Graph Computing Paradigms

Peter M. Kogge

Definitions

- **Paradigm**: “a typical example or pattern of something; a model”
- **Programming Paradigm**: “a style, or “way,” of programming”
- **Execution Model**: “specifies how work takes place” (https://en.wikipedia.org/wiki/Execution_model)
  - what is an indivisible unit of work,
  - what are the constraints on the order in which those units of work take place
  - E.g. C
    - units = statements ended by “;”
    - Statements executed indivisibly, in order”
Aspects of Graph Computing Paradigms

- How to express graphs
- How to express computation on such graphs
- What is syntax of expressions
- What functions are builtin
- What is underlying execution model
- What are options for specifying parallelism
- What might a sample graph computation look like

Types of Paradigms

- **Languages**: complete self-contained programming language designed for graphs
- **Libraries**: packages callable from some conventional language
- **Systems**: combinations of languages, libraries, and specialized runtimes, especially for parallel systems
Languages

• Accumulo
• Cypher
• GraphLab
• GraQL
• Gremlin
• KEL
• Poplar
• SPARQL and RDF 34
• Trinity

Libraries

• GraphBLAS
• GraphChi
• GraphLab
• Parallel Boost Graph Library
• Stinger
• System G
Systems

- DisNet
- FlockDB
- GEMS
- Graph Engine
- Graphulo
- HyperGraphDB
- JENA
- Neo4j
- Pregel
- Powergraph
- GraphX, Scala, Spark

A Standardized Syntax

- **Terminal symbols**: basic characters from the language
  - expressed as the characters themselves,
  - with exception of when same as meta-symbols
  - in which case written with a "\" in front of them.
- **Nonterminal**: formal name of some subset of strings in language
- **Production Rule**: head -> body
  - head is nonterminal “name” of subset of valid strings
  - body is description of valid strings
- **Meta-symbols**: characters in a rule body that are part of rule, and not characters in language being described
**Meta Symbols Used in Body**

- `{}` surrounding a string: treat as if a single unit in terms of other syntax rules,
  - especially those using meta-symbols j, ?, +, and .
- `|` between two strings: either one is acceptable
- `[]` around a string: shorthand for a “|” between each character in the string.
- `ε`: a string of zero length.
- `?` after a string: 0 or 1 occurrences
- `+` after a string: string may be repeated one or more times
- `*` after a string: string may be repeated zero or more times
- `\` in front of a letter: the terminal character itself, not the meta

**Other Conventions**

- **Keywords** in language shown in bold
- Nonterminal names shown in italic
- Nonterminals ending in “_list” means implied rule of form:
  - `nonterm_list -> nonterm(, nonterm)*`
Sample Arithmetic Expression

- $digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
- $pos\_number \rightarrow digit+$
- $number \rightarrow \{\+ \mid \-\}? pos\_number$
- $factor \rightarrow number$
- $factor \rightarrow (expression)$
- $term \rightarrow factor$
- $term \rightarrow term \times factor$
- $expression \rightarrow term$
- $expression \rightarrow term \{\+ \mid \-\} expression$

Your Presentation

- Background:
  - where does it come from and who uses it
  - what in general is its objective
  - where can you get code
- How are graphs expressed
- What is (simplified) syntax of statements
- What graph primitives are supported
- What is execution model
  - Especially options for parallelism
- Simple examples