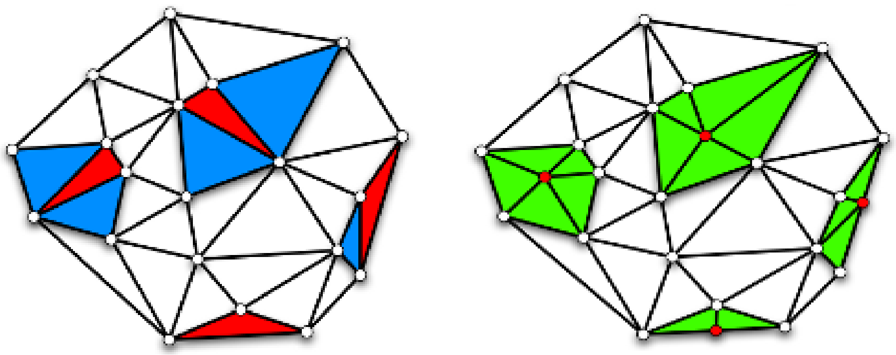


Source: wikipedia

What is a morph?

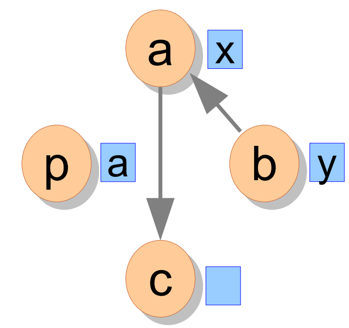
Source: wikipedia

Examples of Morph Algorithms

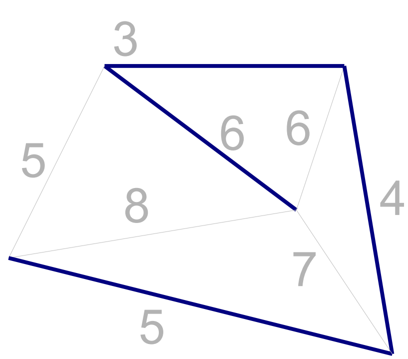


Delaunay Mesh Refinement

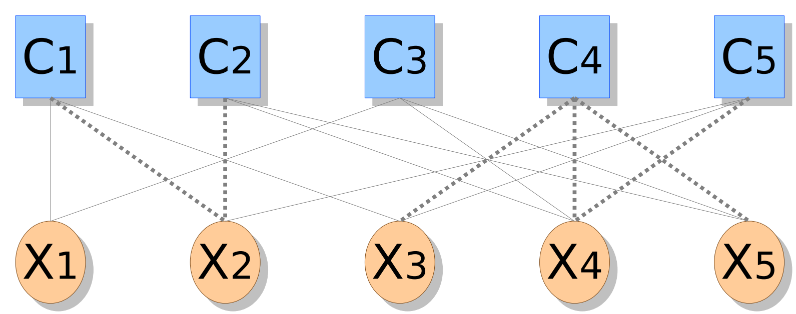
```
a = &x
b = &y
p = &a
*p = b
c = a
```



Points-to Analysis



Minimum Spanning
Tree Computation



Survey Propagation

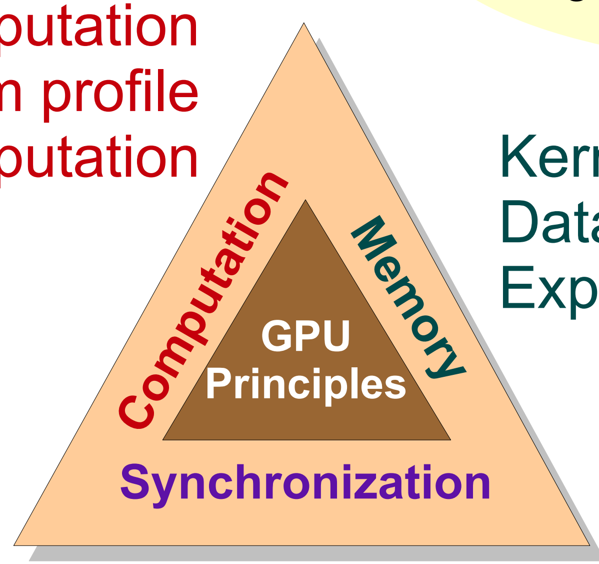
Challenges in Morph Algorithms

- Synchronization
  - locks are prohibitively expensive on GPUs
  - atomic instructions quickly become expensive
- Memory allocation
  - changing graph structure requires new strategies
  - memory requirement cannot be predicted
- Load imbalance
  - different modifications to different parts of the graph
  - work done per node changes dynamically
  - leads to thread-divergence and uncoalesced memory accesses

GPU Optimization Principles

Algorithm selection
Work sorting
Work chunking
Communication onto computation
Following parallelism profile
Pipelined computation

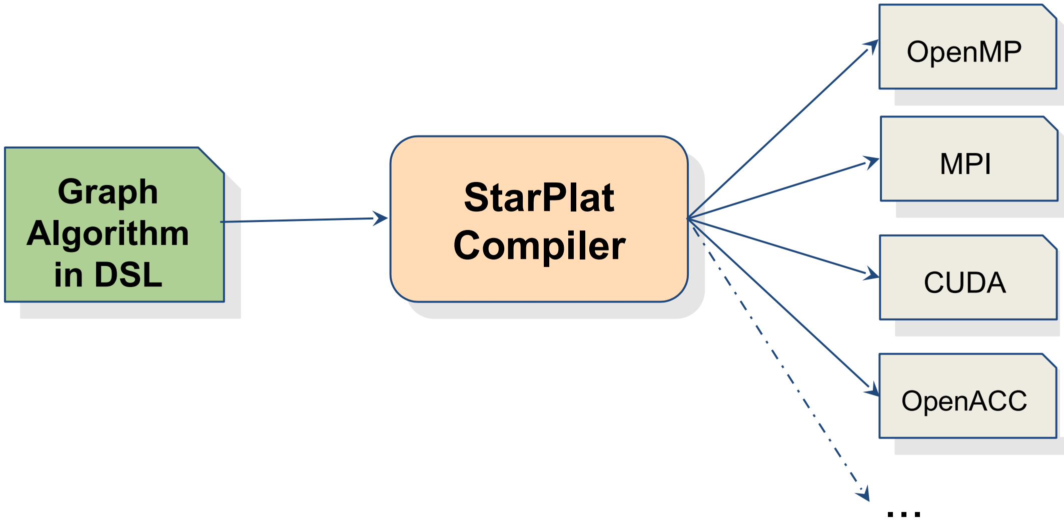
These optimization principles are **critical** for high-performing irregular GPU computations.



Kernel transformations
Data grouping
Exploiting memory hierarchy

Avoiding synchronization
Coarsening synchronization
Race and resolve mechanism
Combining synchronization

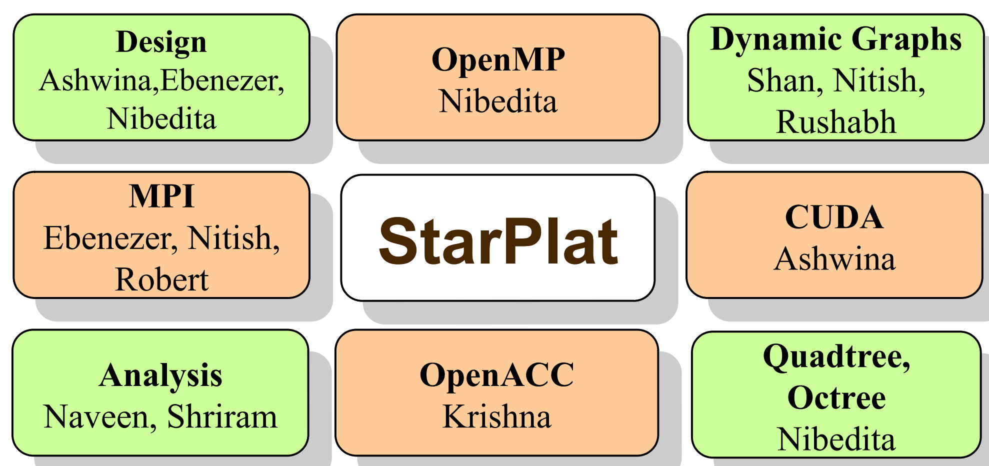
StarPlat: DSL for Parallel Graph Algorithms



- Generate code for different backends from the same algorithmic specification (OpenMP, MPI, CUDA, OpenACC, Sycl, OpenCL)
- Currently works with static as well as dynamic graphs
- Able to generate code for popular algorithms (SSSP, BC, PR, TC).
- In progress: complex algorithms, program analysis, multi-GPU processing, heterogeneous computing, …

Achievements

- Qualcomm Innovation Fellowship 2023
- StarPlat's Sycl backend featured at Intel website
- India Patent 432922
- Hardware access from AMD and Intel
- Small survey indicated productivity benefits



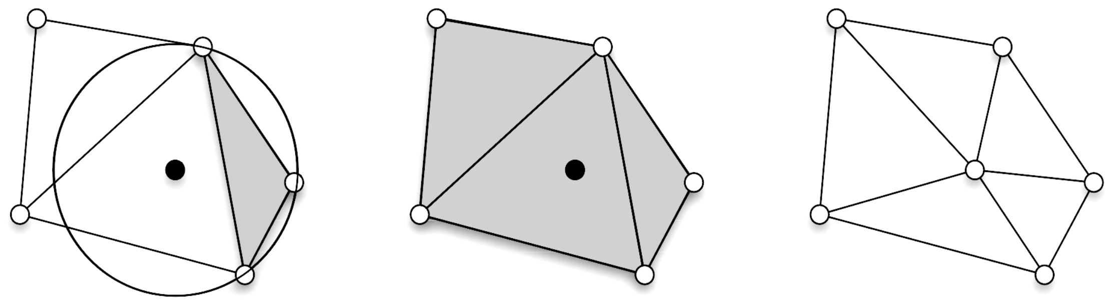
Exercises

- Find if true dependence exists for the loop.

```
for (ii = 0; ii < 10; ++ii) {
    a[2 * ii] = ... a[ii + 1] ...
    a[3 + ii] = ... a[5 * ii] ...
}
```

- Represent a graph as adjacency list on GPU.
- Represent an input graph in CSR format, and then convert it into a COO format.
- Write a kernel to count degrees of various vertices. Check finally that the sum equals the number of edges.
- Implement shortest path algorithm. Check your implementation against that in CUDA SDK.

Parallel Graph Algorithms



Rupesh Nasre.

rupesh@cse.iitm.ac.in

PACE
Programming Languages, Architecture and Compilers Education Laboratory

January 2025

FEEDBACK FOR GPU PROGRAMMING

I shall undergo Thread Divergence
As I launch my Feedback Kernel in a poetic way,
Thank you Sir, for being a Host par excellence
To me, a Thread from another Device, I say.

The Stream of your lectures was appealing,
Each day I was hooked, in Pinned Memory,
Awaiting your videos on the PCI Express bus each morning,
All your programs I did diligently cudaMemcpy.

Owing to Coalescing, I couldn't just watch one lecture,
But had to make Strided Access to subsequent ones too;
Till I watched them all -- one big Vector!
And so in Global DRAM, I want to thank you!

You patiently resolved all Race Condition
Of doubts and questions without making Lost Update,
You encouraged interaction and Synchronization,
In everyone's Shared Memory, you earned a place great!

As a Warp Representative from this class,
I perform an Inclusive Scan of all you taught,
You did Reduction of concepts like no one has;
atomicAdd(&likes, 1) to all your analogies' lot.

The Prefix Sum of my feedback is this:
You taught in a SIMD fashion,
With a Global Barrier to ensure no one did miss,
Thus, __all(Prof Rupesh is awesome) returns 1.

-- Ullas Aparanji, IISc

What did you learn?

Satya Bhagavan • 1st
Mtech CSE IIT Madras | Btech CSE IIT Indore | Ex - Algorithm Developer @ KLA
4mo •

Ever wondered how to sort numbers by simply sleeping? 😴
Sleep Sort the laziest algorithm out there!

Here is how it works:
1. Spawn a thread for each number in your list.
2. Each thread takes a nap proportional to its number's value.
3. As threads wake up, they print their numbers in order.

It's the only sorting method where procrastination is the key to success! 🕰

Disclaimer: Not recommended for actual use unless you have time to kill and a sense of humor.

#multithreading #algorithm #threads #os #SleepSort