FERRAMENTA PARA EXECUÇÃO DE SCRIPTS OPL PARA OTIMIZAÇÃO COMBINATÓRIA EM GPU

Roger William Weber
Orientadora: Joyce Martins, Mestre
FURB - Universidade Regional de Blumenau

Roteiro

- Introdução
- Objetivos do trabalho
- Fundamentação teórica
- Desenvolvimento
- Conclusão
- Demonstração

Introdução

Otimização combinatória é um ramo da ciência da computação e da matemática aplicada que estuda problemas de otimização em conjuntos finitos.

Exemplo de plantação

Maximizar o lucro S1 * X1 + S2 * X2
(valor de venda produto A + valor de venda do produto B)
Sujeito a:
x1 + x2 <= Área total
F * x1 + F * x2 <= Quantidade de fertilizante
P * x1 + P * x2 <= Quantidade de pesticida
x1 e x2 >= 0

Simplex

- George Dantzig, 1947
- Consiste em encontrar uma solução básica inicial e ir sucessivamente buscando outras soluções básicas viáveis, acarretando melhores valores para a função objetivo até ser obtida a solução ótima.

IBM ILOG CPLEX Studio

- Implementação de algoritmos
- Linguagem OPL
- IDE

GPU

- Criada em 2006, NVIDIA
- *Compute Unified Device Architecture* (CUDA)
- Utilização de placas de vídeo para computação de propósito geral.
- CUDA *Toolkit*
- CUDA C

Objetivos do trabalho

- Desenvolver uma ferramenta para interpretar *scripts* OPL, gerando código paralelo para máquinas com placa de vídeo NVidia
- Implementar o algoritmo Simplex na versão GPU

Fundamentação teórica

- Tradutores de linguagem
- OPL
- Algoritmos de otimização combinatória
- Tecnologia GPU
- Trabalhos Correlatos

Tradutores de linguagem

- Programa fonte X programa objeto
- Fases: Léxica, sintática e semântica

OPL

- OPL é uma linguagem de modelagem que acelera o desenvolvimento e a implementação de métodos de otimização combinatória usando programação linear baseada em restrições.

```
{string} Products = ...;
{string} Components = ...;

float Demand[Products][Components] = ...;
float Profit[Products] = ...;
float Stock[Components] = ...;
dvar float+ Production[Products];

maximize
  sum( p in Products )
    Profit[p] * Production[p];

subject to {
  forall( c in Components )
      sum( p in Products )
        Demand[p][c] * Production[p] <= Stock[c];
}
```

```
Products = { "gas" "chloride" };
Components = { "nitrogen" "hydrogen" "chlorine" };

Demand = [ [4 3 2] [3 6 3] ];
Profit = [30 40];
Stock = [30 50 40];
```

Algoritmos de otimização combinatória

- Soluções exatas:
  - Branch e bound
  - Simplex
- Soluções aproximadas
  - GRASP
  - Tabu Search

Tecnologia GPU

- CUDA Toolkit
- CUDA Compiler
- Nsight Eclipse
- Bibliotecas

Exemplo código

```
//kernel definition
__global__ void VecAdd(float* A, float* B, float* C) {
int i = threadIdx.x;
C[i] = A[i] + B[i];
}
int main() {
// invocação de função Kernel com N threads
VecAdd<<<1, N>>>(A, B, C);
}
```

Trabalhos correlatos

- Compilador para código Java 5.0 que gera código C++ para plataforma Palm OS, por Gesser (2007).
- Simplex usando a tecnologia CUDA, por Lalami, Boyer e El-Baz (2011)
- Algoritmo Tabu *search* usando CUDA, por Moreira e Meneses (2012)

Desenvolvimento

- Requisitos
- Especificação
- Implementação
- Operacionalidade
- Resultados e discussão

Requisitos principais

- Analisar scripts OPL, baseados na ferramenta IBM ILOG CPLEX
- Implementar uma versão paralela em C++ para GPU do algoritmo Simplex
- Gerar código C++

Especificação

- Linguagem OPL script e dados

Tokens

token	descrição
Símbolos especiais	{ } = ... ; [ ] ( ) : + * <=
maximize	identifica que o problema é de maximização
sum	identifica o somatório
in	identifica operação em uma lista
subject	identifica restrições
forall	identifica uma iteração em uma lista
dvar	identifica uma variável de decisão
float	usada na declaração de ponto flutuante

Construções sintáticas

```
...
Objective: ... | "maximize" Expression

Expression: ... | PathExpression | BinaryExpression | AggregateExpression | ...

PathExpression: LocationExpression | ArraySlotExpression | ...
LocationExpression: Id | ...
ArraySlotExpression: PathExpression '[' Expression ']' | ...

BinaryExpression: ... | Expression "*" Expression

AggregateExpression: "sum" '(' Qualifiers ')' Expression | ...
Qualifiers: Qualifier | Qualifiers ',' Qualifier
Qualifier: SimpleQualifier | ...
SimpleQualifier: Pattern "in" Expression Filter_opt
Pattern: Expression
Filter_opt: /* empty */ | ...
...
```

construção OPL	construção C++
`{string} Products = ...;`	`extern vector<string> Products;`
`float Demand[Products][Components] = ...`	`double Demand DemandVal;`
`float Profit[Products] = ...`	`double Profit ProfitVal;`

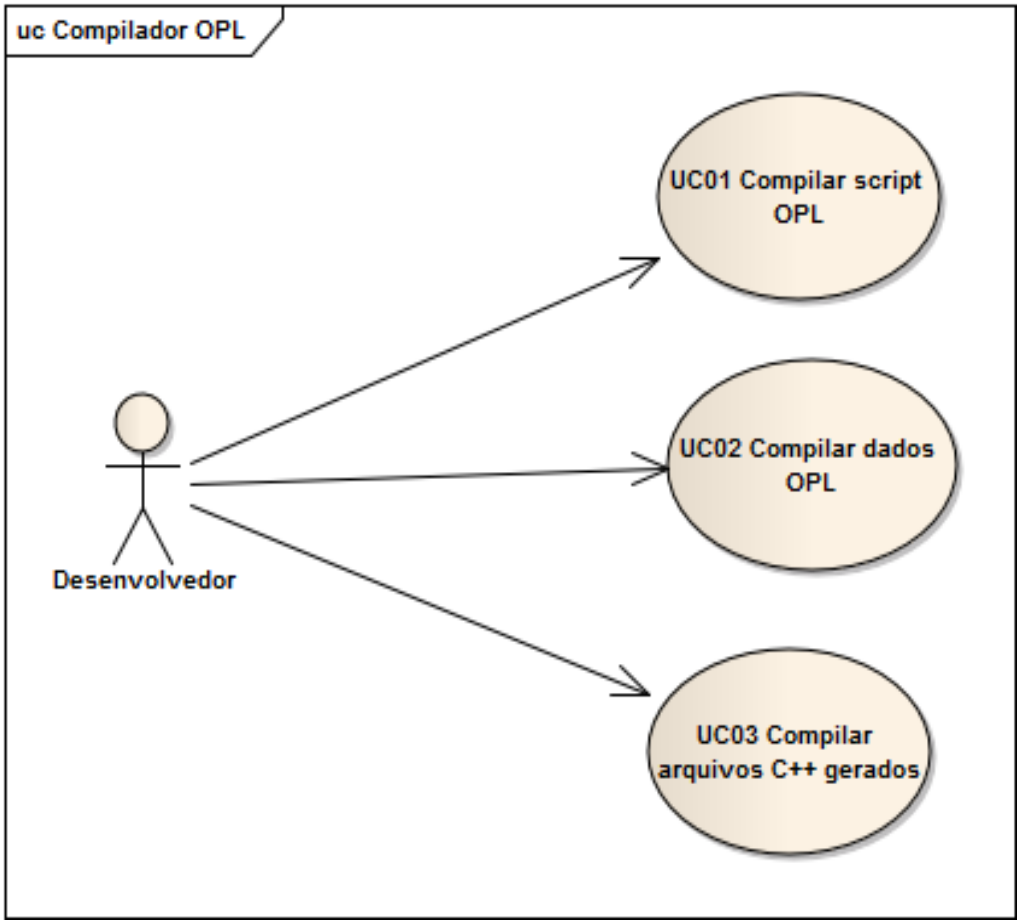
construção OPL	construção C++

`maximize`
`  sum( p in Products )`
`  Profit[p] * Production[p];`
	`sizeArray =`
	`sizeof(Profit)/sizeof(Profit[0]);`
	`for(int i = 0; i < sizeArray; i++){`
	`  obj.valores.push_back(Profit[i]);`
	`}`

construção OPL	construção C++

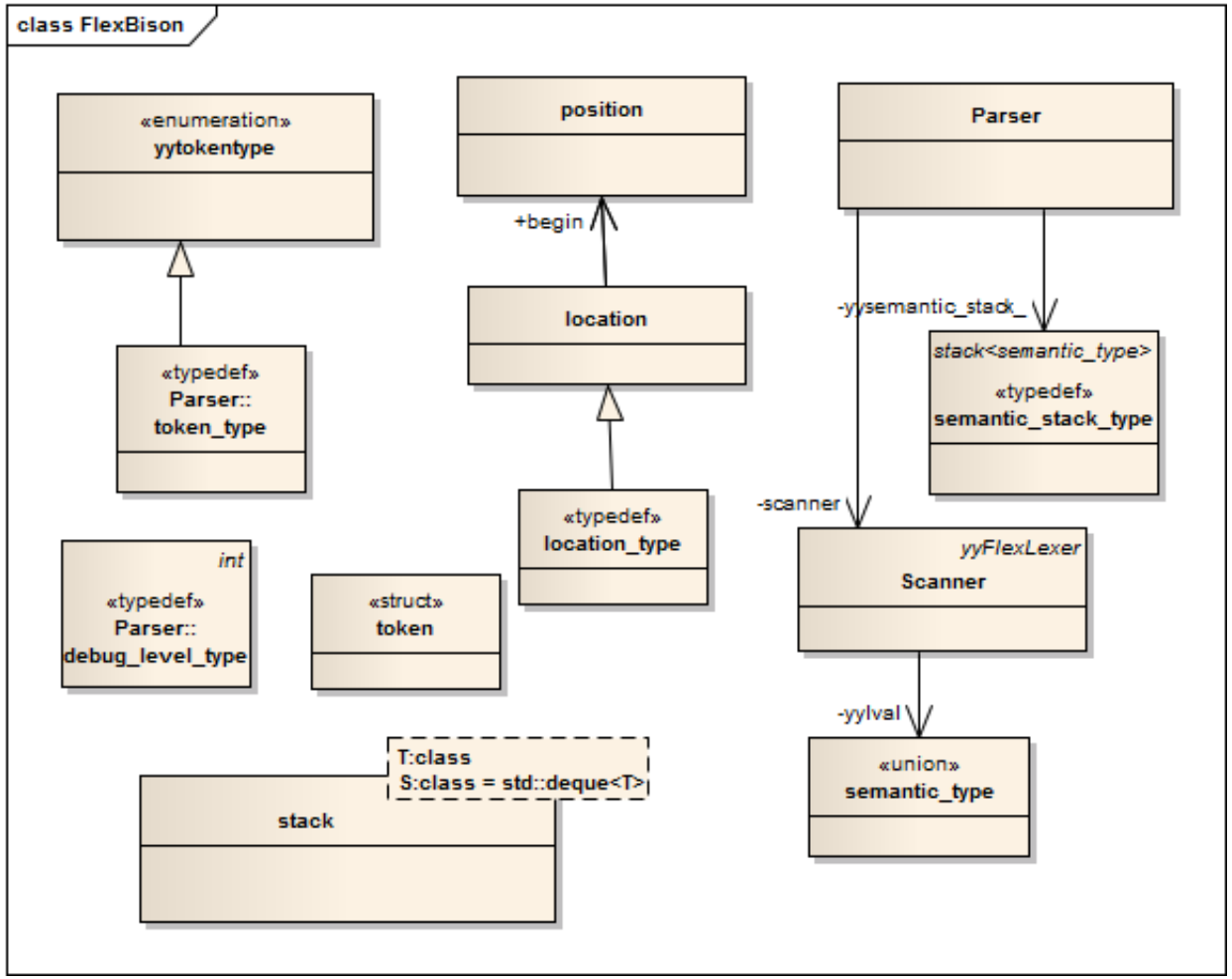
`subject to {`
` forall( c in Components )`
`  ct:`
`  sum( p in Products )`
`  Demand[p][c] * Production[p]`
`  <= Stock[c];`
`}`
	`for(int c=0; c < Components.size(); c++) {`
	`  Restricao _restricao;`
	`  _restricao.id = "ct";`
	`  for(int p = 0; p < Products.size(); p++){`
	`  _restricao.pesos.push_back(Demand[p][c]);`
	`  }`
	`  _restricao.limite = Stock[p];`
	`  restricoes.push_back(_restricao);`
	`}`

Casos de uso

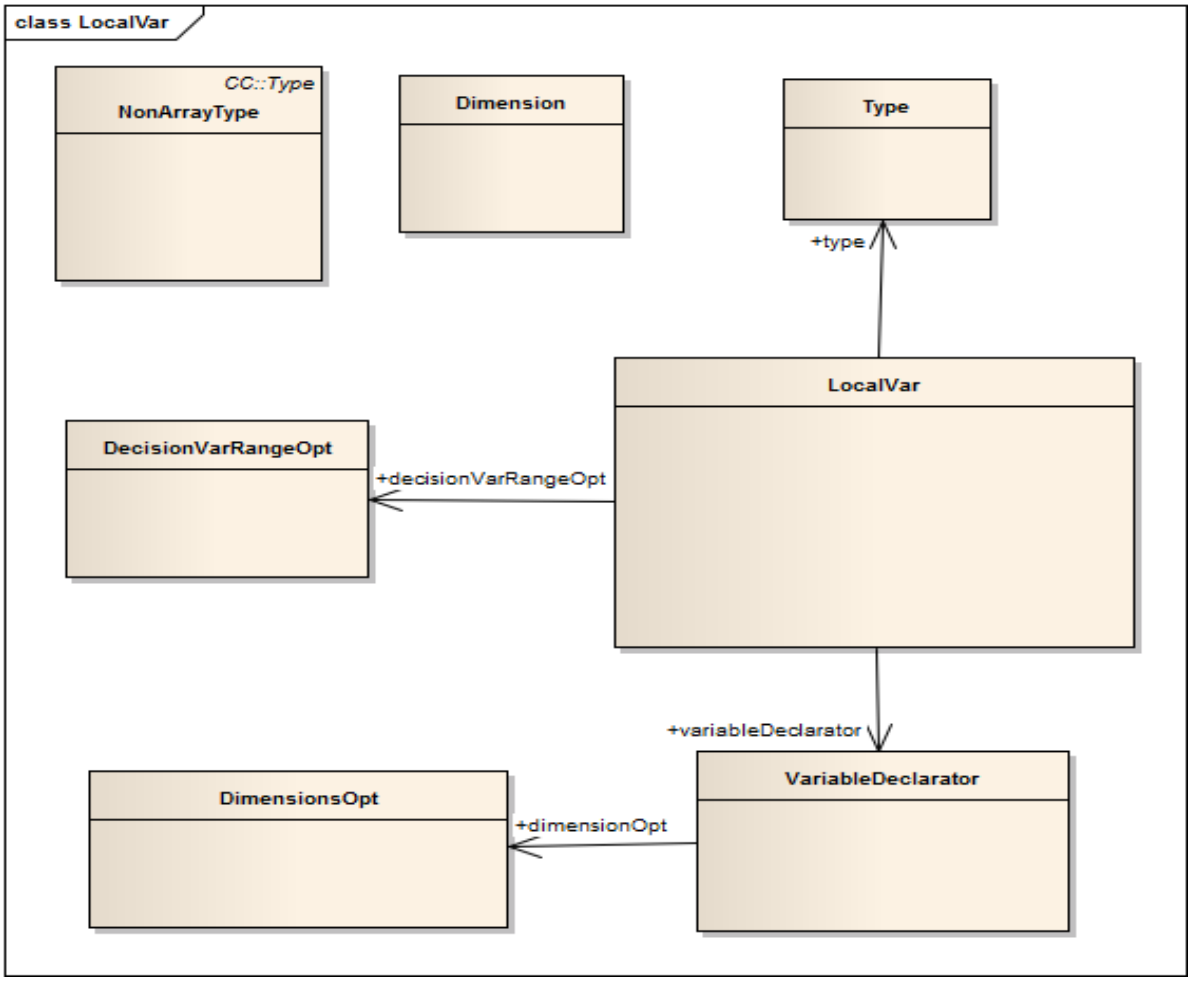

uc Compilador OPL
UC01 Compilar script OPL
UC02 Compilar dados OPL
UC03 Compilar arquivos C++ gerados
Desenvolvedor

Diagramas de classes

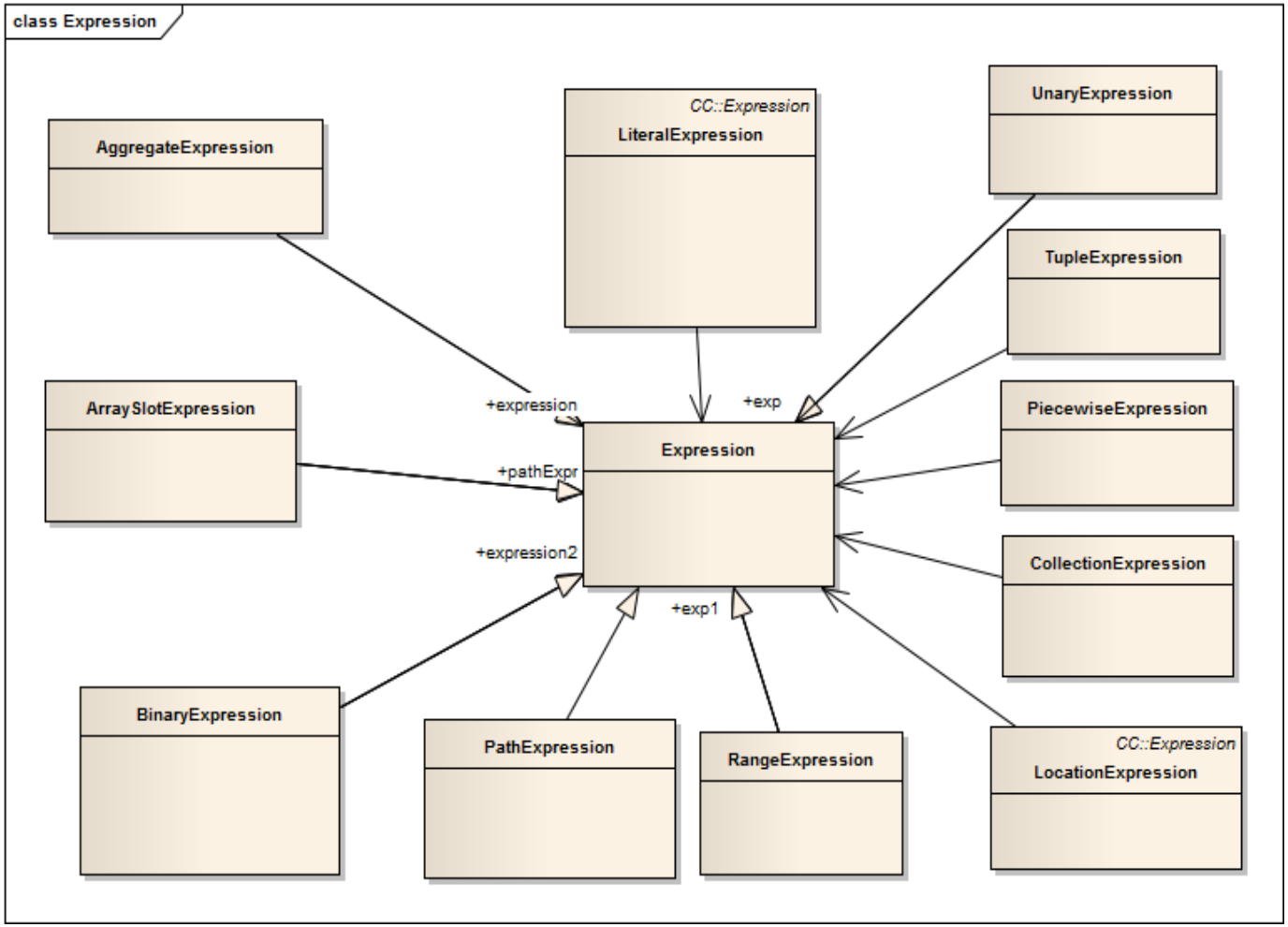
Léxico e sintático



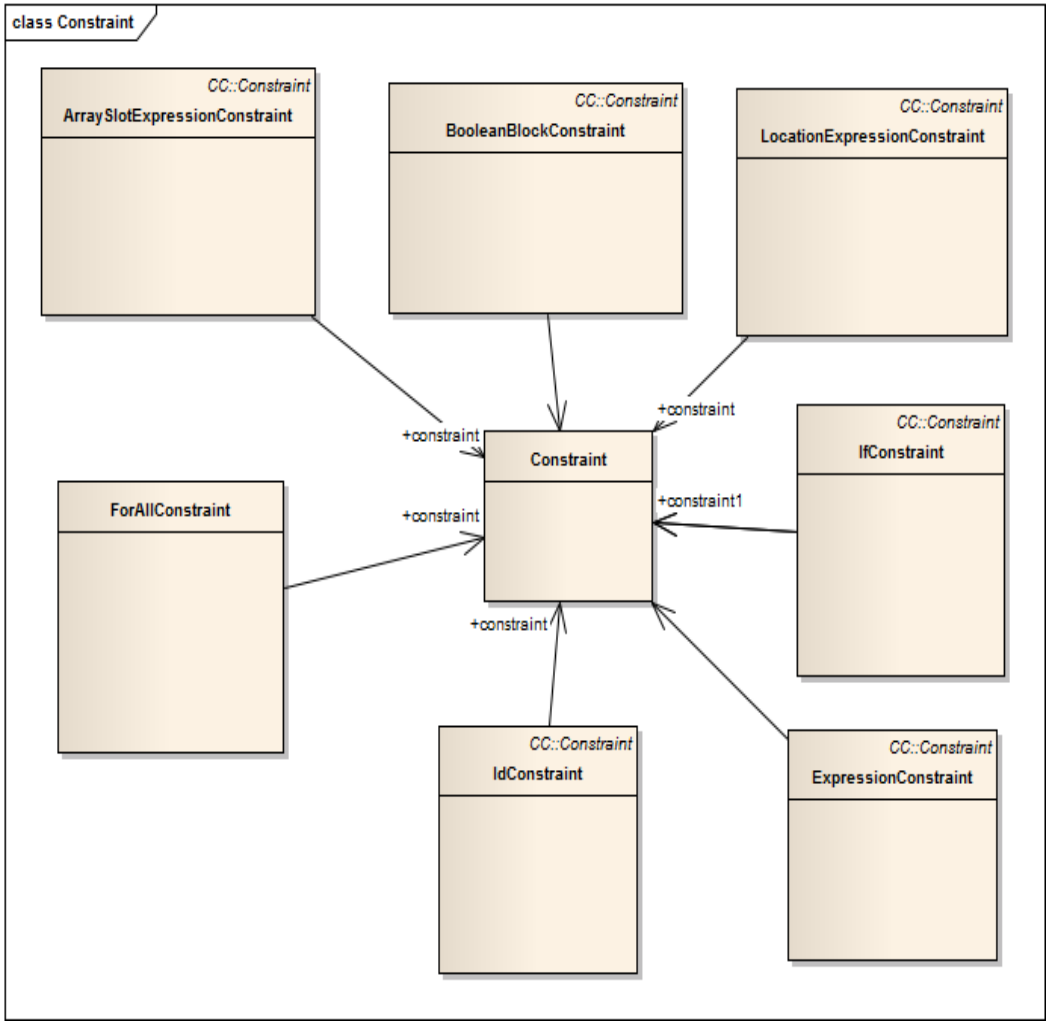
Declaração de variáveis



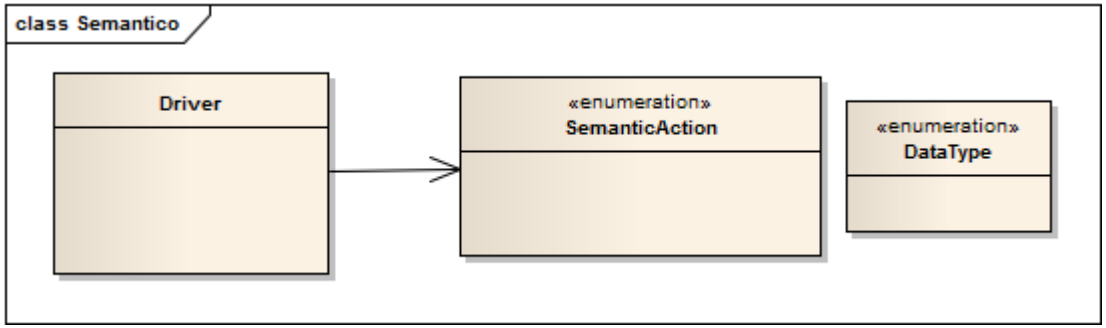
Declaração de expressões



Restrições


class Constraint
CC::Constraint
ArraySlotExpressionConstraint
CC::Constraint
BooleanBlockConstraint
CC::Constraint
LocationExpressionConstraint
+constraint
+constraint
Constraint
+constraint
+constraint1
CC::Constraint
IfConstraint
ForAllConstraint
+constraint
CC::Constraint
IdConstraint
CC::Constraint
ExpressionConstraint

Semântica


class Semantico
Driver
«enumeration»
SemanticAction
«enumeration»
DataType

Simplex

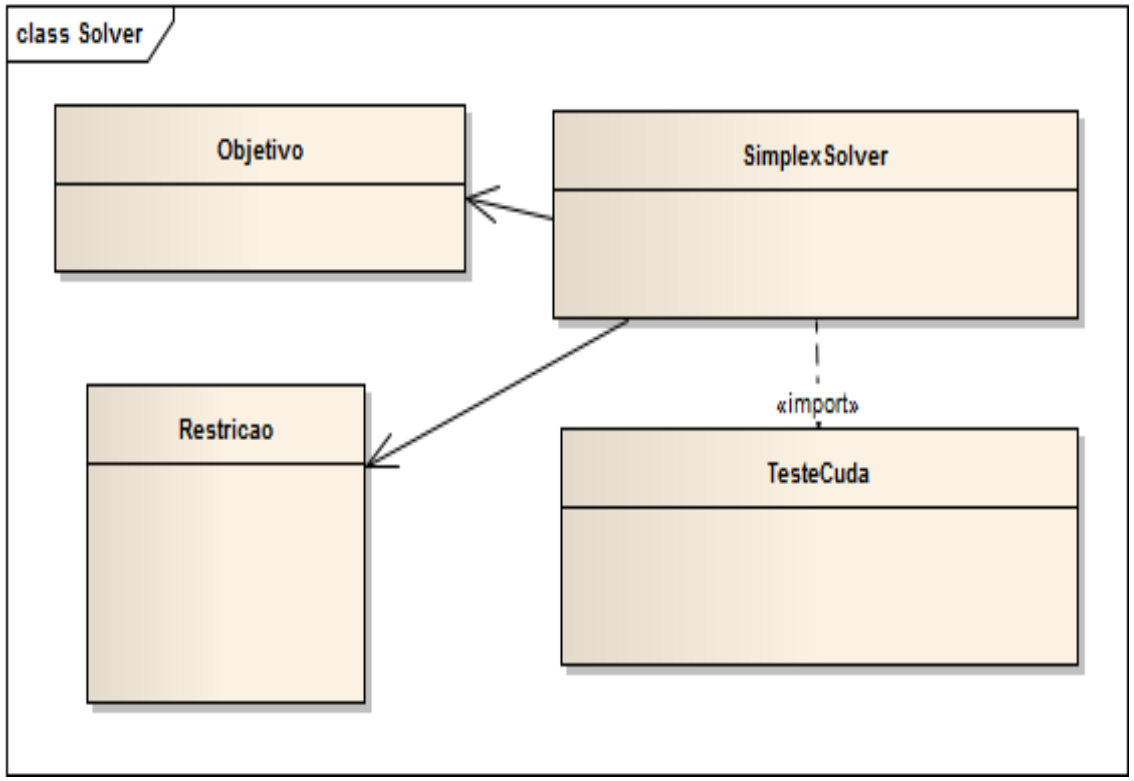
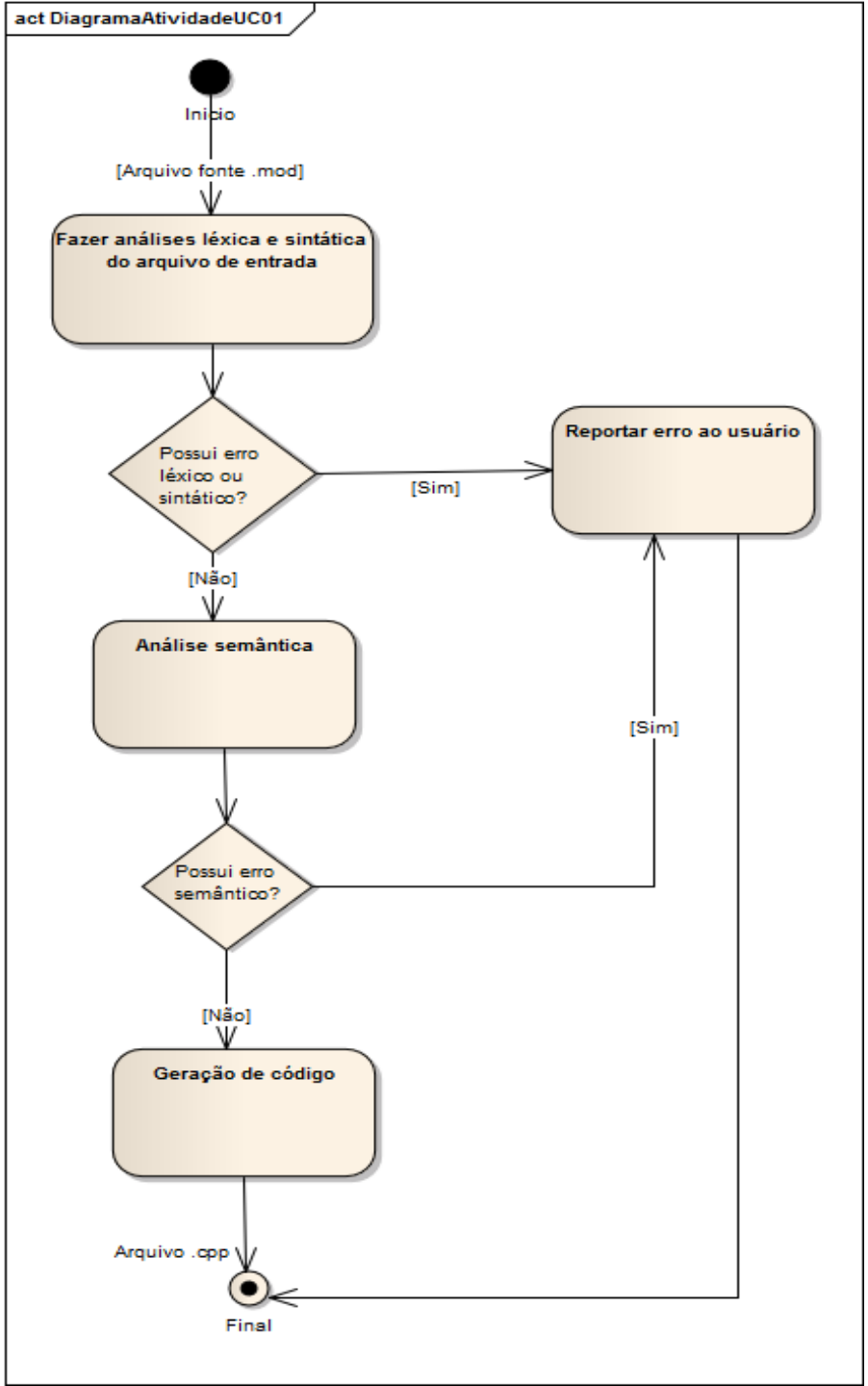

class Solver
Objetivo
SimplexSolver
Restricao
«import»
TesteCuda

Diagrama de atividades


act DiagramaAtividadeUC01
Início
[Arquivo fonte .mod]
Fazer análises léxica e sintática do arquivo de entrada
Possui erro léxico ou sintático?
[Sim]
Reportar erro ao usuário
[Não]
Análise semântica
[Sim]
Possui erro semântico?
[Não]
Geração de código
Arquivo .cpp
Final

Implementação

- Técnicas e ferramentas
- Compiladores para scripts e dados OPL
- Projeto base para testes
- Algoritmo Simplex

Técnicas e ferramentas

- Flex
- Bison
- CUDA Toolkit

Compiladores para script e dados

- Análise léxica feita com Flex, usando a especificação pública da linguagem
- Análise sintática feita com Bison

Geração de código

Declaração externa de variável

```
case THREEPOINTS: {
  LocalVar localVar;
  localVar.variableDeclarator = variableDeclarator;
  localVar.type = types.top();
  types.pop();
  string codigo;
  if(dynamic_cast<NonArrayType*>(localVar.type)) {
    NonArrayType *n = dynamic_cast<NonArrayType*>(localVar.type);
    if(n->dataType == STRING)
      codigo = "extern vector<string> " + localVar.variableDeclarator->id + ";\n";
    else {
      codigo = getTipoDado(n->dataType) + " " + localVar.variableDeclarator->id +
               " " + localVar.variableDeclarator->id + "Val;\n";
    }
    gerar(codigo);
  }
  else
    cout << "Falha" << endl;
  break;
}
```

Função objetivo

```
string AggregateExpression::Generate() {
switch (tipoAggregateExpression) {
  case 0: //sum
  {
  if(expression == NULL || qualifiers == NULL){
    return "Erro ao gerar AggregateExpression";
  }
  ret = "//Objetivo\n";
  BinaryExpression *bExp = dynamic_cast<BinaryExpression*>(expression);
  if(bExp != NULL) {
    vector<Expression*> vectorBinaryExp = bExp->Parser();
    for(int i = 0; i < vectorBinaryExp.size(); i++) {
      bExp = dynamic_cast<BinaryExpression*>(vectorBinaryExp.at(i));
      ArraySlotExpression *aExp =
dynamic_cast<ArraySlotExpression*>(bExp->expression1);
      LocationExpression *lExp =dynamic_cast<LocationExpression*>(aExp->pathExpr);
      ret += "sizeArray = sizeof("+lExp->id+") / sizeof(" + lExp->id + "[0]);\n";
      ret += "for(int i = 0; i < sizeArray; i++)\n{\n";
      ret += "   obj.valores.push_back("+lExp->id + "[i]);\n";
      ret +=  "}\n";
    }
  }
  break;
  }
  default:
  break;
}
return ret;
}
```

Declaração de restrição

```
string ForAllConstraint::Generate() {
  string codigo = "//Restricoes\n";
  BinaryExpression *bExp =
    dynamic_cast<BinaryExpression*>(qualifiers->stackQualifier.top()->expression);
  if(bExp != NULL) {
    if(bExp->op == 14)//IN
    {
      string exp1 = bExp->expression1->Generate();
      codigo += "for(int "+ exp1 + "= 0;" + exp1 + " < " +
                bExp->expression2->Generate() + ".size(); " + exp1 + "++)\n{\n";
      codigo += "Restricao _restricao;\n";
      codigo += constraint->Generate();
      codigo += "}\n";
    }
    else
      return NULL;
  }
  return codigo;
}

string ExpressionConstraint::Generate() {
  string codigo = "";
  AggregateExpression *agrE = dynamic_cast<AggregateExpression*>(exp);
  if (agrE != NULL) {
    if (agrE->tipoAggregateExpression == 0) //sum
    {
      BinaryExpression *bExp = dynamic_cast<BinaryExpression*>
              (agrE->qualifiers->stackQualifier.top()->expression);
      if (bExp != NULL) {
        codigo += "for(int " + bExp->expression1->Generate() + " = 0; " +
                  bExp->expression1->Generate() + " < " +
                  bExp->expression2->Generate() + ".size(); " +
                  bExp->expression1->Generate() + "++)\n{\n";
        codigo += agrE->expression->Generate();
        return codigo;
      }
    }
  }
  else
    return "Falha";
}
```

Projeto teste

- Projeto TesteCuda
- Arquivos padrão: Objetivo.h, Restricao.h
- SimplexSolver.cpp
- TesteCuda.cu

SimplexSolver.cpp

```
#include "SimplexSolver.h"
extern "C" int resolve(double *tableu, int linhas, int colunas);

namespace CC {
void SimplexSolver::MontarTableu(vector<Restricao> restricoes, Objetivo objetivo){
  cout << "restricoes =" << restricoes.size();
  cout << " objetivos =" << objetivo.valores.size() << endl;
  int linhas = restricoes.size() +1;
  int colunas = objetivo.valores.size() + linhas;
  tableu = new double[linhas * colunas];

  linhasCount = linhas;
  colunasCount = colunas;

  //preenche objetivo
  for(int i = 0; i < objetivo.valores.size(); i++) {
    tableu[i+1] = -objetivo.valores[i];
  }

  //preenche restricoes
  for(unsigned int i = 0; i < restricoes.size(); i++) {
    Restricao r = restricoes[i];
    //ignora primeira linha do tableu
    tableu[(i+1) * colunas] = r.limite;
    for(int j = 0; j < r.pesos.size(); j++) {
                tableu[(i+1) * colunas + 1 + j] = r.pesos[j];
    }
  }

  //preenche identidade
  for(unsigned int i = 1; i < linhas; i++) {
    int indice = i*colunas + linhas +(i+1);
    tableu[indice] = 1;
  }
}

int SimplexSolver::Solve(void) {
  return resolve(tableu, linhasCount, colunasCount);
}
} /* namespace CC */
```

Algoritmo Simplex

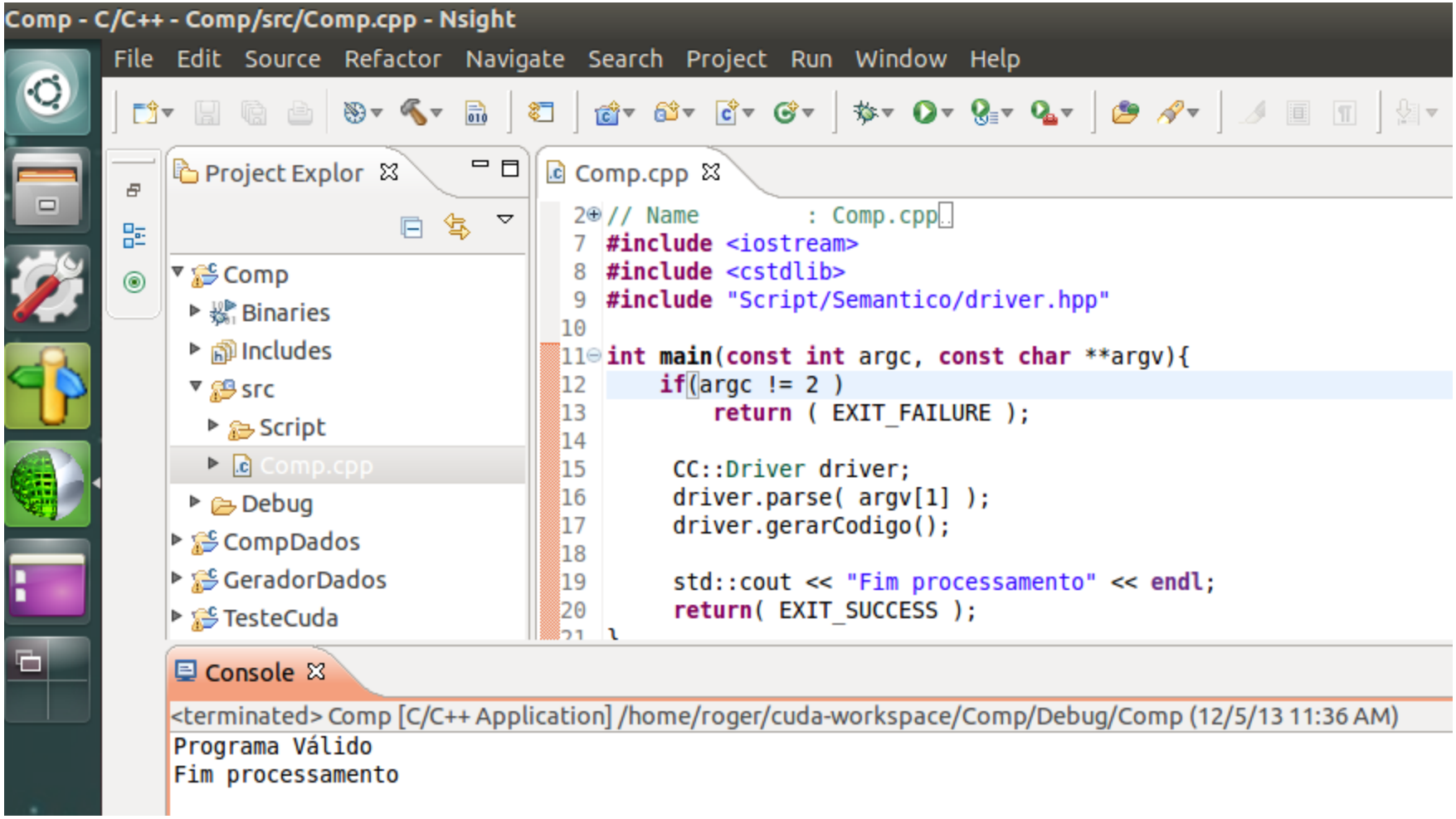
- Implementação Simplex Padrão, restrições do tipo menor igual.
- Tableu N x M

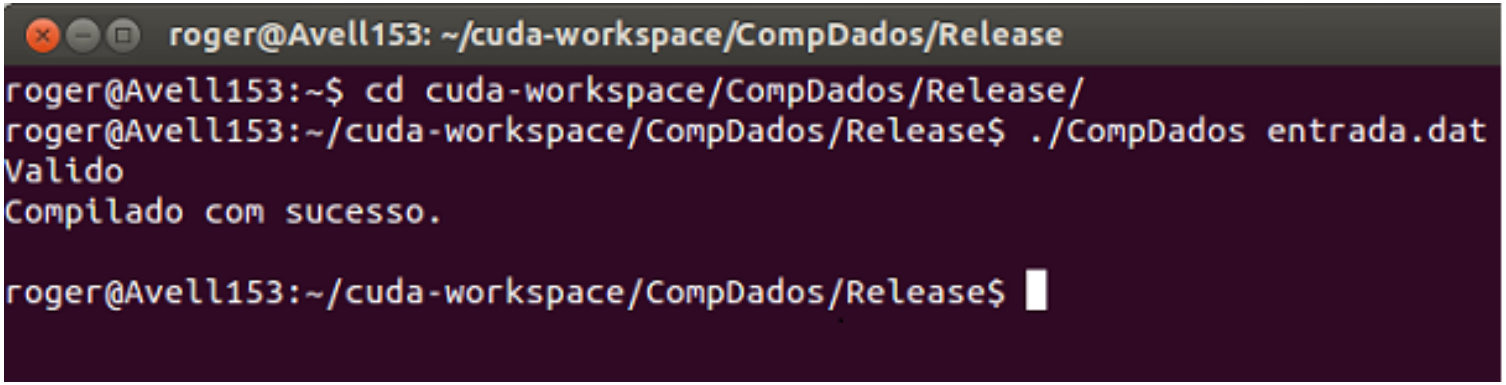
Fluxo Simplex

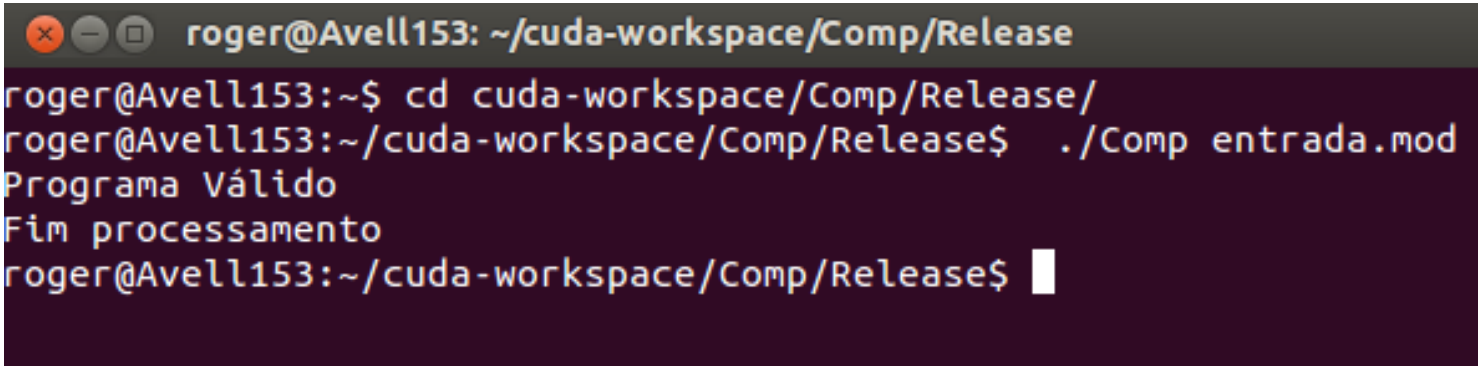
```
1   while(true) {
2     k = findIndiceK(SimplexTableau, COLUNA);
3     Kernel1<<<1,LINHA>>>(dev_tableau, dev_theta, dev_columK, k, COLUNA);
4     cudaMemcpy(teta, dev_theta, sizeColumn, cudaMemcpyDeviceToHost);
5     r = findIndiceR(teta, LINHA);
6     if(r == -1) {
7       printf("Problema Unbounded");
8       break;
9     }
10    Kernel2<<<COLUNA, 1>>>(dev_tableau, dev_columK, k, r, COLUNA);
11    cudaMemcpy(Columnk, dev_columK, sizeColumn, cudaMemcpyDeviceToHost);
12
13    Kernel3<<<LINHA, COLUNA >>>(dev_tableau, dev_columK, k, r, COLUNA);
14    cudaMemcpy(Columnk, dev_columK, sizeColumn, cudaMemcpyDeviceToHost);
15
16    Kernel4<<<1,LINHA>>>(dev_tableau, dev_columK, k, r, COLUNA);
17    cudaMemcpy(SimplexTableau,dev_tableau, sizeTableau,
18  cudaMemcpyDeviceToHost);
19   if(stop(SimplexTableau, COLUNA)) {
20      printSolution(SimplexTableau);
21      break;
22    }
23    if(iteration > LINHA ) {
24      printf("Não foi possível encontrar solucao");
25      break;
26    }
27     iteration++;
28  }
```

Operacionalidade

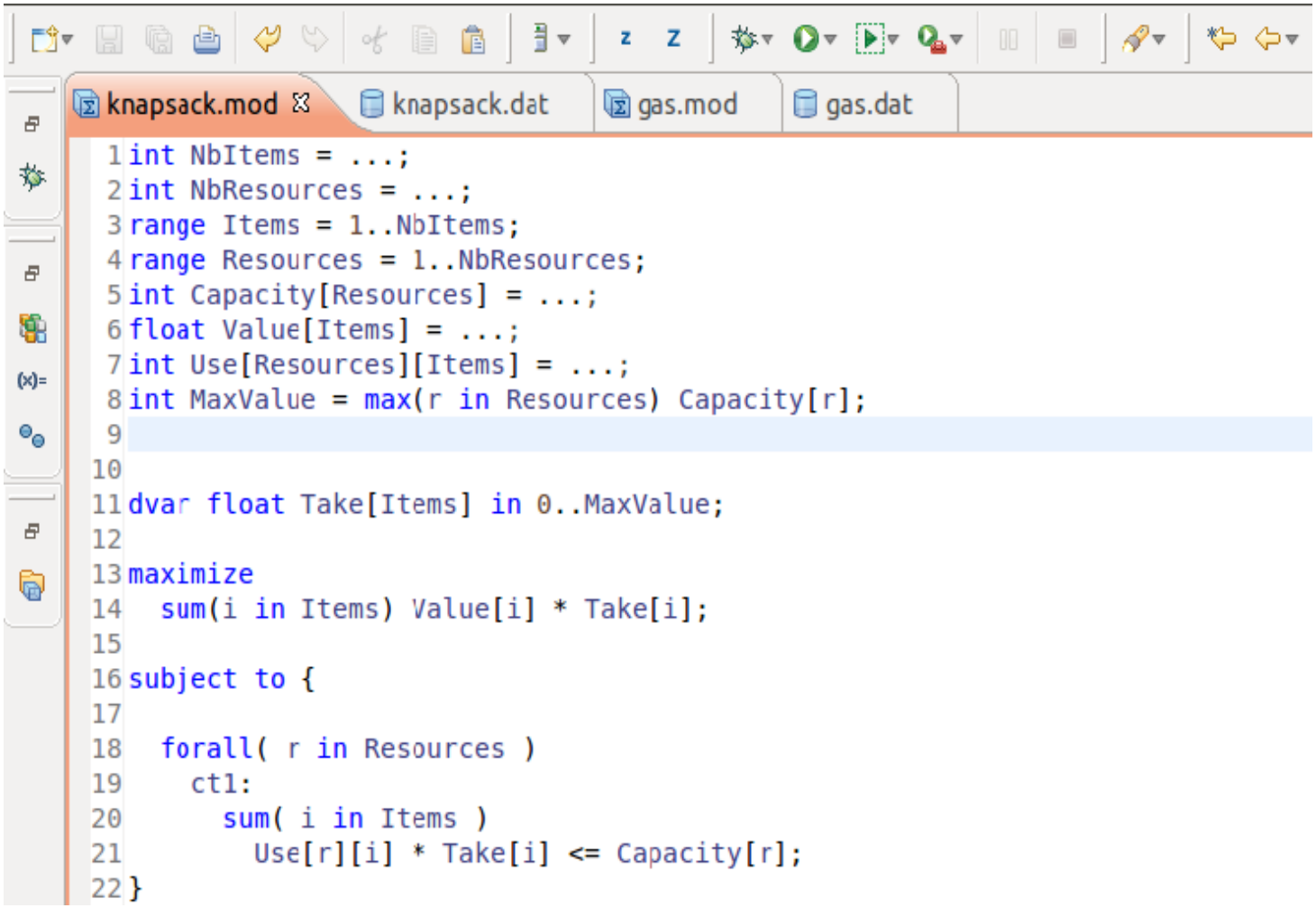
- Comp e CompDados, chamar o programa passando como parâmetro o caminho do arquivo fonte, saída do processamento no console.


Comp - C/C++ - Comp/src/Comp.cpp - Nsight
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explor
Comp
Binaries
Includes
src
Script
Comp.cpp
Debug
CompDados
GeradorDados
TesteCuda
Comp.cpp
2 // Name : Comp.cpp
7 #include <iostream>
8 #include <cstdlib>
9 #include "Script/Semantico/driver.hpp"
10
11 int main(const int argc, const char **argv){
12 if(argc != 2)
13 return (EXIT_FAILURE);
14
15 CC::Driver driver;
16 driver.parse(argv[1]);
17 driver.gerarCodigo();
18
19 std::cout << "Fim processamento" << endl;
20 return(EXIT_SUCCESS);
Console
<terminated> Comp [C/C++ Application] /home/roger/cuda-workspace/Comp/Debug/Comp (12/5/13 11:36 AM)
Programa Válido
Fim processamento

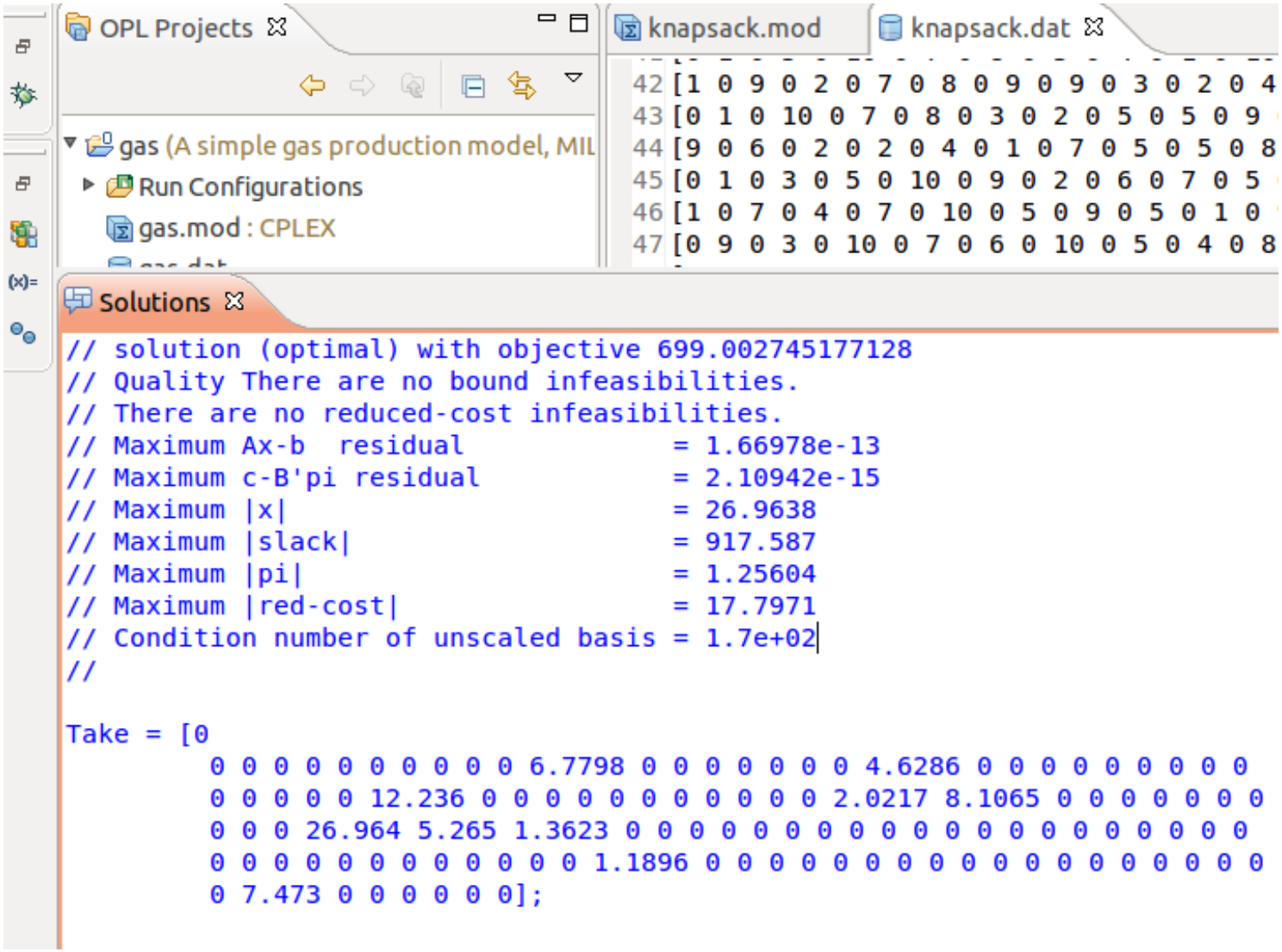

roger@Avell153: ~/cuda-workspace/CompDados/Release
roger@Avell153:~$ cd cuda-workspace/CompDados/Release/
roger@Avell153:~/cuda-workspace/CompDados/Release$ ./CompDados entrada.dat
Valido
Compilado com sucesso.
roger@Avell153:~/cuda-workspace/CompDados/Release$


roger@Avell153: ~/cuda-workspace/Comp/Release
roger@Avell153:~$ cd cuda-workspace/Comp/Release/
roger@Avell153:~/cuda-workspace/Comp/Release$ ./Comp entrada.mod
Programa Válido
Fim processamento
roger@Avell153:~/cuda-workspace/Comp/Release$

Knapsack (mochila)



Execução


OPL Projects
gas (A simple gas production model, MIL
Run Configurations
gas.mod : CPLEX
knapsack.mod
knapsack.dat
42 [1 0 9 0 2 0 7 0 8 0 9 0 9 0 3 0 2 0 4
43 [0 1 0 10 0 7 0 8 0 3 0 2 0 5 0 5 0 9
44 [9 0 6 0 2 0 2 0 4 0 1 0 7 0 5 0 5 0 8
45 [0 1 0 3 0 5 0 10 0 9 0 2 0 6 0 7 0 5
46 [1 0 7 0 4 0 7 0 10 0 5 0 9 0 5 0 1 0
47 [0 9 0 3 0 10 0 7 0 6 0 10 0 5 0 4 0 8
Solutions
// solution (optimal) with objective 699.002745177128
// Quality There are no bound infeasibilities.
// There are no reduced-cost infeasibilities.
// Maximum Ax-b residual = 1.66978e-13
// Maximum c-B'pi residual = 2.10942e-15
// Maximum |x| = 26.9638
// Maximum |slack| = 917.587
// Maximum |pi| = 1.25604
// Maximum |red-cost| = 17.7971
// Condition number of unscaled basis = 1.7e+02
//
Take = [0
0 0 0 0 0 0 0 0 0 6.7798 0 0 0 0 0 0 0 4.6286 0 0 0 0 0 0 0 0 0
0 0 0 0 0 12.236 0 0 0 0 0 0 0 0 0 0 0 2.0217 8.1065 0 0 0 0 0 0 0
0 0 0 26.964 5.265 1.3623 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1.1896 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 7.473 0 0 0 0 0 0];

Resultado execução

```
restricoes =70 objetivos =120
Alocaçao ok


Solução  = 699.002745
Bases
x11 = 6.779755
x19 = 4.628553
x34 = 12.235507
x46 = 2.021739
x47 = 8.106491
x58 = 26.963768
x59 = 5.265008
x60 = 1.362319
x93 = 1.189646
x113 = 7.472989

Ok
```

Resultados

- Os resultados obtidos alcançaram os objetivos.
- No entanto, não foi possível tratar totalmente a linguagem com a ferramenta desenvolvida.

Comparativo dos resultados

variável	IBM ILOG CPLEX	ferramenta
Total	699.002745177	699.002745
x11	6.7798	6.779755
x19	4.6286	4.628553
x34	12.236	12.235507
x46	2.0217	2.021739
x47	8.1065	8.106491
x58	26.964	26.963768
x59	5.265	5.265008
x60	1.3623	1.362319
x93	1.1896	1.189646
x113	7.473	7.472989

Comparativo Correlatos

característica / ferramenta	Gesser (2007)	**Lalami, Boyer, El-Baz (2011)**	**Moreira, Meneses (2012)**	**ferramenta proposta**
linguagem fonte	Java	-	-	OPL
linguagem objeto	C++	C++	C++	C++
plataforma de execução	Palm OS	CPU + GPU	CPU + GPU	CPU + GPU
algoritmo de otimização combinatória	não se aplica	Simplex	Tabu Search	Simplex

Conclusão

- O objetivo principal do trabalho foi alcançado com sucesso. O compilador desenvolvido mostrou-se capaz de detectar os erros léxicos, sintáticos e semânticos nos programas testados.
- A execução do programa gerado pela ferramenta apresentou precisão aceitável nos resultados comparados à ferramenta IBM ILOG CPLEX.

Extensões

- Implementar as construções OPL faltantes neste trabalho.
- Implementar outros algoritmos de resolução de problemas de análise combinatória
- Suportar entrada de dados de banco de dados e arquivos do Excel.
- Portar o algoritmo Simplex para outras plataformas de placas de vídeo (AMD, Intel)
- Automatizar a compilação e a execução.

- Demonstração.