Fine-grained parallelism in probabilistic parsing with Habanero Java

Matthew Francis-Landau[1], Bing Xue[2], Jason Eisner[1] and Vivek Sarkar[2]

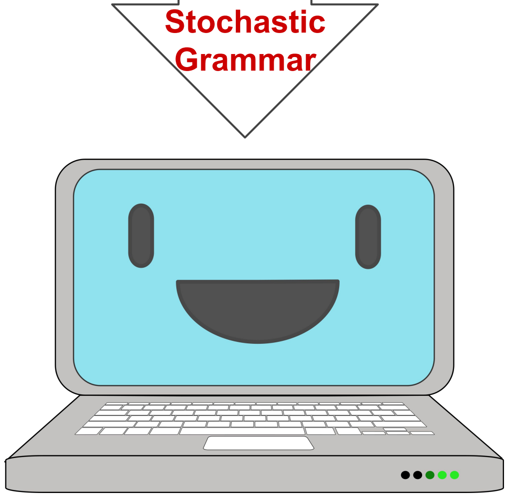
Johns Hopkins University[1] and Rice University[2]

This material is based upon work supported by the National Science Foundation under Collaborative Grants No. 1629564 and 1629459

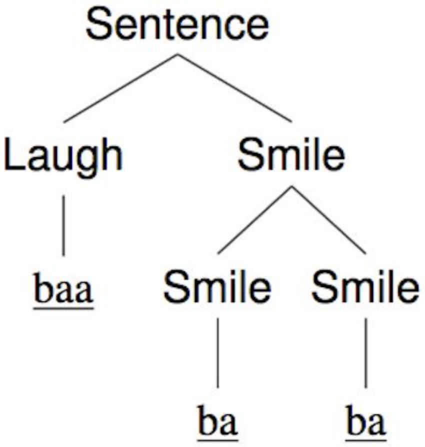
Probabilistic Parsing

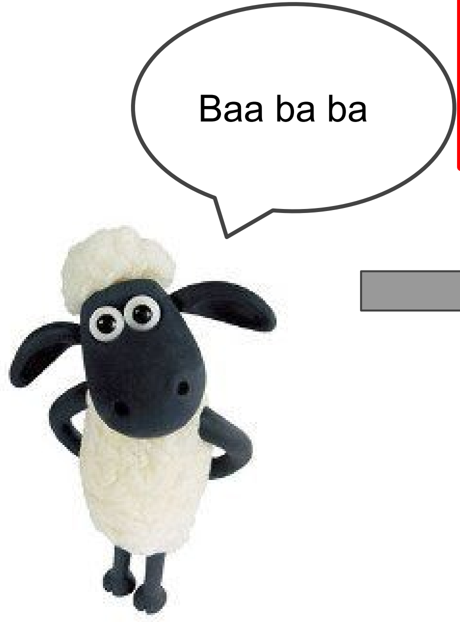
- Core problem in Natural Language Processing (NLP)
  - Computationally expensive
  - Load-balancing is hard at fine-grained level
- Similar programming patterns appear in many machine learning (ML) algorithms
  - Parallelizing probabilistic parsing algorithms can be a proxy task for parallelization of a large set of ML algorithms

Production	Prob
Sentence → Laugh Smile	0.5
Laugh → Laugh Laugh	0.1
Laugh → *baa*	0.9
Smile → Smile Smile	0.2
Smile → *ba*	0.8
⋮	⋮

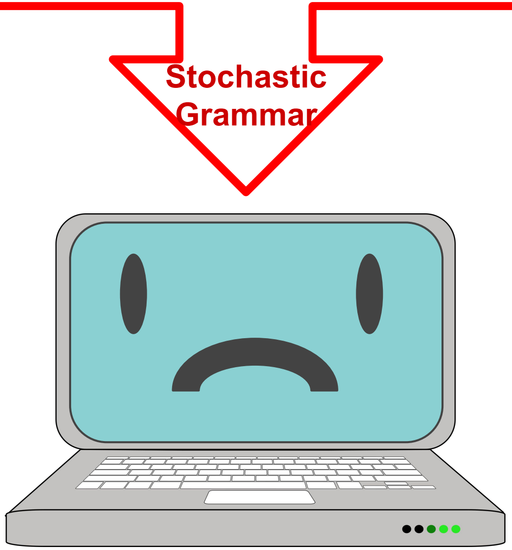


Production: a *rewrite rule* specifying a symbol substitution that can be recursively performed to generate new symbol sequences (from *Wikipedia*)

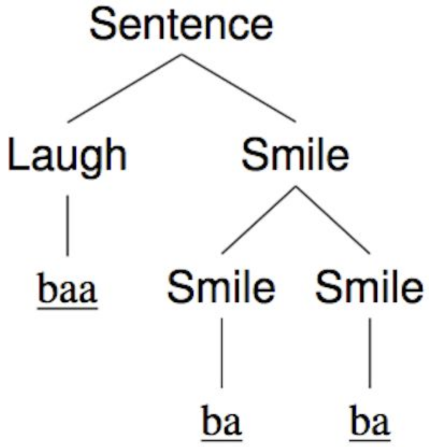




Production	Prob
Sentence → Laugh Smile	0.5
Laugh → Laugh Laugh	0.1
Laugh → *baa*	0.9
Smile → Smile Smile	0.2
Smile → *ba*	0.8
⋮	⋮



Millions of productions!



Probabilistic Parsing

Algorithm 1 Typical CKY algorithm for probabilistic parsing

for $w \in [2, N]$ **do** ▷ *width of constituent*
  for $i \in [0, N - w]$ **do** ▷ *starting location*
    $k \leftarrow w + i$ ▷ *ending location*
    for $x \in T$ **do** ▷ *nonterminals at location*
      $A[x, i, k] = \max_{\substack{i<j<k \\ y,z \in T}} A[y, i, j] \cdot A[z, j, k] \cdot P(x \to y\ z)$

CKY expressed declaratively in Dyna[1]

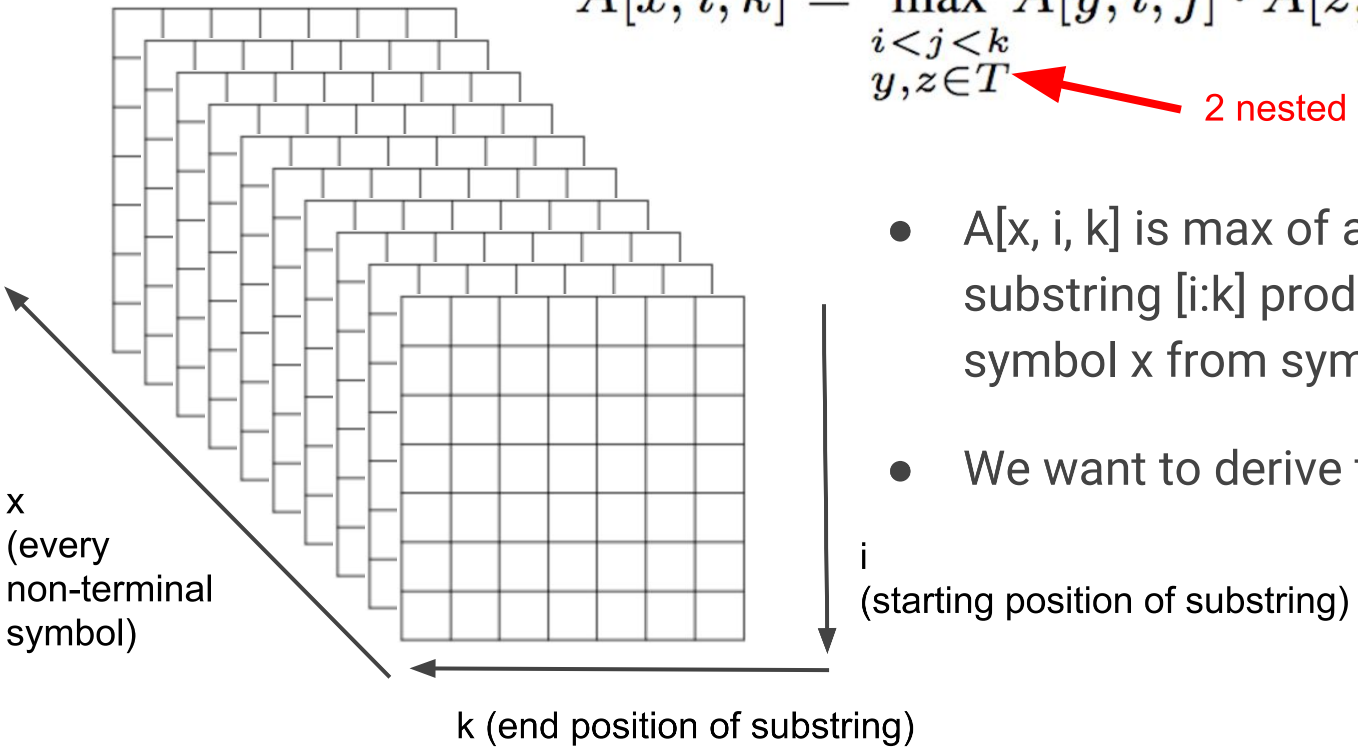
```
a(X,I,K) max= word(W,I,K) * rule_prob(X,W).
a(X,I,K) max= a(Y,I,J) * a(Z,J,K) * rule_prob(X,Y,Z)
goal = a("Sentence", 0, n).
```


[1] J. Eisner and N. W. Filardo, “Dyna: Extending Datalog for modern AI,” in Datalog Reloaded, ser. Lecture Notes in Computer Science, O. de Moor, G. Gottlob, T. Furche, and A. Sellers, Eds. Springer, 2011, vol. 6702, pp. 181–220, longer version available as tech report. [Online]. Available: http://cs.jhu.edu/~jason/papers/#eisner-filardo-2011

Probabilistic Parsing

$$A[x,i,k] = \max_{\substack{i<j<k \\ y,z\in T}} A[y,i,j] \cdot A[z,j,k] \cdot P(x \to y\ z)$$

2 nested loops through all productions



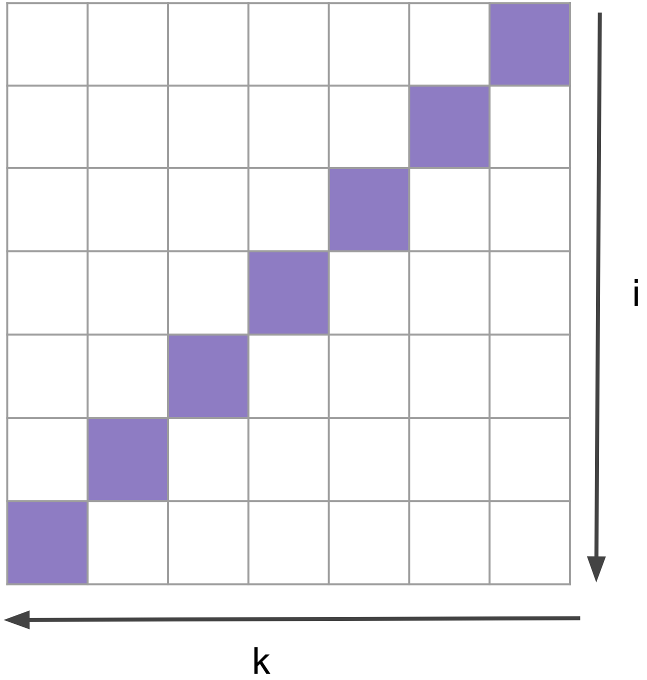
- A[x, i, k] is max of all probabilities of substring [i:k] producing non-terminal symbol x from symbols y and z
- We want to derive the *Sentence* symbol

Probabilistic Parsing

x = *Sentence*

$$A[x,i,k] = \max_{\substack{i<j<k \\ y,z\in T}} A[y,i,j] \cdot A[z,j,k] \cdot P(x \to y\ z)$$

2 nested loops through all productions



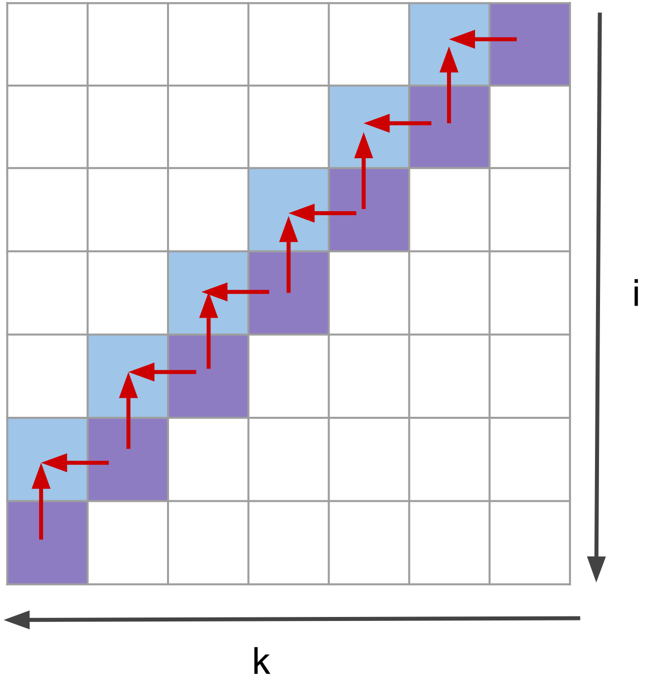
- Fill in each cell for substrings of size 2

Probabilistic Parsing

x = *Sentence*

$$A[x,i,k] = \max_{\substack{i<j<k \\ y,z \in T}} A[y,i,j] \cdot A[z,j,k] \cdot P(x \rightarrow y\ z)$$

2 nested loops through all productions

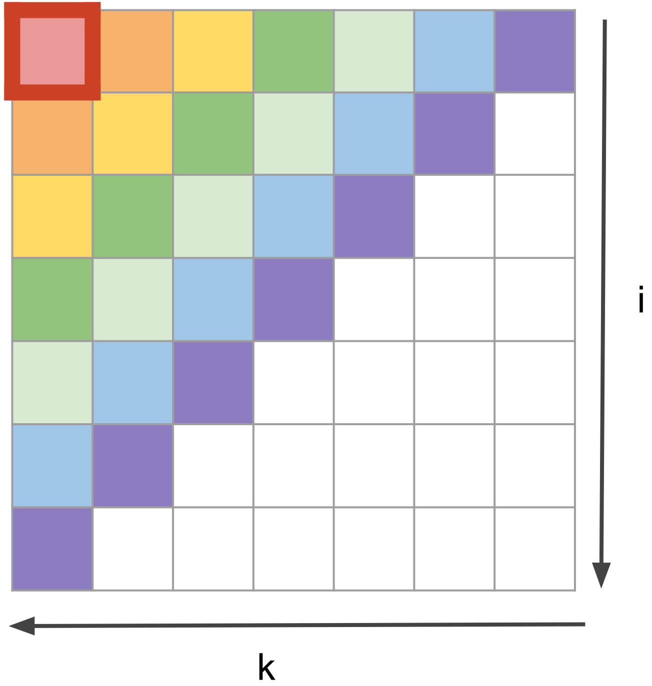


- Fill in each cell for substrings of size 3 based on values from substrings of size 2 according to the production rules

Probabilistic Parsing

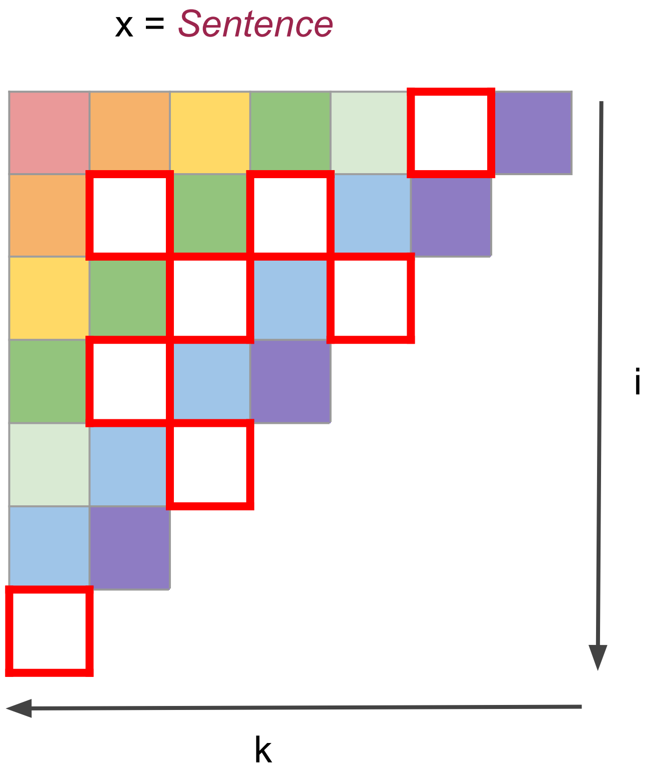
x = *Sentence*

$$A[x, i, k] = \max_{\substack{i<j<k \\ y,z\in T}} A[y, i, j] \cdot A[z, j, k] \cdot P(x \rightarrow y\ z)$$



- The parse tree with the largest probability ends up at position (*Sentence*, 0, N)

Probabilistic Parsing



More realistically, probabilistic parsing is an irregular application ...

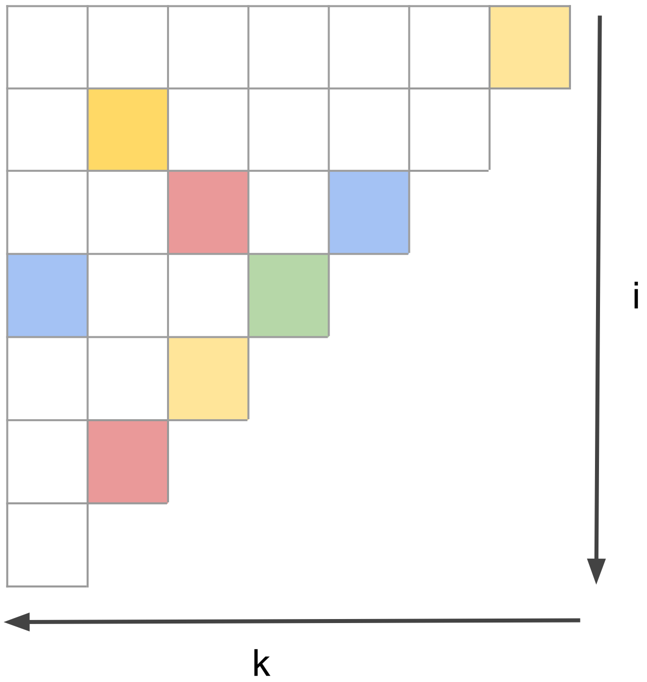
- Not all cells get filled
  - Some productions do not exist for x
  - Lower half of the matrix is unused
- Not all cells take the same amount of time to fill
  - Number of possible productions varies for each substring
- **Most work wasted**
  - Most cells do not contribute to final result (the upper left corner) because their contributions are ultimately beaten in some “max” operation

Alternative to CKY: Agenda Parsing

- Worklist version of CKY parsing (or an approximation)
    - Each update to a cell is a work item, and put them into an agenda
    - Prioritizes updates with higher probability
- **Stop early** and **save work** given "*good enough*" parse tree
    - Eliminates much unneeded computation in CKY
    - Reaches "good enough" parse tree faster with its greedy approach
    - If the priority function is an admissible A* heuristic, the algorithm becomes exact
- A generalized Dijkstra's Algorithm
    - Can be applied to machine learning algorithms similar to probabilistic parsing
    - A "meta-algorithm" for dynamic programming schemes

Cell-level Parallel Agenda Parsing

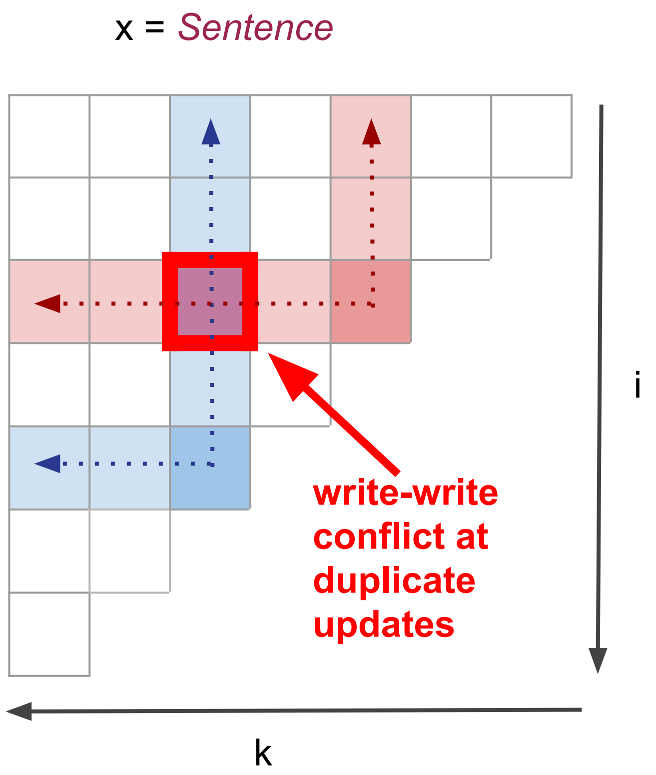
x = *Sentence*



- Need to process multiple agenda items (cell update) in parallel
  - use Java's `BlockingPriorityQueue` for thread-safe worklist and Habanero Java (HJLib)[2] parallel constructs for asynchronous processing
- Need to ensure total order of execution on agenda
  - Capture top *m* items on agenda to process in parallel

[2] V. Cave', J. Zhao, J. Shirako, and V. Sarkar, "Habanero-java: The new adventures of old x10," in Proceedings of the 9th International Conference on Principles and Practice of Programming in Java, ser. PPPJ '11. New York, NY, USA: ACM, 2011, pp. 51–61. [Online]. Available: http://www.cs.rice.edu/~vs3/PDF/hj-pppj11.pdf

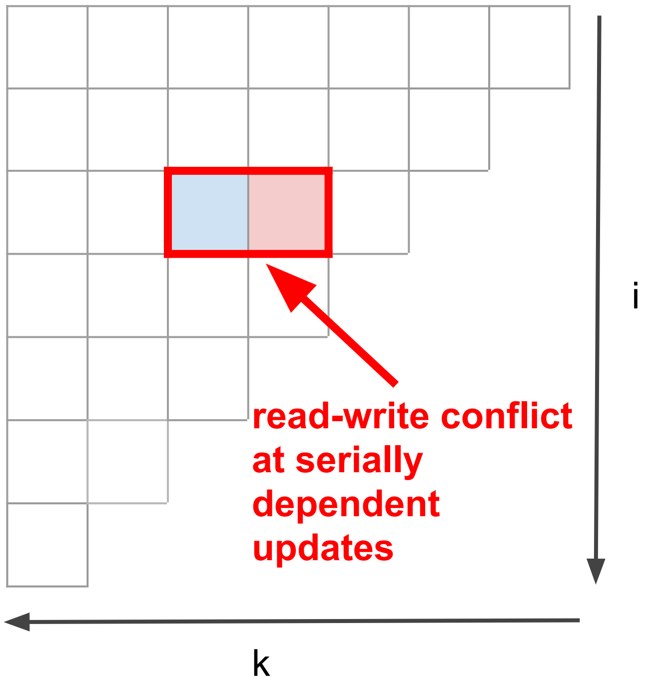
Cell-level Parallel Agenda Parsing



- Write-write conflict happens when two agenda items want to update the same cell
- Need to ensure atomic max operation on cell updates
  - Max operation implemented with CAS (Compare-And-Swap)

Cell-level Parallel Agenda Parsing

x = *Sentence*



- Two serially dependent cells can be updated at the same time
- <u>Need to ensure the most recent maximum value is considered</u>
  - An update to cell value will generate new update with **higher priority** on agenda

Parallel Agenda Parsing with Habanero Java[2]

```
class AgendaParser {
    …
    while(!agenda.isEmpty()){
        Collection<T> taskItems =
                         agenda.slice(0,m);
        forall(taskItems, (t)->{
                    process(t);
         });
         //implicit finish
    }
    …
}
```

- Treat all agenda items as individual asynchronous tasks
- Capture top *m* items on agenda to process in parallel
- HJLib `forall` construct creates asynchronous tasks for each item in a Collection with an implicit barrier

[2] V. Cave', J. Zhao, J. Shirako, and V. Sarkar, "Habanero-java: The new adventures of old x10," in Proceedings of the 9th International Conference on Principles and Practice of Programming in Java, ser. PPPJ '11. New York, NY, USA: ACM, 2011, pp. 51–61. [Online]. Available: http://www.cs.rice.edu/~vs3/PDF/hj-pppj11.pdf

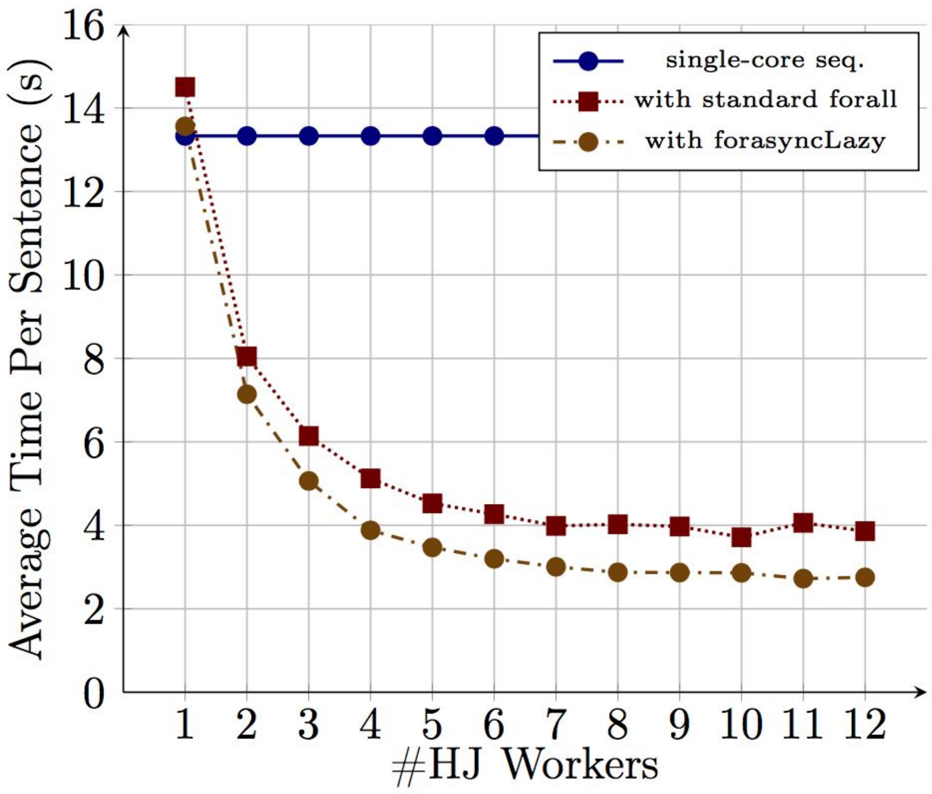
Parallel Agenda Parsing with Habanero Java[2]

```
public static <T> void forasyncLazy(...) {
  finish( () -> {
    for ( int i=0; i < numTasks; i++ ) {
      async( () -> {
        while (taskItems.hasNext()) {
          processBody.apply(
            taskItems.next()
          );
        });
      }
    });
}
```

- New API `forasyncLazy`(numTasks, taskItems, processBody)
  - numTasks - number of `async` processes to create
  - taskItems - an iterator as task generator
  - processBody - lambda expression to operate on an task item
- Agenda as taskItems always returns true for `hasNext`() until parse completes

[2] V. Cave', J. Zhao, J. Shirako, and V. Sarkar, "Habanero-java: The new adventures of old x10," in Proceedings of the 9th International Conference on Principles and Practice of Programming in Java, ser. PPPJ '11. New York, NY, USA: ACM, 2011, pp. 51–61. [Online]. Available: http://www.cs.rice.edu/~vs3/PDF/hj-pppj11.pdf

Experimental Results



- Extend the bubs-parser[3][4] code base's agenda parser
- 2.8GHz Westmere-EP computing nodes
  - 12 Intel Xeon X5660 processor cores
  - 48GB RAM per node
- 25 sentences
  - < 30 words per sentence
- Grammar with ~2 Million productions

[3] "bubs-parser." [Online]. Available: https://code.google.com/archive/p/bubs-parser/ [4] Adaptive Beam-Width Prediction for Efficient CYK Parsing Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark - ACL/HLT 2011, pages 440-449.

Conclusion and Future Work

- ~5x performance improvements due to parallelism without impairment on accuracy
- Methods applicable to general dynamic programming schemes
- Dyna language[1] provides high-level specification of DP schemes
- Our long-term goal is to support source-to-source compilation of Dyna programs into parallel HJ programs for multicore and distributed-memory parallelism

CKY expressed declaratively in Dyna[1]

```
a(X,I,K) max= word(W,I,K) * rule_prob(X,W).
a(X,I,K) max= a(Y,I,J) * a(Z,J,K) * rule_prob(X,Y,Z)
goal = a("Sentence", 0, n).
```

(Incomplete) HJLib code

```
Iterator<ChartCell> agendaItems = new Iterator<>(){
        public boolean hasNext() { return !doneParsing;}
        public T next() { return agenda.take(); }
}
finish( ()->{
        forasyncLazy(numTasks, agendaItems, (c) -> {
                        chartUpdate(c.i, c.k, c.x);
                        agendaUpdate(c, chart);
                }
        });
return chart.get("Sentence")[0][N];
```

References

[1] J. Eisner and N. W. Filardo, “Dyna: Extending Datalog for modern AI,” in Datalog Reloaded, ser. Lecture Notes in Computer Science, O. de Moor, G. Gottlob, T. Furche, and A. Sellers, Eds. Springer, 2011, vol. 6702, pp. 181–220, longer version available as tech report. [Online]. Available: http://cs.jhu.edu/~jason/papers/#eisner-filardo-2011

[2] V. Cave’, J. Zhao, J. Shirako, and V. Sarkar, “Habanero-java: The new adventures of old x10,” in Proceedings of the 9th International Conference on Principles and Practice of Programming in Java, ser. PPPJ ’11. New York, NY, USA: ACM, 2011, pp. 51–61. [Online]. Available: http://www.cs.rice.edu/~vs3/PDF/hj-pppj11.pdf

[3] “bubs-parser.” [Online]. Available: https://code.google.com/archive/p/bubs-parser/

[4] Adaptive Beam-Width Prediction for Efficient CYK Parsing Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark - ACL/HLT 2011, pages 440-449.

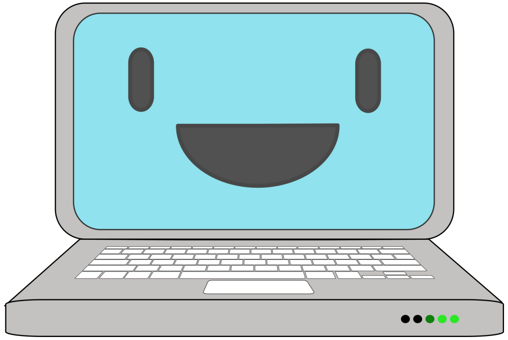
Thank you for your time!

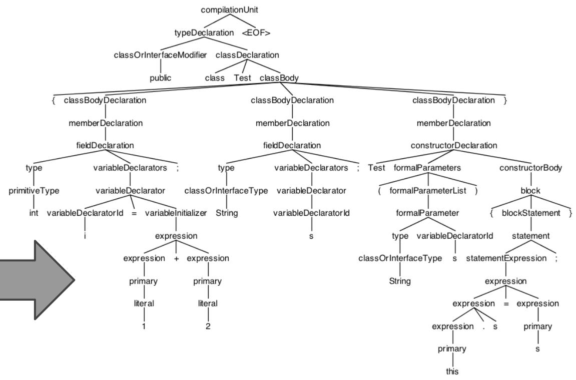
Questions?

Parsing Java Programs

Grammar: a set of production rules that describes how valid strings are formed according to a language’s syntax

Java Grammar

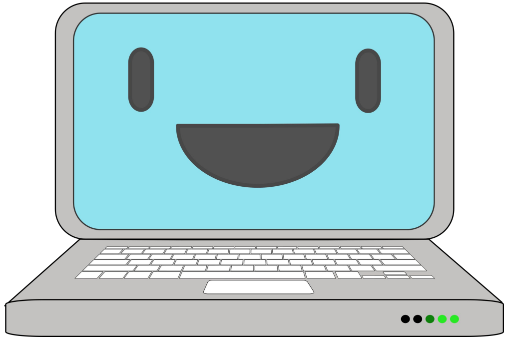


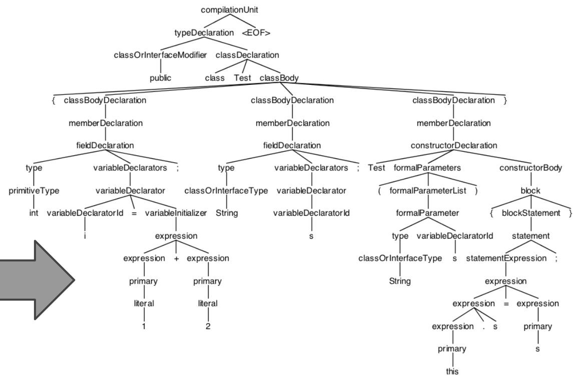


Parsing Java Programs

Deterministic grammar
Small number of grammar rules

Java Grammar





Parsing Natural Language