How Much Parallelism is There in Irregular Applications?

- Problems needed to solve:
  - In many programs, amorphous data-parallelism cannot be uncovered using static techniques, and its exploitation requires runtime strategies such as optimistic parallel execution.
  - The amount of amorphous data-parallelism in irregular programs varies in complex ways during the computation
  - How much amorphous data-parallelism exists in irregular programs?

- Solution provided by this paper
  - The paper presents a tool called ParaMeter that can determine how much amorphous data-parallelism is latent in irregular programs
    - ParaMeter can measure the amount of amorphous data-parallelism at different points in the execution of an algorithm.

Previous work

- Data parallel in regular programs
  - The data parallel in regular programs rises when an operation is performed on disjoint subsets of data elements of vectors and matrices.
  - Thanks to several decades of research, we now have powerful tools based on integer linear programming for finding and exploiting data-parallelism in regular programs
- In contrast, we understand relatively little about the patterns of parallelism and locality in irregular programs.
- This paper describes a tool called ParaMeter that uses instrumented execution to generate parallelism profiles of irregular programs.

Note: This paper did not talk about other techniques for producing parallelism profile of irregular programs, because this paper's tool, ParaMeter, is the first one for that. There is also no the state-of-the-art techniques for measuring the amount of data parallel in irregular programs. Therefore, on next slide I will talk the state-of-the-art application in the relevant field (data parallelism).

The current state-of-the-art application: Data parallelism in deep learning

- Background
  - Deep learning neural networks have achieved state-of-the-art results in prediction task, such as image classification, speech recognition.
  - The size of dataset is increasing dramatically.

- Data parallel in neural Networks
  - Parallelism enables us to harness the unprecedent amount of computation.
    - Distributing training examples across different processors and computing updates to the neural network in parallel.
  - For the most common neural network training algorithms (e.g., synchronous stochastic gradient descent), the scale of data parallelism corresponds to the number of training examples used to compute each update to the neural network.

Details on the implementation of ParaMeter

- ParaMeter can generate parallelism profiles for data parallel loops, measure parallelism intensity and model constrained parallelism.
  - ParaMeter generates parallelism profiles by simulating the execution of a program on an infinite number of processors.

- Each computation step can be thought of as two phases:
  - Inspection phase: Generating the computation graph from the current state of the worklist and identifies a maximal independent set of elements to execute.
  - Execution phase: Executing that independent set and sets up the worklist for the next step.

Details on the implementation of ParaMeter

- Unordered loops
  - Choosing elements at random from the worklist → Executing them speculatively → Discarding those elements that conflict with previously executed elements
  - For example, Delaunay mesh refinement, Delaunay triangulation.

- Ordered loops
  - The parallel execution model for ordered loops requires that iterations appear to complete in order.
  - Less important than unordered loops, so we only talk unordered loops here.

```
1  Set<Element> wl; //worklist of elements
2  wl = /* initialize worklist */
3  while (!wl.empty()) {
4    Set<Element> commitPool;
5    Set<Element> new_wl;
6    for (Element e : wl) { //random order
7      if (specProcess(e)) { //speculatively execute
8        new_wl.add(newWork(e)); //add new work
9        commitPool.add(e); //set e to commit
10     } else {
11       new_wl.add(e); //retry e in next round
12       abort(e);  //roll back e
13     }
14   }
15   for (Element e : commitPool) {
16     commit(e); //commit execution
17   }
18   wl = new_wl; //move to next round
19 }
```

Figure 1. ParaMeter pseudocode for unordered loops

Results for Delaunay mesh refinement

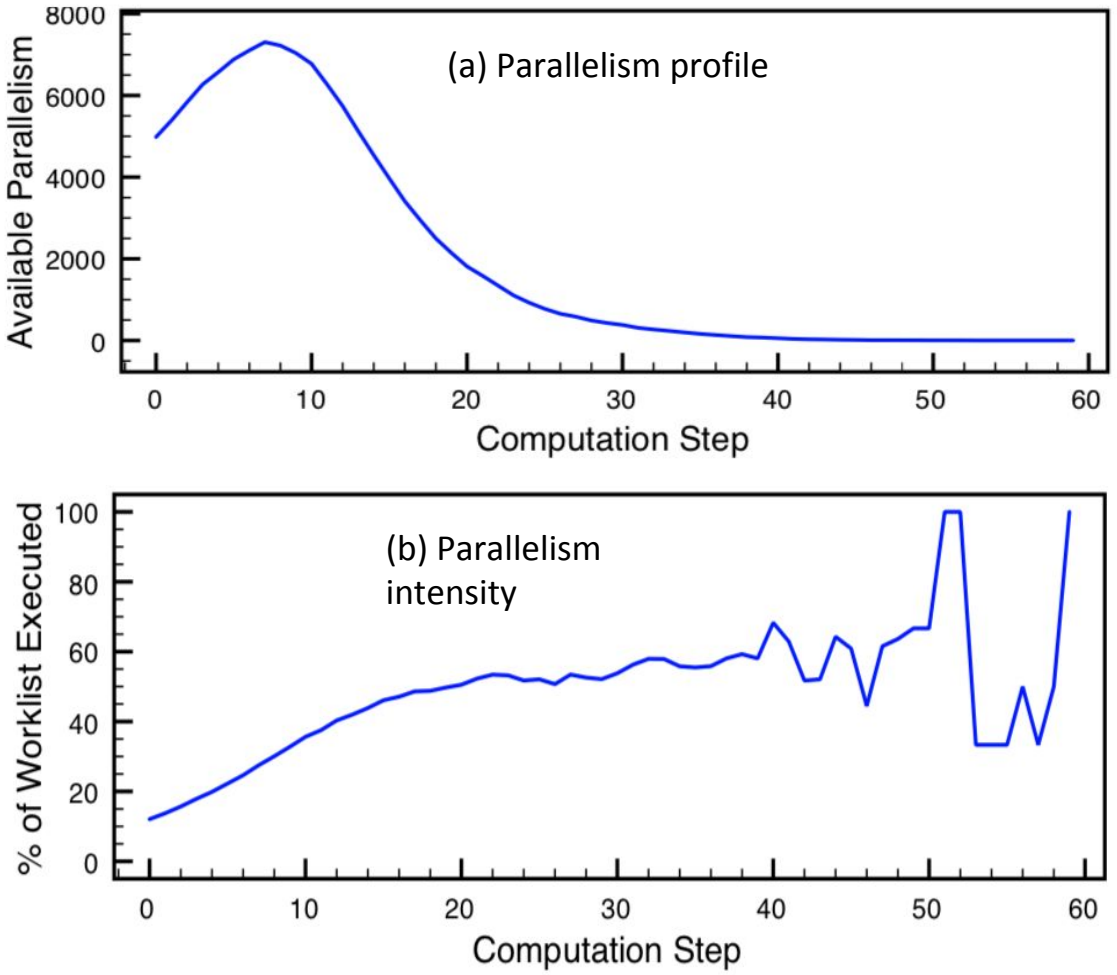

Figure 2. Parallelism metrics for Delaunay mesh refinement.

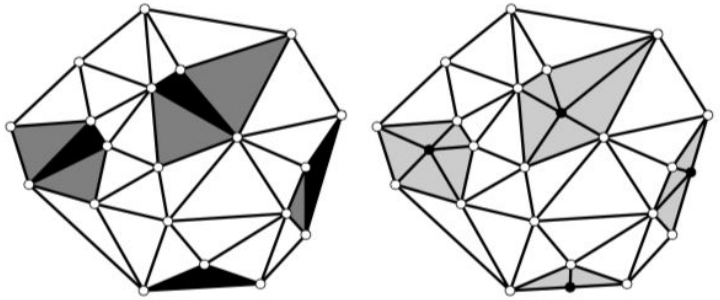

Figure 3. Mesh refinement

- In the application of Delaunay mesh refinement, the available parallelism increases initially and then drops when the computation starts to run out of work, while the probability of conflicts decreases as the computation progresses.
- Reason for the parallelism behavior: As the application progresses, the graph becomes larger. However, the average size of a cavity remains the same, regardless of the overall size of the mesh. Hence, as the graph becomes refined, the likelihood that two cavities overlap decreases.

Results for Delaunay triangulation

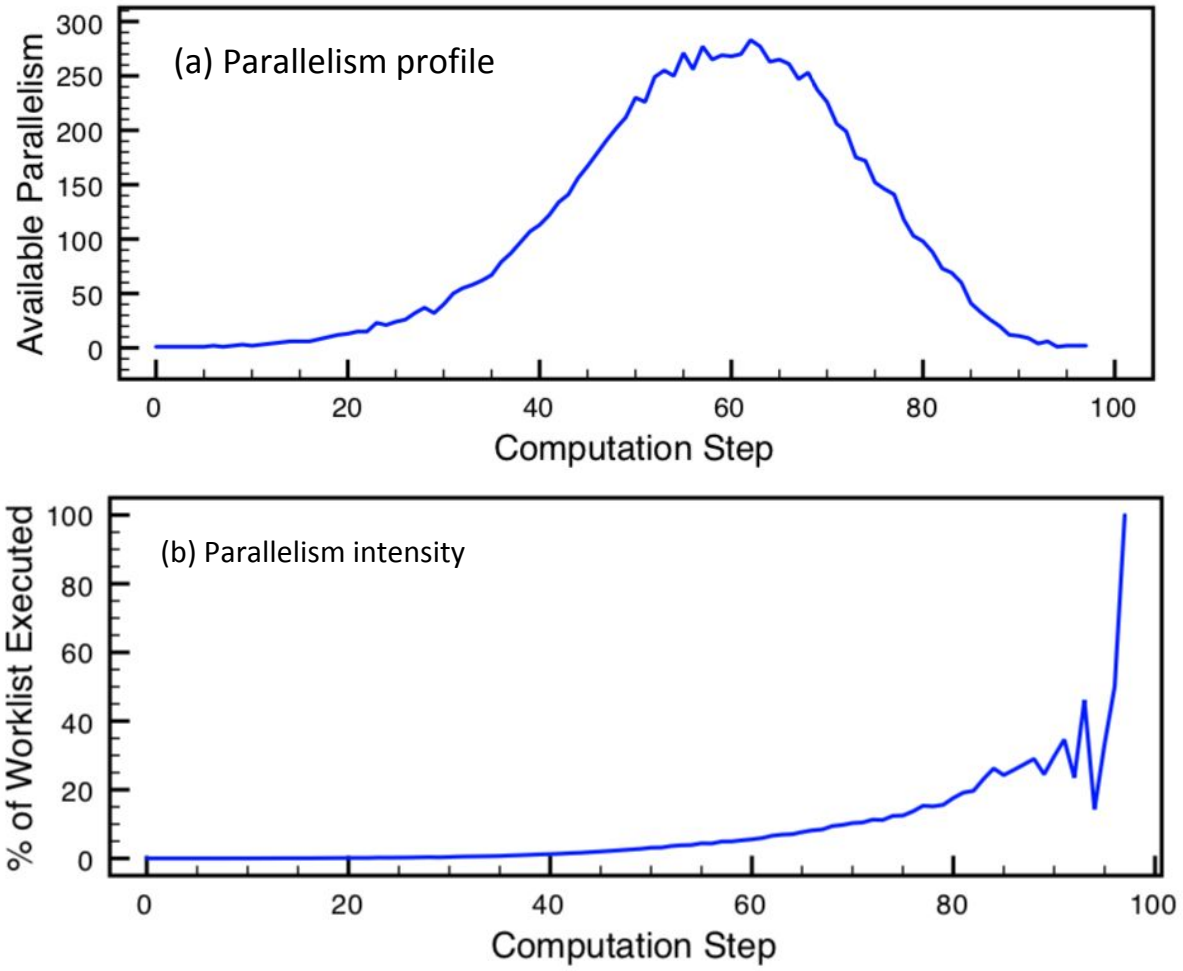

Figure 4. Parallelism metrics for Delaunay triangulation.

- Initially, there is no parallelism (only one element can be executed per round), but as execution progresses, the amount of parallelism increases, and then decreases as the worklist is drained.

- The parallelism profile of Delaunay triangulation has the same characteristic bell shape as that of Delaunay mesh refinement. The parallelism intensity demonstrates the increase in parallelism that we expect of refinement codes, whereas the parallelism profile reflects a lack of work in later rounds.

Note: There are more experiments in the paper, but those experiments show the similar thing (in x and y contribution). Therefore, we only consider the two experiments in the results.

My opinion and conclusion

- The profiles produced by ParaMeter is a good approach to guide the design of parallel algorithms.
  - The paper used the parallelism profile of available parallelism vs computation steps, which is clear and straightforward. The parallelism profile is a good example for showing how much parallel is happening in different computation steps, which can be used for comparing different algorithms.
- The tool, ParaMeter, provides a good idea about how to evaluate the parallel capability of an algorithm.
  - For example, this parameter (the parallelism intensity) clearly describes how difficult to make scheduling at different computation steps.
- Using the computation step in x-axis instead of only showing total execution time makes others better understand how parallel computation performs at different execution periods.

Opinion: How does the technique in this paper potentially contribute to data parallel in deep learning?

- The data parallel profiles generated by ParaMeter can be used to analyze the data parallel extent at different computation steps in deep learning.

- Potentially check the performance of optimized deep learning algorithms.
  - If a parallel implementation approaches the speedups predicted by the parallelism profile, continuing optimizing the implementation may not be necessary.

- Estimate hardware requirements in deep learning.
  - How performance will be affected by changing the number of processors.