AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wybrane metody automatycznej generacji kodu na procesory wielordzeniowe

Student realizujący: Piotr Listkiewicz
Opiekun pracy: dr inż. Marcin Pietroń

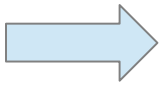
Cel pracy

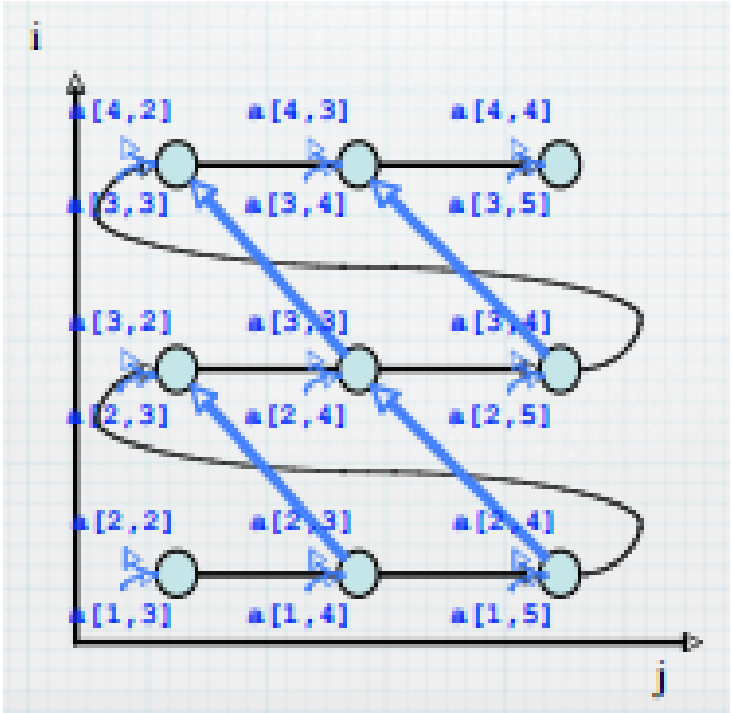
- **Analiza metod automatycznej detekcji kodu który można zrównoleglić**
- **Wytworzenie narzędzia umożliwiającego automatyczną generację kodu w wersji równoległej**
- **Obecnym celem jest implementacja narzędzia dla języka C , które przekształca go do postaci OpenMP**
- **Sprawdzenie działania narzędzia na algorytmach Machine Learningu**

Testowanie zależności

- **Wykrywanie zależności występujących w pętli pomiędzy iteracjami i instrukcjami**
- **Jest kilka możliwych rodzajów zależności pomiędzy instrukcjami**
- **Na ich podstawie buduje się graf zależności**

```
  for i = 2 to 4
    for j = 2 to 4
1:    a[i,j] = a[i-1,j+1];
```

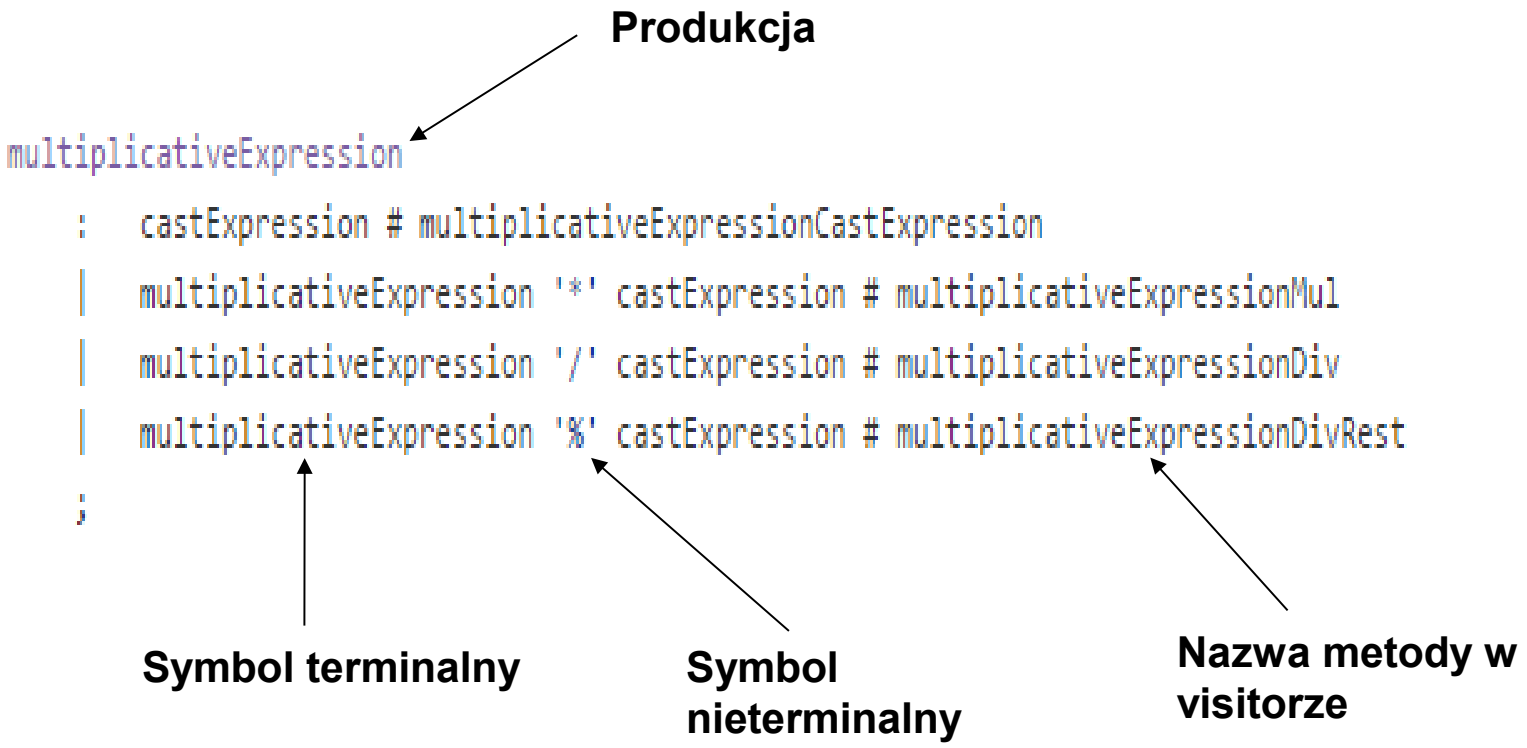




Przekształcenie drzewa rozbioru na AST

- Drzewo rozbioru oparte o gramatykę, AST(Abstract Syntax Tree) własne drzewo udostępniające odpowiednie informację
- Wykorzystuje wzorzec visitor z antlr4, polega na nadpisaniu odpowiednich funkcji z utworzonego przez antlr4 domyślnego visitora
- Antlr4 jest zintegrowany z mavenem – przy każdej budowie projektu na podstawie gramatyki są tworzone odpowiednie parsery,visitory itp.

Gramatyka języka C – przykład produkcji



Visitor – przykład metody

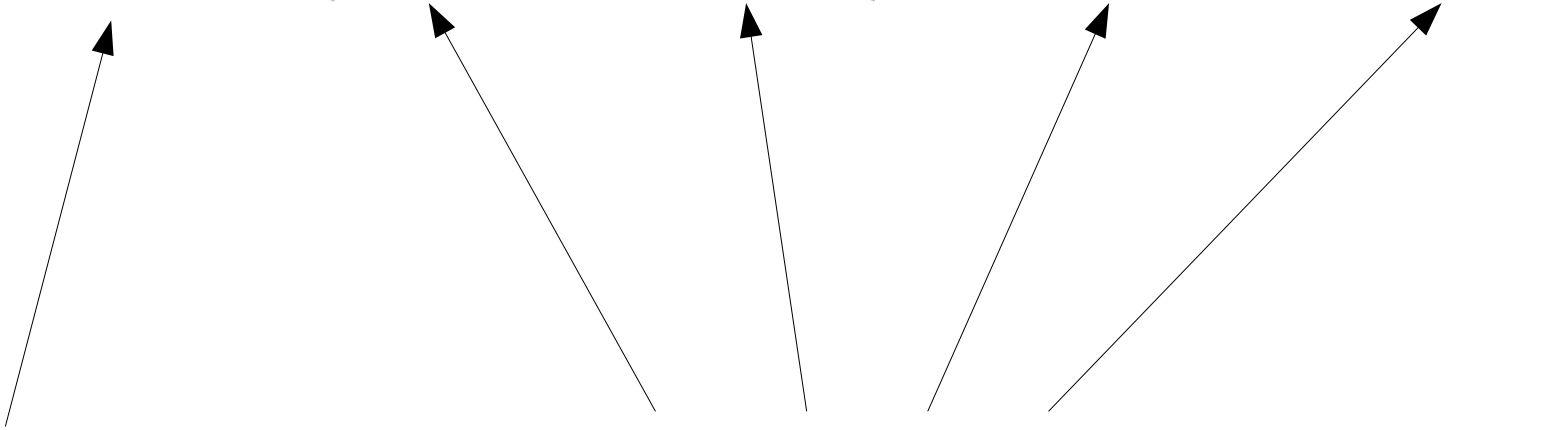
```
@Override
public DetailAst visitMultiplicativeExpressionDivRest(C11Parser.MultiplicativeExpressionDivRestContext ctx) {
    List<DetailAst> childrens = new LinkedList<DetailAst>();
    childrens.add(visit(ctx.multiplicativeExpression()));
    childrens.add(visit(ctx.castExpression()));
    return new DivRestExpressionsAst(getText(ctx), childrens);
}
```

Testowanie

- Wymagana duża poprawność rozwiązania – odczytanie wywołania funkcji jako deklaracji zmiennej może totalnie wypaczyć działanie algorytmów testowania zależności
- Testy jednostkowe – w tym momencie około 200 testów
- Całość wytwarzana przy pomocy TDD
- Testy integracyjne – bierzemy kody w C , parsujemy do AST, potem z AST przekształcamy do postaci tekstu i porównujemy z plikiem wejściowym
- W tym momencie znajduje się 10 plików, ponad 7000 linii kodu w C

Zależności instrukcji

```
len = (num > MAXLINE) ? MAXLINE : num;
```



Zależność wyjściowa

Zależności wejściowe

Obliczenie Data Flow Graph

- **Węzły reprezentują instrukcję**
- **Krawędź reprezentuje zależność pomiędzy instrukcjami: kierunek krawędzi reprezentuje która instrukcja następuje po której, typ krawędzi reprezentuje jakiego rodzaju jest zależność**
- **Rodzaje zależności : Read-After-Read, Read-After-Write, Write-After-Read, Write-After-Write**
- **Krawędź występuje tylko do najbliższej instrukcji ze względu na daną zmienną, w stosunku do instrukcji późniejszych można odczytać zależność „przechodnio” np.:**
  - **1 -RAW-> 2 -RAR-> 3 => 1 -RAW-> 3**

Obliczenie Data Flow Graph - cd

- Zaimplementowany algorytm generujący
- Generacja DFG jest w testowana w szczególności przypadki takie jak pętle, wyrażenia tablicowe
- Testowanie przy pomocy unit testów, jest też generacja plików dot odczytywanych przez graphviz
- Było sprawdzanych kilka możliwości generacji grafów, ale narzędzia takie jak jung dawały nieczytelne ułożenie wierzchołków

Następne kroki

- **Posiadając DFG będzie można podpiąć istniejące algorytmy testowania zależności po dostosowaniu ich do API**
- **Z wyników algorytmów testowania zależności i DFG będzie następowała generacja wstawek w OpenMP**
- **Testowanie rozwiązania na algorytmach machine learningu na maszynach wieloprocesorowych**

Pytania?