Ferramenta educacional para auxiliar no ensino de Fork/Join

Lucas da Silva Marcos[1], **Rodolfo Adamshuk Silva**[1]

[1]Coordenação do Curso de Engenharia de Software
Universidade Tecnológica Federal do Paraná (UTFPR) – Dois Vizinhos, PR – Brazil

lucasmarcos@alunos.utfpr.edu.br, rodolfoa@utfpr.edu.br

Abstract.

Resumo.

1. Introdução

1.1. Problema

Programação concorrente ou programação simultânea é um paradigma de programação para a construção de programas de computador que fazem uso da execução simultânea de várias tarefas computacionais interativas, que podem ser implementadas como programas separados ou como um conjunto de threads criadas por um único programa. Essas tarefas podem ser executadas por um único processador, vários processadores em um único equipamento ou processadores distribuídos por uma rede. Programação concorrente é relacionada com programação paralela, mas foca mais na interação entre as tarefas. A interação e a comunicação correta entre as diferentes tarefas, além da coordenação do acesso simultâneo aos recursos computacionais são as principais questões discutidas durante o desenvolvimento de sistemas simultâneos. Pioneiros na área de pesquisa incluem Edsger Dijkstra, Per Brinch Hansen, e C.A.R. Hoare.

Vantagens do paradigma incluem o aumento de desempenho, pois aumenta-se a quantidade de tarefas sendo executadas em determinado período de tempo, e a possibilidade de uma melhor modelagem de programas, pois determinados problemas computacionais são simultâneos por natureza.

Ensino de programação concorrente.

Ensino de programação no ensino superior. Unidade curricular de Programação Concorrente e Distribuída.

Na computação paralela, o modelo fork-join (bifurcar-juntar) é uma forma de configurar e executar programas em paralelo de tal forma que a execução das tarefas concorrentes "juntam-se"(join) em um ponto subsequente, onde a execução passa a ser sequencial. As seções paralelas podem se bifurcar recursivamente até que uma determinada granularidade da tarefa seja atingida. Fork-join pode ser considerado como um padrão de projeto paralelo.[1] Foi formulado já em 1963.[2][3]

Anos 60. Algumas implementações.

Decomposição. Grafo de tarefas. Métricas.

Linguagem de alto nível.

Um compilador é um programa de computador (ou um grupo de programas) que, a partir de um código fonte escrito em uma linguagem compilada, cria um programa semanticamente equivalente, porém escrito em outra linguagem, código objeto.[1] Classicamente, um compilador traduz um programa de uma linguagem textual facilmente entendida por um ser humano para uma linguagem de máquina , específica para um processador e sistema operacional. Atualmente, porém, são comuns compiladores que geram código para uma máquina virtual que é, depois, interpretada por um interpretador. Ele é chamado compilador por razões históricas; nos primeiros anos da programação automática, existiam programas que percorriam bibliotecas de sub-rotinas e as reunia, ou compilava,[Nota 1] as subrotinas necessárias para executar uma determinada tarefa.[2][3]

O nome "compilador"é usado principalmente para os programas que traduzem o código fonte de uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível (por exemplo, Assembly ou código de máquina). Contudo alguns autores citam exemplos de compiladores que traduzem para linguagens de alto nível como C.[4] Para alguns autores um programa que faz uma tradução entre linguagens de alto nível é normalmente chamado um tradutor, filtro[5] ou conversor de linguagem. Um programa que traduz uma linguagem de programação de baixo nível para uma linguagem de programação de alto nível é um descompilador.[6] Um programa que faz uma tradução entre uma linguagem de montagem e o código de máquina é denominado montador (assembler).[5] Um programa que faz uma tradução entre o código de máquina e uma linguagem de montagem é denominado desmontador (disassembler).[6] Se o programa compilado pode ser executado em um computador cuja CPU ou sistema operacional é diferente daquele em que o compilador é executado, o compilador é conhecido como um compilador cruzado.[7]

Na ciência da computação, análise léxica, lexing ou tokenização é o processo de converter uma sequência de caracteres (como em um programa de computador ou página da web) em uma sequência de tokens (strings com um significado atribuído e, portanto, identificado). Um programa que realiza análise lexical pode ser denominado lexer, tokenizer,[1] ou scanner, embora scanner também seja um termo para o primeiro estágio de um lexer. Um lexer geralmente é combinado com um analisador, que juntos analisam a sintaxe de linguagens de programação, páginas da Web e assim por diante.

Em ciência da computação e linguística, a análise sintática (do inglês: parsing) é um processo de um compilador (de uma linguagem de programação), é a segunda fase da compilação onde se analisa uma sequência que foi dada entrada (via um arquivo de computador ou via teclado, por exemplo) para verificar sua estrutura gramatical segundo uma determinada gramática formal. Este processo trabalha em conjunto com a análise lexical (primeira etapa, onde se verifica de acordo com determinado alfabeto) e análise semântica (terceira etapa, onde verificam-se os erros semânticos).[1]

A análise sintática transforma um texto na entrada em uma estrutura de

dados, em geral uma árvore, o que é conveniente para processamento posterior e captura a hierarquia implícita desta entrada. Através da análise lexical é obtido um grupo de tokens, para que o analisador sintático use um conjunto de regras para construir uma árvore sintática da estrutura.

Em termos práticos, por exemplo, pode também ser usada para decompor um texto em unidades estruturais para serem organizadas dentro de um bloco.

A vasta maioria dos analisadores sintáticos implementados em compiladores aceitam alguma linguagem livre de contexto para fazer a análise. Estes analisadores podem ser de vários tipos, como o LL, LR e SLR.

1.2. Motivação

Conceitos complexos de programação concorrente.

Escassez de material e ferramentas de apoio e suporte a educação.

Dificuldades de aprendizado.

1.3. Objetivo

Construção de uma ferramenta de apoio ao ensino por meio de um compilador para uma linguagem de alto nível que represente o modelo fork-join.

Desenvolver uma ferramenta educacional para auxiliar no ensino do modelo Fork/Join de concorrência no contexto de uma unidade curricular para cursos de Ensino Superior em Engenharia de Software com foco nas atividades desenvolvidas na disciplina de Programação Concorrente Distribuída.

2. Trabalhos relacionados

Fork/Join [Conway 1963] e [Dennis and Van Horn 1966].

Sistemas Distribuídos [Coulouris et al. 2001] e [Van Steen and Tanenbaum 2017].

Compiladores [Louden 2004].

Ferramentas educacionais.

Materia de apoio para aprendizagem de Programação Concorrente.

Compiladores em sala de aula.

Compiladores de Pascal usados para o ensino de matemática.

3. Proposta

O desenvolvimento de um compilador e plataforma com o foco no uso em sala de aula.

Será especificado uma gramática formal da linguagem de alto nível utilizada. A linguagem possuirá os primitivos Fork e Join como modelo de concorrência.

Será construído um analisador léxico, reportando aos alunos erros nos programas.

Será construído um analisador sintático, que modelara a estrutura do programa e reportará erros de construção.

Como saída esperada teremos, além das mensagens de erro ou de sucesso, um grafo de tarefas e dependências, além de métricas do programa especificado, como o caminho crítico e o grau máximo de concorrência.

Também será construído uma plataforma web, como um editor de textos integrado e renderização do grafo e métricas. Exemplo Compiler Explorer https://godbolt.org.

4. Metodologia

Pesquisa aplicada e exploratória.

Aplicaremos o protótipo em sala de aula, coletaremos dados e feedback dos alunos. Feedback pesquisa qualitativa.

Cronograma.

5. Resultados esperados

Espera-se que a ferramenta proposta contribua significativamente para o aprimoramento do ensino de programação concorrente, fornecendo um recurso para o aprendizado dos alunos e auxiliando os professores no processo de ensino.

Referências

Conway, M. E. (1963). A multiprocessor system design. In *Proceedings of the November 12-14, 1963, fall joint computer conference*, pages 139–146.

Coulouris, G., Dollimore, J., and Kindberg, T. (2001). *Sistemas distribuidos*, volume 6. Addison Wesley Madrid.

Dennis, J. B. and Van Horn, E. C. (1966). Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155.

Louden, K. (2004). *Compiladores - Princípios e Práticas*. Pioneira Thomson Learning.

Van Steen, M. and Tanenbaum, A. S. (2017). *Distributed systems*. Maarten van Steen Leiden, The Netherlands.