The 道(Tao) of Parallelism in Algorithms

Zhuoran Shi

Exploiting parallelism in irregular algorithms

Parallel programming needs new abstractions

- dependence graphs are inadequate: cannot represent parallelism in irregular algorithms

New abstraction:

- operator formulation
- data centric abstraction

(regular algorithms become special case)

Key insights

- amorphous data parallelism (ADP) is ubiquitous
- TAO analysis: structure in algorithms
    - use TAO structure to exploit ADP efficiently

Inadequacy of static dependence graphs

Node: computations

Ege: dependeces between computations

> computations that are not ordered by the transitive closure of the dependence relation can be executed in parallel.

> dependences between computations in irregular algorithms are functions of runtime data values, so they cannot be rep- resented usefully by a static dependence graph.

Inadequacy of static dependence graphs - Example

Delaunay mesh refinement
- Don’t care non-determinism
- final mesh depends on order in
- which bad triangles are processed

Data structure: graph
- nodes: triangles
- edges: triangle adjacencies

Parallelism
- triangles with disjoint cavities can be processed in parallel
- parallelism depends on runtime values
- static dependence graph cannot be generated
- parallelization must be done at runtime

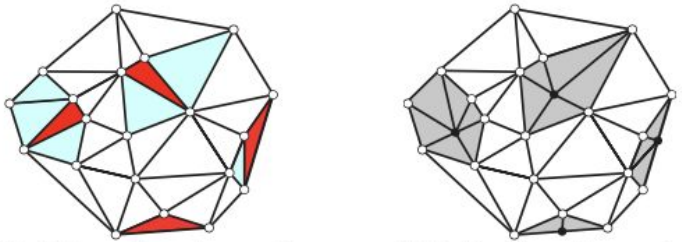


Figure 1. Fixing bad triangles

example: Delaunay Mesh Refinement (DMR)

Operator formulation of algorithms

> algorithm is viewed in terms of its action(operator) on data structure

- active elements
- neighbourhoods: nodes that might be read/written
- ordering

> Amorphous data parallelism (ADP)

- process active nodes in parallel, subject to neighborhood and ordering constraints
- how do we exploit ADP

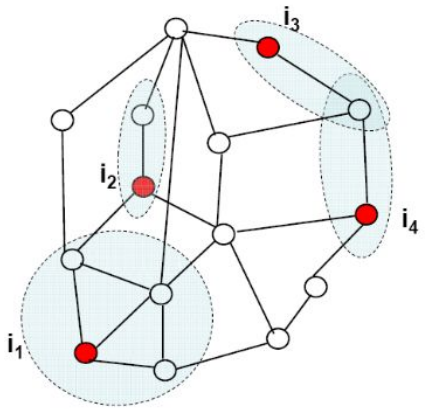


Figure 4. Active elements and neighborhoods

Tao-analysis of algorithms

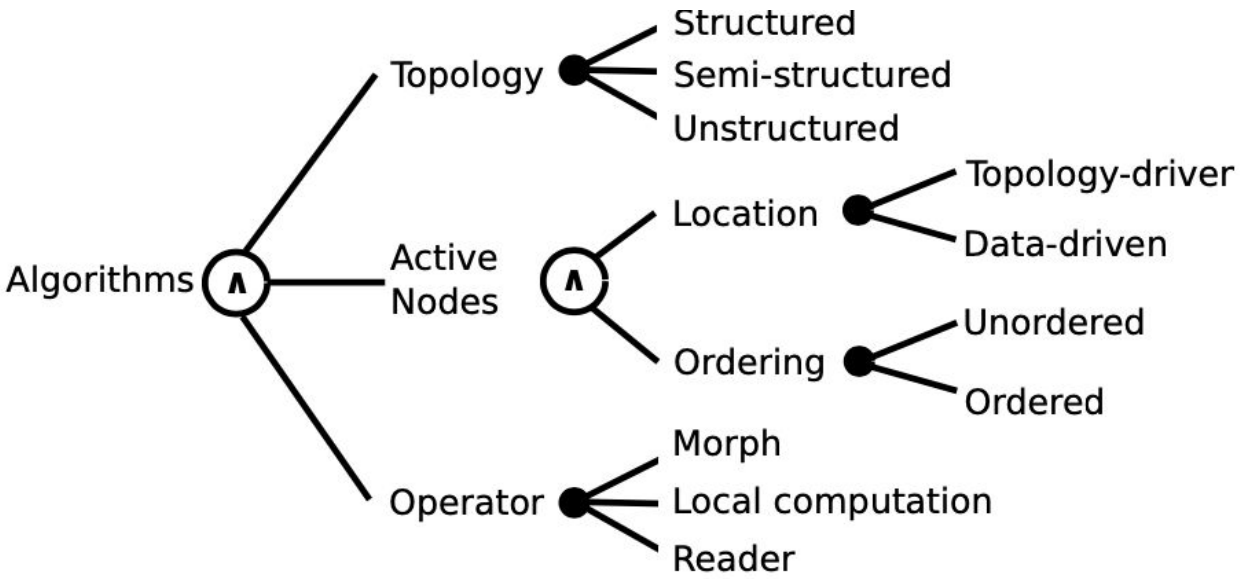


Figure 6. Structural analysis of algorithms

Amorphous data-parallelism - Implementation Strategy

- Baseline: speculative parallel execution
- Exploiting structure to reduce overheads: is it a cautious algorithm
- Exploiting structure for coordinated scheduling
  - autonomous（baseline implementation）
    - uncoordinated: online conflict detection, rollback for correct execution
  - coordinated (eliminating speculative overheads)
    - coordinated: only non-conflicting iterations are scheduled for parallel execution

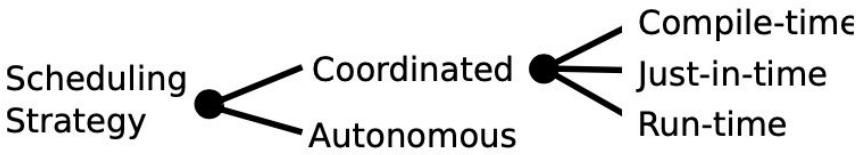


Figure 9. Scheduling strategies

Case studies of algorithms: tao-analysis + implementation strategy

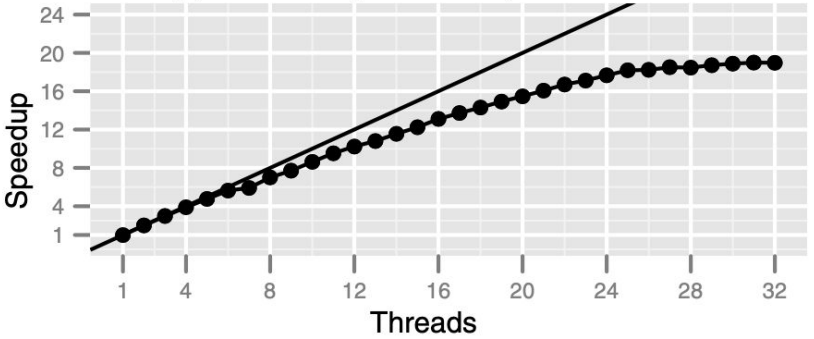
- Operator: Morph
  - refinement
  - coarsening
    - edge contraction
    - node elimination
    - sub-graph contraction
  - general morph
- Operator: Local computation
  - topology-driven
  - data-driven
- Operator: Reader

Case studies of applications

Barnes-Hut n-body simulation: four phases, each with a different structural classification

1. Tree-build:
   a. topology: tree
   b. operator: refinement morph
   c. active nodes: topology-driven and unordered
2. Summarize:
   a. topology: tree
   b. operator: local computation
   c. active nodes: topology-driven and ordered
3. Force-computation:
   a. topology: tree
   b. operator: reader
   c. active nodes: unordered
4. Force update:
   a. topology: set
   b. operator: local computation
   c. active nodes: topology-driven and unordered

(a) Parallelism profile for 10,000 bodies



(b) Speedup for 1,000,000 bodies; sequential runtime is 122.2 s.

Extensions

1. Nested amorphous data-parallelism
    a.
2. Pipeline parallelism
    a.
3. Task parallel execution
    a.

Conclusion

- to do