Aplicação da Análise de Complexidade na Exploração do Paralelismo na Programação em Lógica

Jorge Luis Victória Barbosa[1,2] Patrícia Kayser Vargas[2] Cláudio Fernando Resin Geyer[2]

[1] Universidade Católica de Pelotas - Escola de Informática
Caixa Postal 402 - CEP 96010-000
Pelotas, RS, Brasil
barbosa@atlas. ucpel.tche.br

[2] Universidade Federal do Rio Grande do Sul - Instituto de Informática
Caixa Postal 15064 - CEP 91591-970
Porto Alegre, RS, Brasil
{barbosa, kayser, geyer}@inf.ufrgs.br

Resumo

Este artigo apresenta um estudo sobre a aplicação da análise de complexidade na paralelização de programas em lógica. Inicialmente, o texto descreve a estrutura do módulo Analisador de Complexidade proposto pelo modelo GRANLOG. Este modelo realiza a análise de granulosidade de programas em lógica. Logo após, o artigo apresenta resultados obtidos durante a avaliação de um analisador de complexidade, denominado CASLOG. Finalmente, o texto descreve a aplicação destes resultados no aperfeiçoamento do escalonamento no modelo OPERA. Este modelo realiza a execução paralela de programas em lógica.

Palavras Chaves: Análise de Complexidade, Programação em Lógica e Processamento Paralelo.

Abstract

This paper presents a study about the application of the complexity analysis in logic programs parallelization. First of all, the text describes the structure of Complexity Analyzer module proposed by GRANLOG model. This model makes granularity analysis of logic programs. After that, the paper presents the results obtained during the valuation of a complexity analyzer called CASLOG. Finally, it describes the application of these results in the improvement of scheduling in OPERA model. This model executes logic programs in parallel.

Keywords: Complexity Analysis, Logic Programming and Parallel Processing.

1 Introdução

A **complexidade computacional de um programa** consiste do montante de recursos computacionais, tais como tempo e espaço de memória, consumidos durante sua execução. Por sua vez, a **análise de complexidade** consiste na inferência de informações a respeito da complexidade computacional de um programa através do exame do seu texto. Normalmente, a análise de complexidade dedica-se à previsão do tempo necessário para execução de um programa.

Atualmente, a comunidade científica vem dedicando esforços significativos para automatização dessa análise, criando assim a **análise automática de complexidade.**

A metodologia a ser empregada na análise de complexidade depende do paradigma de programação no qual foram desenvolvidos os programas a serem analisados. Os primeiros trabalhos sobre este tema dedicaram-se ao estudo da complexidade no paradigma tradicional (imperativo). Em trabalhos posteriores, foram pesquisados os paradigmas de programação funcional e lógica. Em [LIN93] encontra-se uma interessante retrospectiva sobre a análise de complexidade nos paradigmas de programação imperativa, funcional e lógica. Diversos trabalhos de pesquisa exploram a análise de complexidade na programação em lógica [DEB 93] [LIN 93] [DEB 94] [GAR 94] [BAR 96] [BAR 97] [SHE 97].

A análise de complexidade encontra diversas aplicações no universo dos computadores. No texto [LIN 93] são discutidas genericamente várias aplicações e especificamente duas delas para a programação em lógica, ou seja, a exploração do paralelismo e a otimização de programas. Conforme constata-se em [DEB 93] e [LIN 93], a exploração do paralelismo é uma das principais aplicações para a análise de complexidade.

Este trabalho dedica-se a análise de complexidade na programação em lógica e sua aplicação na exploração do paralelismo durante a execução de programas em lógica. O texto descreve o módulo Analisador de Complexidade (AC) proposto pelo modelo GRANLOG [BAR 00]. Além disso, são apresentados resultados de experimentos realizados para avaliação do modelo CASLOG [LIN 93]. Este modelo é utilizado na implementação do módulo AC. Durante a discussão destes experimentos, destacam-se as conclusões relacionadas com a exploração do paralelismo na programação em lógica. Finalmente, são apresentadas conclusões alcançadas durante a aplicação das informações de complexidade no aperfeiçoamento do modelo OPERA [WER 94]. Nesta fase do texto, destaca-se a determinação dos custos de comunicação para exploração do paralelismo em sistemas com memória distribuída.

O texto está organizado em cinco seções. A segunda seção apresenta o módulo Analisador de Complexidade do GRANLOG. A terceira seção discute os resultados dos experimentos realizados com o CASLOG. Esta seção enfatiza a importância da análise de complexidade na paralelização de programas em lógica. Por sua vez, a quarta seção apresenta os resultados da aplicação das informações de complexidade no aperfeiçoamento do escalonamento no OPERA. Finalmente, a quinta seção apresenta as conclusões do trabalho.

2 Análise no GRANLOG: Módulo Analisador de Complexidade

Conforme apresentado em [BAR 00], o modelo GRANLOG é organizado em três módulos, ou seja, Analisador Global [AZE 99], Analisador de Grãos [BAR 96] e Analisador de Complexidade.

A figura 2.1 mostra uma visão genérica do módulo Analisador de Complexidade. Nesta figura, verifica-se que o módulo possui como entrada o **programa particionado** e como saída o **programa granulado**. O programa particionado constitui-se do programa em lógica acrescido de uma anotação contendo informações de paralelismo. Por sua vez, o programa granulado constitui-se do programa particionado acrescido de uma anotação contendo informações de complexidade. O programa granulado é o produto final do GRANLOG.

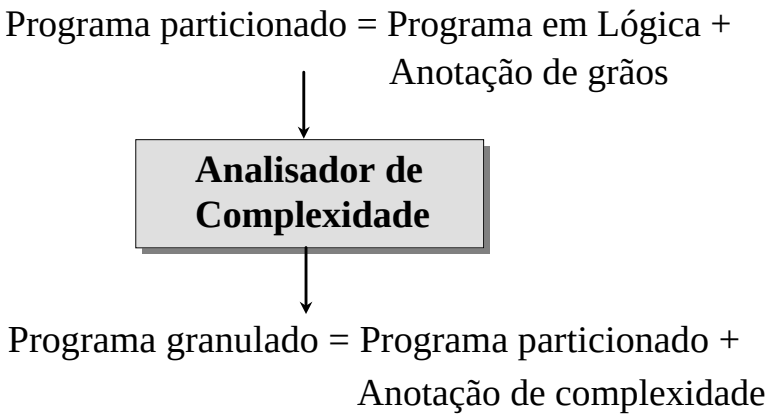

Figura 2.1 - Visão geral do módulo Analisador de Complexidade

Na figura 2.2 é apresentada a estrutura interna do módulo AC. O módulo AC é constituído por três submódulos, ou seja: **Analisador de Complexidade E** (ACE), **Analisador de Complexidade OU** (ACO) e **Gerador de Anotação de Complexidade** (GAC). O GRANLOG utiliza dois tipos de complexidade para a programação em lógica, ou seja, a complexidade E e a complexidade OU [BAR 97]. Define-se **complexidade OU** como o montante de recursos computacionais consumidos durante a execução de um caminho da árvore de busca. Por sua vez, define-se **complexidade E** como o montante de recursos computacionais consumidos durante a execução de uma parte do corpo de uma cláusula. O submódulo ACE realiza a análise de complexidade E, gerando informações para o submódulo GAC. O submódulo ACO realiza a análise de complexidade OU, fornecendo informações para o submódulo GAC. Finalmente, o submódulo GAC cria a anotação de complexidade, gerando assim o programa granulado.

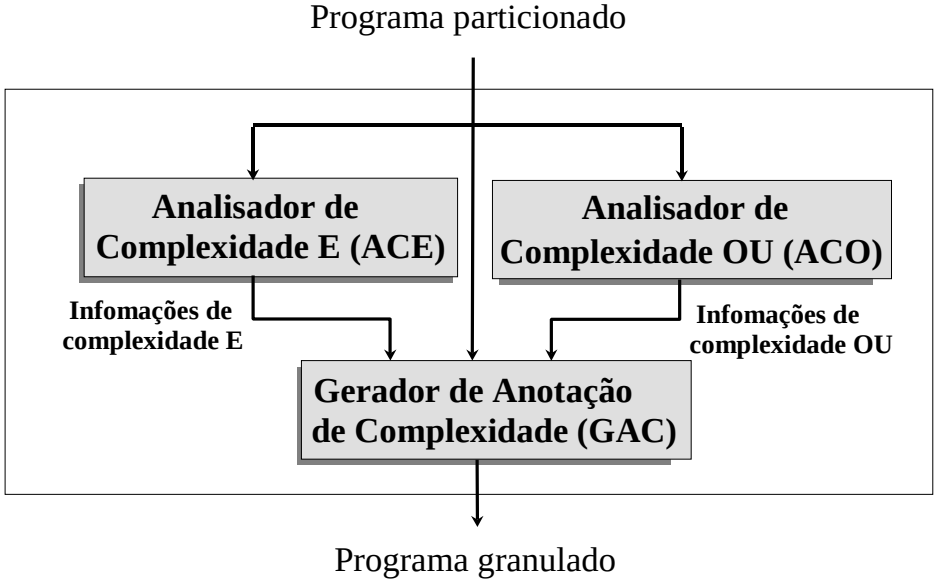

Figura 2.2 - Organização interna do módulo Analisador de Complexidade

O ACE possui como base o sistema proposto em [DEB 93] e [LIN 93], denominado CASLOG (*Complexity Analysis System for LOGic*). O CASLOG realiza uma análise estática de um programa em lógica e produz para cada procedimento uma expressão de complexidade. Esta expressão

representa a complexidade do procedimento em função do tamanho de suas entradas. Essas expressões podem ser resolvidas durante a execução para determinar com precisão a complexidade de cada procedimento do programa. Além disso, deve-se ressaltar que o CASLOG realiza uma análise de complexidade do pior caso, considerando o não-determinismo dos programas em lógica e o número de possíveis soluções que podem ser geradas durante a execução. A próxima seção apresenta resultados obtidos através de avaliações do CASLOG e discute sua aplicação na exploração do paralelismo na programação em lógica.

O módulo ACO e a metodologia utilizada na geração de informações de complexidade OU é apresentada em [BAR 97]. Por isso, apresenta-se nas próximas seções apenas a discussão sobre a geração de informações de complexidade E (submódulo ACE) e a aplicação dessas informações na exploração do paralelismo.

3 Análise de Complexidade E: Avaliação do CASLOG

A figura 3.1 apresenta as expressões de complexidade geradas pelo CASLOG para os programas *append*, *nrev*, *fibonacci* e *hanoi*. No CASLOG, a geração de expressões de complexidade depende dos modos e medidas atribuídos aos argumentos dos procedimentos.

Cappend = *$1* + 1
Cnrev = 0.5 * exp(*$1*,2) + 1.5 * *$1* + 1
Cfibonacci = 1.45 * exp(1.62,*$1*) + 0.55 * exp(-0.62,*$1*) - 1
Chanoi = *$1* * exp(2,*$1*) + exp(2,*$1*-1) - 2

Figura 3.1 - Expressões de Complexidade

A notação das expressões utiliza operadores matemáticos (*,/,+,-), funções matemáticas (por exemplo, *exp(a,b)* significa a elevado na potência b) e o símbolo *$* significando o tamanho de determinado argumento do procedimento (*$1* por exemplo, significa o tamanho do primeiro argumento). As complexidades dos quatro procedimentos utilizados para obtenção das expressões mostradas na figura 3.1 dependem apenas do tamanho primeiro argumento (*$1*).

As expressões de complexidade geradas pelo CASLOG utilizam como medida de complexidade o número de resoluções, ou seja, a solução de uma expressão resulta no número de resoluções (chamadas) que serão realizadas durante a execução de um procedimento.

Durante o desenvolvimento do GRANLOG foram realizados experimentos para avaliação da precisão das informações de complexidade geradas pelo CASLOG. Os experimentos foram realizados em uma estação SUN SLC no laboratório do Instituto de Informática da Universidade Federal do Rio Grande do Sul (UFRGS) e consistiram dos seguintes passos:

1. Submissão dos procedimentos *append*, *nrev*, *fibonacci* e *hanoi* à análise de complexidade do CASLOG. Esse passo gera as expressões de complexidade apresentadas na figura 3.1;

2. Execução dos quatro procedimentos em três ambientes diferentes, ou seja, C-Prolog, Sicstus Prolog e o emulador WAM seqüencial utilizado no projeto OPERA-E [WER 94]. Cada procedimento foi executado com vários tamanhos para o argumento de entrada, gerando assim, diversas previsões de complexidade e diversos tempos de execução. As tabelas 3.1, 3.2, 3.3 e 3.4 mostram os resultados obtidos nos passos 2, 3, 4 e 5. Cada tabela

contém os resultados do teste de um procedimento. A primeira coluna contém o tamanho da entrada utilizada em cada execução (valor de *$1*). A terceira coluna contém o tempo da execução em milisegundos para o Sicstus, a quinta coluna para o C-Prolog e a sétima para o emulador WAM. As demais colunas são preenchidas pelos próximos passos;

3. Resolução das expressões de complexidade para cada entrada (coluna 1) utilizada no passo 2, ou seja, obtenção do número de resoluções previstas pelo CASLOG para cada execução. Este passo resulta na segunda coluna das quatro tabelas;

4. Obtenção do tempo necessário para uma resolução na execução dos procedimentos em cada ambiente. Este valor é obtido com a divisão do tempo total de execução (coluna 3, 5 e 7) pelo número de resoluções previstas pelo CASLOG (coluna 2). Os resultados são apresentados nas colunas 4, 6 e 8;

5. Obtenção do tempo médio de uma resolução para execução de um procedimento em um determinado ambiente, ou seja, obtenção da média para cada uma das colunas 3, 5 e 7. As médias são apresentadas na última linha das tabelas.

Tabela 3.1 - Teste do CASLOG com procedimento *append*

APPEND							
TAM.	**RES.**	**SICSTUS**		**C-PROLOG**		**WAM**	
		Tempo (ms)	**Tempo/Res.**	**Tempo (ms)**	**Tempo/Res.**	**Tempo (ms)**	**Tempo/Res.**
10	11	0.95	0.09	0.83	0.08	4.00	0.36
20	21	1.45	0.07	2.50	0.12	7.00	0.33
30	31	3.00	0.10	3.33	0.11	10.00	0.32
40	41	3.45	0.08	5.00	0.12	16.00	0.39
50	51	3.95	0.08	6.66	0.13	19.00	0.37
60	61	4.95	0.08	7.50	0.12	22.00	0.36
70	71	5.45	0.08	8.33	0.12	25.00	0.35
80	81	6.50	0.08	9.17	0.11	29.00	0.36
90	91	6.95	0.08	10.00	0.11	33.00	0.36
100	101	8.45	0.08	11.67	0.12	36.00	0.36
1000	1001	82.00	0.08	121.67	0.12	349.00	0.35
Média Tempo/Resoluções			0.08		0.12		0.35

Tabela 3.2 - Teste do CASLOG com procedimento *nrev*

NREV							
TAM.	**RES.**	**SICSTUS**		**C-PROLOG**		**WAM**	
		Tempo (ms)	**Tempo/Res.**	**Tempo (ms)**	**Tempo/Res.**	**Tempo (ms)**	**Tempo/Res.**
10	66	6.50	0.10	7.50	0.11	27.00	0.41
20	231	20.50	0.09	24.17	0.10	87.00	0.38
30	496	41.50	0.08	56.67	0.11	185.00	0.38
40	861	71.00	0.08	101.67	0.12	313.00	0.36
50	1326	108.50	0.08	155.00	0.12	482.00	0.36
60	1891	155.95	0.08	220.83	0.12	685.00	0.36
70	2556	209.45	0.08	295.83	0.12	919.00	0.36
80	3321	271.95	0.08	391.67	0.12	1201.00	0.36
90	4186	341.50	0.08	493.33	0.12	1513.00	0.36
100	5151	420.95	0.08	605.00	0.12	1864.00	0.36
Média Tempo/Resoluções			0.08		0.12		0.36

Tabela 3.3 - Teste do CASLOG com procedimento *fibonacci*

FIBONACCI							
TAM.	RES.	SICSTUS		C-PROLOG		WAM	
		Tempo (ms)	Tempo/Res.	Tempo (ms)	Tempo/Res.	Tempo (ms)	Tempo/Res.
1	2	0.00	0.00	0.00	0.00	1.00	0.50
2	3	1.00	0.33	0.83	0.28	3.00	1.00
3	5	1.45	0.29	1.67	0.33	6.00	1.20
4	9	2.45	0.27	2.45	0.27	12.00	1.33
5	15	4.50	0.30	5.00	0.33	19.00	1.27
6	25	7.45	0.30	7.50	0.30	32.00	1.28
7	41	11.95	0.29	13.33	0.32	51.00	1.24
8	67	18.95	0.28	22.50	0.30	83.00	1.24
9	109	31.45	0.29	35.00	0.32	138.00	1.27
10	177	51.50	0.29	58.33	0.33	225.00	1.27
11	287	85.95	0.30	94.17	0.33	357.00	1.24
12	465	140.95	0.30	152.50	0.33	584.00	1.26
13	753	225.00	0.30	248.33	0.33	949.00	1.26
14	1218	364.95	0.30	402.50	0.33	1534.00	1.26
15	1972	593.50	0.30	649.17	0.33	2479.00	1.26
Média Tempo/Resoluções			0.30		0.33		1.26

Tabela 3.4 - Teste do CASLOG com procedimento *hanoi*

HANOI							
TAM.	RES.	SICSTUS		C-PROLOG		WAM	
		Tempo (ms)	Tempo/Res.	Tempo (ms)	Tempo/Res.	Tempo (ms)	Tempo/Res.
1	1	0.00	0.00	0.00	0.00	0.00	0.00
2	8	1.00	0.12	0.83	0.10	4.44	0.56
3	26	3.00	0.11	3.33	0.13	15.56	0.60
4	70	8.45	0.12	10.00	0.14	35.56	0.51
5	174	18.50	0.11	22.50	0.13	120.00	0.69
6	414	43.00	0.10	54.17	0.13	217.78	0.53
7	958	99.50	0.10	125.00	0.13	468.89	0.49
8	2174	220.00	0.10	279.17	0.13	1024.44	0.47
9	4862	486.00	0.10	618.33	0.13	2253.33	0.46
10	10750	1054.45	0.10	1395.00	0.13	4887.50	0.45
Média Tempo/Resoluções			0.10		0.13		0.53

A análise das informações apresentadas nas tabelas permite as seguintes considerações:

1. Durante a execução de um procedimento em um determinado ambiente, a complexidade de resolução mantém-se praticamente constante. A estabilidade da complexidade de resolução de um procedimento em um ambiente permite previsões precisas do tempo envolvido na execução;
2. A complexidade de uma resolução para um procedimento varia de acordo com o ambiente de execução (*software*). Sendo assim, não é possível utilizar para todos os ambientes a mesma complexidade para uma resolução em um procedimento;
3. A complexidade de uma resolução para diferentes procedimentos executados no mesmo ambiente pode ser diferente. Portanto, não é possível utilizar para todos os procedimentos um único valor para complexidade de uma resolução em um ambiente.

4 Análise de Complexidade E: Aplicação no Paralelismo

Esta seção apresenta os resultados da aplicação das informações de complexidade (discutidas na seção anterior) na integração dos modelos OPERA e GRANLOG. O OPERA é um modelo para

exploração do paralelismo na execução de programas em lógica. Por sua vez, o GRANLOG fornece informações de granulosidade para aperfeiçoamento do escalonamento no OPERA.

Atualmente, o sistema OPERA possui suporte para controle de granulosidade. A introdução deste controle ocasionou duas alterações no sistema, ambas no código do processo responsável pela execução de programas Prolog (processo *Solver*). Este processo implementa uma versão da *Warren Abstract Machine* (WAM [AIT 91]). Em primeiro lugar, sempre que uma nova aplicação é carregada para execução, o processo *Solver* carrega também as informações de granulosidade da aplicação. Em segundo lugar, o código que implementa a instrução WAM responsável pela paralelização de objetivos (instrução *push_call*) foi alterado para utilização das informações de granulosidade. Na versão original do OPERA, a instrução *push_call* simplesmente empilha um objetivo em uma *Pilha de Objetivos Paralelos* (POP), sem considerações quanto a complexidade das tarefas paralelas e o custo de paralelização. A nova versão implementa o algoritmo apresentado na figura 4.1.

```
se Tamanho do Argumento > Tamanho Limite  então
        Empilha objetivo na POP (paralelização)
senão
        Executa o objetivo localmente
```

Figura 4.1 - Algoritmo para controle de granulosidade no OPERA

Atualmente, o sistema OPERA manipula apenas procedimentos que tenham sua complexidade dependente de apenas um argumento de entrada. Diversos procedimentos enquadram-se nesse padrão. Por exemplo, analisando-se as expressões de complexidade mostradas na figura 3.1, constata-se que a complexidade dos procedimentos *append*, *nrev*, *fibonacci* e *hanoi* depende de apenas um argumento de entrada.

O **Tamanho do Argumento** somente poderá ser obtido em tempo de execução. A dificuldade para obtenção desse tamanho depende da medida empregada. Por exemplo, se o argumento for um inteiro, o tamanho é o valor numérico do argumento e pode ser obtido diretamente nos registradores da WAM. Por outro lado, se o argumento for uma lista, a obtenção do tamanho envolverá uma análise mais complexa. Em [HER 94] é proposta uma metodologia para realização dessa análise. Atualmente, o protótipo OPERA somente executa em paralelo procedimentos com argumentos do tipo inteiro.

O **Tamanho Limite** para um procedimento depende do custo para paralelização de um objetivo no ambiente OPERA (**Custo Limite**). Basicamente, esse custo consiste do tempo necessário para troca de mensagens entre os trabalhadores durante o processos de paralelização de um procedimento. Para realização desse processo, o OPERA utiliza quatro mensagens, conforme mostra a figura 4.2.

A primeira mensagem consiste da **exportação do trabalho**, ou seja, o exportador envia para o importador os dados necessários para execução do procedimento. A segunda mensagem contém um **aviso de sucesso ou falha** na execução do procedimento. A terceira mensagem é uma **requisição dos resultados**, ou seja, o exportador solicita ao importador os resultados do processamento. Finalmente, o importador envia os **resultados**.

Figura 4.2 - Troca de mensagens para paralelização no OPERA

O custo envolvido na troca de mensagens depende do montante de informações a serem trocadas entre trabalhadores, ou seja, depende do tamanho das mensagens. As únicas mensagens que variam de tamanho são a **exportação de trabalho** e os **resultados**. O tamanho da mensagem **exportação de trabalho** está relacionado com o tamanho dos argumentos de entrada do procedimento. Esse tamanho pode ser obtido em tempo de execução. O tamanho da mensagem **resultados** está relacionada com o tamanho dos argumentos de saída do procedimento. Esse tamanho pode ser obtido, antes da paralelização do procedimento, através das relações armazenadas nas anotações *out_size* fornecidas pelo GRANLOG [BAR 00]. Resumindo, o custo para paralelização de um procedimento pode variar de acordo com o tamanho dos seus argumentos de entrada e saída, os quais somente podem ser determinados em tempo de execução. Além disso, esse custo pode ser previsto, antes da execução paralela, através das relações de tamanho entre entradas e saídas.

Atualmente o OPERA somente exporta procedimentos com argumentos do tipo inteiro. Portanto, o volume de informações trocadas entre trabalhadores não varia de forma significativa. Sendo assim, o custo para paralelização de um procedimento pode ser considerado constante. Em horário de baixa utilização da rede na UFRGS e utilizando duas estações SUN SLC, foram realizados diversos experimentos que permitiram o dimensionamento deste custo. **O OPERA possui um Custo Limite de 0.013455 segundos**. Com base neste custo, pode-se determinar o Tamanho Limite para os procedimentos a serem executados no OPERA. Por exemplo, para o procedimento *fibonacci* pode ser realizada a seguinte análise.

Na tabela 3.3 obtém-se que na WAM seqüencial utilizada no OPERA uma resolução do *fibonacci* possui uma complexidade de 1.26 milisegundos. Além disso, conforme demonstrado em [YAM 94], a degradação de desempenho da execução do *fibonacci* na WAM paralela em relação a execução na WAM seqüencial é aproximadamente 20 %. Desta forma, a complexidade de uma resolução do *fibonacci* na WAM paralela é aproximadamente 1.512 ms. Portanto, o custo limite em resoluções pode ser obtido pelo seguinte cálculo:

Custo Limite em Resoluções = Custo Limite / Complexidade de uma Resolução

Custo Limite em Resoluções = 0.013455 / 0.001512 = 8.8988 resoluções

Além disso, na figura 3.1 obtém-se a expressão de complexidade para o procedimento *fibonacci*.

Cfibonacci = 1.45 * exp(1.62,*$1*) + 0.55 * exp(-0.62,*$1*) - 1

Essa expressão determina a complexidade em resoluções. Portanto, com base no **Custo Limite em Resoluções** e na **Expressão de Complexidade**, determina-se o **Tamanho Limite** para o argumento de entrada que influencia na complexidade (*$1*).

Cfibonacci > Custo Limite em Resoluções

1.45 * exp(1.62,*$1*) + 0.55 * exp(-0.62,*$1*) - 1 > 8.8988

Se *$1* = 3 então Cfibonacci = 5 resoluções

Se *$1* = 4 então Cfibonacci = 9 resoluções

Portanto, o **Tamanho Limite** para o procedimento *fibonacci* é 3, ou seja, o procedimento *fibonacci* somente deverá ser exportado (colocado na POP) quando o tamanho do seu argumento de entrada for maior do que 3. Quanto isso ocorrer, a complexidade para execução do procedimento supera o custo para sua paralelização.

5 Conclusões

Este artigo apresentou um estudo relacionado com a aplicação da análise de complexidade na paralelização de programas em lógica. Destacaram-se no decorrer do texto a descrição do módulo Analisador de Complexidade proposto pelo modelo GRANLOG, os resultados obtidos com experimentos do modelo CASLOG e conclusões obtidas durante a aplicação da análise de complexidade no aperfeiçoamento do escalonamento no modelo OPERA. Entre as principais conclusões e constatações deste artigo encontram-se as seguintes:

- o estudo do CASLOG demonstrou que a complexidade de uma resolução para um procedimento sofre diversas instabilidades que devem ser dimensionadas para avaliação precisa da complexidade nos programas em lógica;
- a utilização da medida de complexidade **resolução** introduz imprecisão na análise de complexidade;
- a complexidade de uma resolução para um procedimento varia de acordo com o poder computacional do processador (*hardware*). Desta forma, não é possível utilizar em arquiteturas que possuam processadores com poder computacional diferente, ainda que com o uso do mesmo ambiente, uma mesma complexidade para uma resolução em um procedimento;
- um dos próximos passos de pesquisa no âmbito da análise de complexidade na programação em lógica deve ser o desenvolvimento de novas medidas de complexidade;
- a análise de complexidade é indispensável para realização da análise de granulosidade, a qual depende de comparações de custos de paralelização e complexidade dos grãos.

Recentemente, surgiram várias propostas para aplicação do GRANLOG em sistemas relacionados com exploração do paralelismo na programação em lógica. Neste contexto, destacam-se as propostas para os sistemas PLoSys [FER 99], Andorra-I [DUT 99] e DSLP [COS 98]. Trabalhos futuros poderão aperfeiçoar o estudo apresentado neste artigo. Os experimentos realizados com o CASLOG serão aperfeiçoados através da análise de mais procedimentos. Novas medidas de complexidade deverão ser desenvolvidas, permitindo assim, mais precisão para as

informações de complexidade. A imprecisão introduzida pela medida resolução é um problema que deve ser resolvido.

Referências Bibliográficas

[AIT 91] AÏT-KACI, Hassan. **Warren's Abstract Machine - A Tutorial Reconstruction**. Cambridge: MIT Press, 1991. 114p.

[AZE 99] AZEVEDO, Silvana C. de; GEYER, Cláudio F. R.; BARBOSA, Jorge L. V. **Automatização da Análise Global no Modelo GRANLOG.** Congresso LatinoAmericano de Informática (CLEI), Paraguai, v.1.; p. 601-612, 1999.

[COS 98] COSTA, Cristiano André Da; GEYER, Cláudio F. R. **Uma Proposta de Escalonamento Distribuído Para Exploração do Paralelismo na Programação em Lógica**. Simpósio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho, Búzios – RJ, COPPE/UFRJ, v.10, 1998.

[BAR 96] BARBOSA, Jorge L. V.; GEYER, Cláudio F. R. **Análise de Grãos na Programação em Lógica**. Anais do Seminário Integrado de Software e Hardware (SEMISH), Recife, SBC, p.345-356, Agosto 1996.

[BAR 97] BARBOSA, Jorge L. V.; GEYER, Cláudio F. R. **Análise de Complexidade na Programação em Lógica: Taxonomia, Modelo GRANLOG e Análise OU**. Anais do Congresso LatinoAmericano de Informática (CLEI), Chile, Novembro 1997.

[BAR 00] BARBOSA, Jorge L. V.; VARGAS, Patrícia Kayser; GEYER, Cláudio F. R.; DUTRA, Inês de Castro. **GRANLOG: An Integrated Granularity Analysis Model for Parallel Logic Programming**. Proceedings of CL2000 Workshop on Parallelism and Implementation, 2000. (to be published)

[DEB 93] DEBRAY, S. K. e LIN, N. **Cost Analysis of Logic Programs**. ACM Transactions on Programming Languages and Systems. New York, v.15, n.5, p.826-875, November 1993.

[DEB 94] DEBRAY, S. K.; GARCÍA, Pedro L.; HERMENEGILDO, Manuel; LIN, Nai-Wei. **Estimating the Computacional Cost of Logic Programs**. Proceedings of the International Static Analysis Symposium, Namur, Belgium, Springer-Verlag, 1994. (Lecture Notes in Computer Science, v.864).

[DUT 99] DUTRA, Inês C.; SANTOS COSTA, Vítor; BARBOSA, Jorge L. V.; GEYER, Cláudio F. R. **Using Compile-Time Granularity Information to Support Dynamic Work Distribution in Parallel Logic Programming Systems**. Simpósio Brasileiro de Arquitetura de Computadores - Processamento de Alto Desempenho, Natal – RN, SBC, p.248-254, 1999.

[FER 99] FERRARI, Débora N.; GEYER, Cláudio F. R.; BARBOSA, Jorge L. V. **Modelo de Integração PLoSys-GRANLOG: Aplicação da Análise de Granulosidade na Exploração do Paralelismo OU.** Congresso LatinoAmericano de Informática (CLEI), Asuncion – Paraguai, Universidad Autonoma De Asuncion, v. 2, p. 911-922, 1999.

[GAR 94] GARCÍA, Pedro L.; HERMENEGILDO, Manuel; DEBRAY, S. K. **Towards Granularity Based Control of Parallelism in Logic Programs**. Proceedings of the International Symposium on Parallel Computation, Linz, Austria, September 1994.

[HER 94] HERMENEGILDO, M.;GARCÍA, P.L. **A Technique for Dynamic Term Size Computation via Program Transformation**. Madrid, Universidad Politécnica de Madrid, March 1994. 20p. (Technical Report - CLIP 8/93.1).

[LIN 93] LIN, N. **Automatic Complexity Analysis of Logic Programs**. University of Arizona, Tucson, USA, 1993. 244p. (Ph.D. Thesis).

[SHE 97] SHEN, K.; SANTOS COSTA, V.; KING, A. **A New Metric for Controlling Granularity for Parallel Execution**. Proceedings of ILPS97 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages, Port Jefferson – USA, 1997.

[WER 94] WERNER, Otília; YAMIN, Adenauer C.; BARBOSA, Jorge L. V. e GEYER, Cláudio F. R. **OPERA Project: an Approach Towards Parallelism Exploitation on Logic Programming**. Proceedings of the Tenth Logic Programming Workshop, Zurich, Institut für Informatik der Univesität Zürich, September 1994.

[YAM 94] YAMIN, Adenauer C. **Um Ambiente para Exploração de Paralelismo na Programação em Lógica**. Porto Alegre, CPGCC-UFRGS, 1994. 204p. (Dissertação de Mestrado).