Quim. Nova, Vol. 25, No. 3, 465-469, 2002.

Divulgação

TÉCNICAS DE ANÁLISE DO PERFIL DE EXECUÇÃO E OTIMIZAÇÃO DE PROGRAMAS EM QUÍMICA COMPUTACIONAL#

André S. P. Gomes, Lucimara R. Martins e Pedro A. M. Vazquez*
Instituto de Química, Universidade Estadual de Campinas, CP 6154, 13083-970 Campinas - SP

Recebido em 25/4/01; aceito em 12/9/01

TECHNIQUES FOR THE EXECUTION PROFILE ANALYSIS AND OPTIMIZATION OF COMPUTATIONAL CHEMISTRY PROGRAMS. In this paper we review the basic techniques of performance analysis within the UNIX environment that are relevant in computational chemistry, with particular emphasis on the execution profile using the gprof tool. Two case studies (in *ab initio* and molecular dynamics calculations) are presented in order to illustrate how execution profiling can be used to effectively identify bottlenecks and to guide source code optimization. Using these profiling and optimization techniques it was possible to obtain significant speedups (of up to 30%) in both cases.

Keywords: performance analysis; computational chemistry; *ab initio* and molecular dynamics calculations.

INTRODUÇÃO

O avanço tecnológico dos microprocessadores ao longo da última década[1] trouxe aos microcomputadores pessoais níveis de desempenho computacional antes disponíveis apenas em computadores de grande porte. A este desempenho com baixo custo passou a associar-se, a partir de 1991, a disponibilidade de sistemas operacionais gratuitos de 32 bits, como o Linux[2] e o FreeBSD[3], com qualidade, características e desempenho análogos aos do UNIX[4-7]. Esta combinação de hardware e software estimulou diversos pesquisadores a iniciarem a adaptação de programas *ab initio* para estes sistemas. O programa GAMESS[8], por exemplo, foi adaptado ao FreeBSD por um dos autores (PAMV) em 1993[9-12] e logo em seguida ao Linux por Gulden. Em sucessão, o programa Gaussian94 foi objeto do mesmo processo de adaptação e, em 1995, a revisão D.2 já possuia suporte do fabricante para a (atualmente extinta) distribuição Yggdrasil do Linux e FreeBSD[13]. Paralelamente, inúmeros programas de química e física computacional foram adaptados para estes sistemas por diversos pesquisadores, de tal forma que, hoje em dia virtualmente qualquer programa de química computacional, cujo código fonte esteja acessível, encontra-se disponível para estes sistemas operacionais. À época destes esforços não existiam compiladores FORTRAN[14], a linguagem historicamente mais importante em química computacional, comerciais para estes sistemas operacionais, o compilador FORTRAN g77 da GNU[15] ainda encontrava-se nos primeiros estágios de seu desenvolvimento. O único compilador disponível e estável computacionalmente e numericamente era o tradutor de FORTRAN para C, f2c, da AT&T[16] usado em conjunto com o compilador C da GNU[17], gcc. A compilação era realizada em duas etapas: inicialmente o programa em FORTRAN era traduzido para C pelo f2c, a seguir este código em C era compilado com o gcc. Somente neste estágio é que ocorriam otimizações no programa e todas eram realizadas em torno da linguagem C gerada pelo conversor. Na segunda metade da década passada, ao mesmo tempo que o compilador FORTRAN g77 da GNU atingiu maturidade e estabilidade suficientes para seu uso em programas científicos de grande porte, começaram a surgir no mercado vários compiladores FORTRAN comerciais para o Linux. Estes compiladores comerciais, em associação com bibliotecas de álgebra linear otimizadas[18], mostraram-se extremamente eficientes para a otimização dos programas e, em alguns casos como a versão 98 do Gaussian que utiliza o compilador Portland[19], tornaram-se os compiladores de referência de vários fabricantes de programas de cálculos de estrutura molecular. Esta tendência de migração para compiladores comerciais, no entanto, foi contrabalançada com o surgimento de bibliotecas científicas gratuitas de alto desempenho como ATLAS[20], PhiPAC[21], FFTW[22] e pelo expressivo aumento de velocidade obtido na versão 2.95.2 da geração egcs do compilador gcc. Estudos recentes realizados por Morgon e colaboradores[23,24], verificaram que o uso criterioso de técnicas de otimização e análise de desempenho permite obter resultados comparáveis ou superiores aos obtidos com compiladores comerciais, como ilustra a Tabela 1.

O objetivo deste texto é apresentar as técnicas básicas de medida de tempo de processamento e construção de perfis de execução visando a otimização de programas de interesse da química computacional e utilizando compiladores públicos e gratuitos. Inicialmente serão apresentadas as técnicas básicas de medida de tempo de processamento. A seguir o programa gprof[1,25] será descrito e, por último, este programa será utilizado para a análise e otimização de dois programas de química computacional de natureza distinta. Cabe ressaltar que estas técnicas não estão restritas em sua aplicação à linguagem FORTRAN embora os exemplos utilizados tenham sido escritos nessa linguagem. Ressaltamos também que todo este trabalho foi desenvolvido em computadores de arquitetura Intel IA32 em ambiente FreeBSD, e que toda a discussão a seguir é válida para sistemas UNIX e suas variantes onde estas ferramentas estejam disponíveis independente da arquitetura de *hardware*.

TÉCNICAS DE MEDIDA DO TEMPO DE EXECUÇÃO

Existem diversas maneiras de medir o desempenho de um programa[1]. A mais simples de todas é a medida do tempo total de execução utilizando o comando /usr/bin/time do BSD UNIX ou o comando time embutido no interpretador (csh, bash). Estes comandos fornecem informações sobre o tempo total de execução (convencionalmente de-

*e-mail: vazquez@iqm.unicamp.br
Os autores dedicam este artigo à memória do Prof. Dr. Antonio Luiz Pires Valente

Tabela 1. Desempenho do programa Gaussian98 utilizando o compilador FORTRAN Portland (pgf) em combinação com a biblioteca ASCI-BLAS (ASCIblas) e utilizando o compilador GNU FORTRAN (g77) em combinação com a biblioteca ATLAS em um FreeBSD/Athlon/700 MHz

Tipo de Cálculo	Número de Funções de Base	Compilador e Biblioteca	Tempo
Otimização	144	pgf /ASCIblas	44m 21s
Hartree-Fock de Geometria	144	g77/ATLAS	35m 40s
Cálculo Hartree-Fock	108	pgf /ASCIblas	17m 41s
de frequências	108	g77/ATLAS	14m 38s
CISD	108	pgf /ASCIblas	10m 56s
	108	g77/ATLAS	10m 29s
CCSD(T)	90	pgf /ASCIblas	17m 5s
	90	g77/ATLAS	14m 23s

nominado *wall time*, que reflete o tempo real gasto na execução), o tempo gasto exclusivamente pelo processo para sua execução (convencionalmente denominado *user time*) e o tempo gasto exclusivamente em chamadas ao sistema operacional: paginação do executável e memória virtual[5,7,26], acesso a arquivos, etc. (convencionalmente denominado *system time*). Abaixo é ilustrada a saída do comando /usr/bin/time para a execução do programa flops[27], sobre o qual pode-se obter informações em um trabalho recente de um dos autores[28]:

```
318.44 real      157.08 user      0.25 sys
```

O relatório sumário deste comando informa apenas tempo total (aqui em segundos) decorrido entre o início e a real conclusão do programa, a fração deste tempo utilizada por ele (user) e a fração gasta em chamadas ao sistema operacional (sys). O comando time embutido no csh, por outro lado, fornece informações mais detalhadas, a saída deste comando para a execução do mesmo programa é ilustrada abaixo:

```
157.077u  0.226s  5:19.12 49.2%   15+179k  0+0io  0pf+0w
```

Nela, além das informações de *wall time* (em minutos), *user time* (u) e *system time* (s), também são fornecidas informações sobre o padrão de uso de memória compartilhada com o restante do sistema (aqui 49.2%) ou de uso privado em kbytes (15+179k), a quantidade de operações de leitura e escrita em disco (0+0io), o número de paginações de código executável (convencionalmente denominado *page faults*) e o número de operações de acesso ao espaço de *swap* do sistema de memória virtual (0pf+0w). Todas estas informações são muito úteis quando deseja-se avaliar o desempenho de um programa como um todo e o seu ajuste aos recursos de hardware e sistema operacional do computador onde é executado, elas também permitem comparar se houve uma melhora ou não de desempenho em executáveis compilados com diferentes opções de otimização. No entanto, não é possível estimar, a partir delas, em quais partes, subrotinas ou funções o programa dispendeu a maior parte do seu tempo de execução. Para contornar este problema, muitos programadores inserem em pontos estratégicos do programa chamadas a funções do sistema operacional que medem o tempo gasto em cada etapa, abaixo é ilustrado o relatório de execução (neste relatório a coluna "Name" reúne as subrotinas chamadas, a coluna "#calls" o número de chamadas e as outras colunas os tempos em segundos gastos em cada subrotina) gerado pelo programa *ab initio* Dalton[29] para a parte de otimização MCSCF utilizando esta técnica de medida de tempo:

```
(TIMOPT) Timing of MC optimization:
Name .      # calls .   total CPU .   CPU / call .   total wall .   wall / call.
GRAD            2         159.24         79.62          160.00         80.00
MAKDM                      90.87         45.43           91.00         45.50
FCKMAT                      0.11          0.05            1.00          0.50
GETH2                       0.13          0.07            0.00          0.00
ORDIAG                      0.00          0.00            0.00          0.00
ORBGRD                      0.00          0.00            0.00          0.00
CIGRAD                     68.13         34.07           68.00         34.00
```

Este relatório possui bem mais informações que os anteriores e permite determinar de forma segura o perfil de execução desta parte do programa no que diz respeito aos tempos de execução e ao número de vezes que cada função foi chamada. Em um processo de otimização, estas informações fornecem a granularidade adequada para se determinar qual o impacto de um dado conjunto de modificações do código ou de opções de otimização do compilador sobre cada uma das subrotinas. Entretanto, esta técnica mostra-se adequada somente na medida em que estas funções para medir o tempo de execução são inseridas ao longo da história de desenvolvimento do programa. Em termos práticos, é inviável alterar um programa de grandes dimensões e inserir manualmente nele dezenas ou centenas de chamadas com o único objetivo de otimizá-lo para uma dada combinação de hardware e sistema operacional. Esta técnica, também, não é capaz de medir adequadamente chamadas feitas às funções de bibliotecas externas ao programa como libm, BLAS ou LAPACK. Uma forma mais simples, eficiente e sem as limitações apontadas é fornecida pelo programa gprof que é descrito a seguir.

gprof**: UM PROGRAMA PARA GERAÇÃO E ANÁLISE DE PERFIS DE EXECUÇÃO**

O programa gprof foi desenvolvido em 1982 por McKusick[30,31] para o 4.2BSD UNIX, com objetivo de fornecer uma ferramenta de geração e análise de perfis de execução de programas e bibliotecas modulares do sistema. Isto é, no gprof o nível de detalhamento da execução alcança no máximo a sequência e o tempo de execução dos módulos (subrotinas, funções) que compõe o programa, não é possível gerar análises da execução de linhas de instruções como no tcov ou lprof[1]. Atualmente existem duas versões disponíveis publicamente: aquela que evoluiu a partir do trabalho original de McKusick e que encontra-se instalada em qualquer sistema operacional BSD, e uma outra versão que foi posteriormente desenvolvida pela GNU e faz parte do pacote de ferramentas binutils[32] disponível para um grande

número de sistemas operacionais. Para atingir os seus objetivos o gprof requer apenas que o programa seja recompilado indicando, com uma opção específica do compilador (-pg), que este insira no código objeto gerado de cada módulo uma chamada à função .mcount. Esta função se encarregará de coletar e acumular informações durante a execução do programa para a análise posterior. Em última análise, a filosofia de operação do gprof é semelhante àquela onde o programador insere manualmente chamadas em partes do programa que deseja fazer medidas de tempo, mas ela gera resultados mais completos e de forma automática sem exigir alterações no código fonte.

A partir das informações coletadas por .mcount durante a execução do programa, um sumário é gerado e salvo em um arquivo com sufixo .gmon. Este sumário é analisado pelo gprof e um relatório gerado onde constam:

- O perfil do grafo de chamadas de funções, isto é, quais funções chamaram outras funções;
- O perfil plano dos tempos de execução, isto é, quantas vezes cada função foi chamada, e o tempo associado a sua execução.

Estes perfis são descritos em detalhe nos dois exemplos a seguir, no primeiro, eles serão utilizados para realizar a otimização parcial de um programa *ab initio* visando acelerar um tipo específico de cálculo. No segundo exemplo eles serão empregados para realizar a otimização global de um programa de dinâmica molecular.

OTIMIZAÇÃO PARCIAL DE UM PROGRAMA *ab initio*

Abaixo é reproduzido parcialmente o perfil plano de execução do programa Dalton para um cálculo de intensidades Raman contendo as rotinas que consomem 51.0% do tempo total de execução

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
23.1	181.50	181.50	1452291	0.12	0.12	smpab_ [12]
7.4	239.66	58.16				ATL_daxpy [31]
7.0	294.24	54.58	2904576	0.02	0.02	ampatb_ [32]
5.6	338.50	44.27	8740065	0.01	0.01	dzero_ [34]
5.6	382.47	43.96	2775714	0.02	0.02	mxma_ [35]
4.8	420.37	37.90	20	1894.92	2908.13	tracd_ [30]

O perfil plano apresenta sete colunas. Na coluna “% time” são apresentados os percentuais do tempo total atribuídos a cada subrotina listada na coluna “name”. As colunas “self seconds” e “cumulative seconds” apresentam, respectivamente, o tempo total gasto na execução da rotina em questão e o tempo acumulado por ela e por aquelas acima dela na lista. A coluna “calls” indica o número total de chamadas a cada subrotina, enquanto a coluna “self ms/call” indica o tempo médio (em milisegundos) gasto em cada chamada da subrotina. A coluna “total ms/call” indica o tempo médio (em milisegundos) gasto na execução da subrotina e de subrotinas chamadas no decorrer de sua execução. Espaços em branco nestas três últimas colunas indicam somente que a biblioteca ou programa do qual esta subrotina faz parte não foi compilada com o suporte para “profiling” e, portanto, não é possível coletar estas informações.

Da análise deste perfil de execução constata-se que uma subrotina, smpab, domina o perfil de execução deste cálculo tomando 23,1 % do tempo total. Embora ela gaste em média apenas 0,12 milisegundos cada vez que é executada, ela é chamada 1452291 vezes ao longo do programa. Uma procura no grafo de chamadas do relatório gerado pelo gprof mostra que, neste cálculo, esta subrotina é utilizada apenas na subrotina fonemu:

		0.51	323.54	98464/98464	rspoli_ [10]
[11]	25.9	0.51	323.54	98464	fonemu_ [11]
		323.46	0.00	1451520/1451523	smpab_ [12]
		0.08	0.00	1451520/2903345	ampab_ [237]

A subrotina fonemu faz parte do conjunto de funções de cálculos de resposta linear do Dalton e, portanto, qualquer otimização que venha a ser feita em smpab não irá se refletir no desempenho do restante do programa. Um exame do programa fonte revela que a subrotina smpab realiza a operação matricial $C = C - A \cdot B$ e pode ser substituída por uma das funções otimizadas da biblioteca ATLAS[20] para reduzir o tempo de execução destas operações. Para otimizar esta parte da execução a smpab foi modificada para apenas chamar a subrotina dgemm da BLAS3 que faz parte da biblioteca ATLAS. O programa foi recompilado e o mesmo cálculo repetido. O tempo total de execução foi reduzido em 15% e uma análise do perfil de execução mostra que ele passa a ser dominado por apenas por ATL_daxpy,

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
10.5	88.66	88.66				ATL_daxpy [27]
7.6	152.47	63.81	2775714	0.02	0.02	mxma_ [32]
7.2	212.98	60.51	8272658	0.01	0.01	read [35]
6.9	271.38	58.40	2903040	0.02	0.02	ampatb_ [36]
5.9	320.91	49.53	20	2476.51	3460.92	tracd_ [30]
5.1	363.92	43.01	8740066	0.00	0.00	dzero_ [37]

A otimização completa deste programa exige a repetição destas sequências de análise dos perfis de execução, alterações no programa ou nas opções de compilação, reexecução e nova análise dos perfis. Nem sempre a simples substituição de uma rotina matricial por uma versão otimizada, como neste caso, implica em um ganho de desempenho. Dependendo das dimensões das matrizes de dados, as rotinas otimizadas podem mostrar uma velocidade inferior pois foram otimizadas para problemas de grandes dimensões.

OTIMIZAÇÃO DE UM PROGRAMA DE DINÂMICA MOLECULAR

O processo empregado para otimizar o programa mdsol[33] de dinâmica molecular será descrito a seguir. O programa mdsol realiza simulações de líquidos e soluções compostos por moléculas rígidas, não polarizáveis, com potenciais de Lennard-Jones e Coulomb sobre sítios de interação. Ele utiliza o algoritmo *leap frog* de integração com orientações moleculares formuladas em quaternions e o somatório de Ewald para forças coulômbicas de longo alcance[33,34]. Este programa foi utilizado por um dos autores (LRM) em uma das etapas do seu projeto de mestrado. Teste preliminares indicaram que essa etapa das simulações tomaria um intervalo de seis meses de processamento para ser realizado. Visando reduzir esta quantidade de tempo o perfil de execução do programa foi obtido com o gprof para um cálculo de teste, utilizando f2c e a otimização padrão (-O2) do compilador gcc. Este cálculo gastou 2791,1 segundos de tempo de execução e produziu o seguinte perfil de execução:

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
78.4	2186.95	2186.95	5000	437.39	437.39	forces_ [2]
20.6	2760.76	573.80	5000	114.76	114.76	furier_ [3]

Este perfil de execução mostra que o tempo de execução do programa é dominado por duas subrotinas apenas. Em uma primeira tentativa o programa foi recompilado com níveis mais altos de

otimização (-O3 -m486) e o cálculo repetido. O tempo de execução foi reduzido para 2774,1 segundos e apresentou o seguinte perfil plano de execução:

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
78.3	2171.18	2171.18	5000	434.24	434.24	forces_ [2]
20.6	2743.83	572.64	5000	114.53	114.53	furier_ [3]

Estes resultados mostraram que o compilador não consegue otimizar o código fonte gerado pelo f2c e acelerar significativamente a execução do programa. Um exame das subrotinas forces e furier mostrou a existência de várias sequências de instruções do tipo:

```
do i=1,n
  x(i)=x(i)*(a + b)
  y(i)=y(i)*(a + b)
  z(i)=z(i)*(a + b)
enddo
```

onde a sub-expressão constante[1] (A+B) não estava sendo eliminada pelo compilador mas recalculada a cada etapa (mais detalhes sobre a técnica de "*cse*" (common subexpression elimination") em compiladores e como ela pode falhar podem ser obtidos na referência número "1"). Estas partes de código foram modificadas manualmente e reescritas na forma:

```
do i=1,n
  xtemp=a+b
  x(i)=x(i)*xtemp
  y(i)=y(i)*xtemp
  z(i)=z(i)*xtemp
enddo
```

O programa foi recompilado e teve seu tempo de execução reduzido para 2124,3 segundos, um novo perfil de execução foi obtido e o relatório gerado mostra

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
76.6	1628.24	1628.24	5000	325.65	325.65	forces_ [2]
21.9	2093.66	465.42	5000	93.08	93.08	furier_ [3]

O maior impacto da otimização manual foi observado na rotina forces onde houve uma redução para 74,4% do tempo de execução, este resultado, em conjunto com a redução para 81,1% do tempo de execução de furier, resultou em uma razão, entre os tempos totais de execução antes e após a otimização manual, de 2124,33/2791,12 = 0,76 permitindo uma redução do prazo inicial de seis meses dessa etapa do projeto para cerca de quatro meses e meio. Em termos de velocidade do processador este resultado é equivalente a um *upgrade* do Pentium II 350 MHz utilizado no projeto para um processador idêntico mas com 460 MHz de velocidade de processamento.

CONSIDERAÇÕES FINAIS

Apresentamos aqui técnicas para a análise de desempenho particularmente úteis para programas de computação científica. A análise de desempenho deve ser rápida e sistemática, de modo a fornecer ao pesquisador as informações necessárias para a identificação de gargalos de execução e de falhas de programação. A utilização de ferramentas como o gprof na coleta de informações permite que estas duas condições sejam alcançadas e que seja possível passar à etapa de otimização com uma visão global do funcionamento do programa. A obtenção de programas o mais otimizados possível para um dado conjunto de tarefas e/ou condições de execução é o objetivo final do processo de análise de desempenho. Esta otimização, no entanto, nem sempre é óbvia e pode inclusive implicar a reformulação de partes do programa, na substituição de algumas subrotinas por outras mais rápidas ou na seleção do conjunto de opções do compilador que forneça o melhor desempenho para um dado conjunto de tarefas. Um estudo desta natureza é descrito por Gomes[35] onde uma extensa análise de desempenho e otimização da versão 1998 do programa GAMESS foi realizada. A importância da otimização não pode deixar de ser enfatizada. Como foi visto nos exemplos (onde observou-se uma redução do tempo total de execução da ordem de 30%) os ganhos de desempenho podem ser expressivos, e podem significar a transformação de um problema intratável em um tratável computacionalmente.

AGRADECIMENTOS

L. R. Martins e A. S. P. Gomes agradecem à FAPESP pelo apoio financeiro, P. A. M. Vazquez agradece aos Drs. F. Gozzo, N. H. Morgon e R. Custódio pelo acesso aos seus computadores para realização de algumas das medidas cujos resultados foram apresentados neste trabalho e ao apoio financeiro do FAEP/Unicamp (auxílio 0692/00) e NIC/Br.

REFERÊNCIAS

1. Dowd, K.; *High Performance Computing*; O'Reilly & Associates, Inc.: Sebastopol: USA, 2 ed., 1998.
2. Torvalds, L.; *GNU/Linux The Debian Distribution*; Debian Org: http://www.debian.org, 1995.
3. CSRG - UCB.; FreeBSD - Inc., *The FreeBSD Project*; FreeBSD Inc.: http://www.freebsd.org, 1992.
4. Ritchie, D. M.; Thompson, K. Em *4.4BSD Programmer's Supplementary Documents*; The USENIX Association and O'Reilly & Associates, Inc.: Sebastopol: USA, 1994, chap. 1.
5. Bach, M. J.; *The Design of the UNIX Operating System*, Prentice Hall: USA, 1987.
6. Salus, P. H.; *A Quarter Century of Unix*, Addison-Wesley: USA, 1994.
7. McKusick, M. K.; Bostic, K.; Karels, M. J.; Quaterman, J. S.*; The Design and Implementation of The 4.4BSD Operating System*, Addison-Wesley: USA, 1996.
8. Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S. J.; Windus, T. L.; Dupuis, M.; Montgomery, J. A.; *J. Comput. Chem.* **1993**, *14*, 1347.
9. Vazquez, P. A. M.; Morgon, N. H.; *Resumos da 16ª Reunião Anual da Sociedade Brasileira de Química*, Caxambu, Brasil, 1994.
10. Vazquez, P. A. M.; Freitas, L. C. G.; *Resumos da 19ª Reunião Anual da Sociedade Brasileira de Química*, Poços de Caldas, Brasil, 1996.
11. Schmidt, M. W.; *GAMESS User's Guide*, Department of Chemistry, Iowa State University, Ames: EUA, 1995.
12. Schmidt, M. W. Em *GAMESS User's Guide*, Department of Chemistry, Iowa State University: http://www.msg.ameslab.gov/FTP/GAMESS_Manual/TEXT/IRON.DOC, 2001, chap. 6.
13. Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Gill, P. M. W.; Johnson, B. G.; Robb, M. A.; Cheeseman, J. R.; Keith, T.; Petersson, G. A.; Montgomery, J. A.; Raghavachari, K.; Al-Laham, M. A.; Zakrzewski, V. G.; Ortiz, J. V.; Foresman, J. B.; Cioslowski, J.; Stefanov, B. B.; Nanayakkara, A.; Challacombe, M.; Peng, C. Y.; Ayala, P. Y.; Chen, W.; Wong, M. W.; Andres, J. L.; Replogle, E. S.; Gomperts, R.; Martin, R. L.; Fox, D. J.; Binkley, J. S.; Defrees, D. J.; Baker, J.; Stewart, J. P.; Head-Gordon, M.; Gonzalez, C.; Pople, J. A.; *Gaussian 94, Revision D.2*; Gaussian, Inc., Pittsburgh: PA, 1995.
14. Feldman, S. I.; Weinberger, P. J.; Berkman, J. Em *4.4BSD Programmer's Supplementary Documents*, The USENIX Association and O'Reilly & Associates Inc., Sebastopol: USA, 1994, chap. 8.
15. Burley, C.; *The GNU Project FORTRAN77 Compiler*, Free Software Foundation: http://gcc.gnu.org/onlinedocs/g77_news.html , Cambridge: EUA, 1992.

16. Feldman, S. I.; Gay, D. M.; Maimone, M. W.; Schryer, N. L.*; A FORTRAN-to-C Converter*, Computing Science Technical Report No. 149 AT&T Bell Laboratories, Murray Hill: USA, 07974, 1990.
17. Stalman, R. M.; *The GNU project C and C++ Compiler*, Free Software Foundation: http://gcc.gnu.org, Cambridge: EUA, 1991-99.
18. Henry, G.; *ASCI Red Pentium II BLAS 1.2F*, ASCI Red Project Archives: http://www.cs.utk.edu/~ghenry/distrib/, 1997.
19. PGI, *The Portland Group Fortran Compiler*, Portland Group Inc.: http://www.pgroup.com/, 1998.
20. Dongarra, J.; *ATLAS Automatically Tuned Linear Algebra Software*, University of Tenesee Kentucky: http://www.netlib.org/atlas, 1999.
21. Bilmes, J. A.; Asanovic, K.; Vudoc, R.; Iyer, S.; Demmel, J.; Chin, C.; Lam, D.; *The PHiPAC, Portable High Performance ANSI C for BLAS3 Compatible Fast Matrix Matrix Multiply*, UCB: http://www.icsi.berkeley.edu/~bilmes/phipac/, 1997.
22. Frigo, M.; Johnson, S. G.; *FFTW: An Adaptive Software Architecture for the FFT*; vol. 3 ICASSP Proceedings: http://www.fftw.org, 1998.
23. Morgon, N. H.; Gozzo, F. C.; Vazquez, P. A. M.; trabalho não publicado, morgon@canario.iqm.unicamp.br, 2000.
24. Morgon, N. H.; *Resumos da 24ª Reunião Anual da Sociedade Brasileira de Química*, Poços de Caldas, Brasil, 2001.
25. Loukides, M.; *Unix for FORTRAN Programmers*, O'Reilly & Associates, Inc., Sebastopol: USA, 1990.
26. Goodheart, B.; Cox, J.; *The Magic Garden Explained: The Internals of Unix System V Release 4. An Open-Systems Design*, Prentice-Hall: USA, 1994.
27. Aburto, A.; *flops.c Version 2.0*, NOSC: ftp://ftp.nosc.mil/pub/aburto, 1992.
28. Vazquez, P. A. M.; *Quim. Nova* **2002**, *25*, 117.
29. Helgaker, T.; Jensen, H. J. A.; Jorgensen, P.; Olsen, J.; Ruud, K.; Agren, H.; Andersen, T.; Bak, K. L.; Bakken, V.; Christiansen, O.; Dahle, P.; Dalskov, E. K.; Enevoldsen, T.; Heiberg, H.; Hettema, H.; Jonsson, D.; Kirpekar, S.; Kobayashi, R.; Koch, H.; Mikkelsen, K. V.; Norman, P.; Packer, M. J.; Saue, T.; Taylor, P. R.; Vahtras, O.; *Dalton, an ab initio electronic structure program, Release 1.0*, University of Oslo, Norway: http://www.kjemi.uio.no/software/dalton/dalton. html, 1997.
30. Graham, S. L.; Kessler, P. B.; McKusick, M. K.; *Software Pract. Exper.* **1983**, *13*, 671.
31. Graham, S. L.; Kessler, P. B.; McKusick, M. K. Em *4.4BSD Programmer's Supplementary Documents*; The USENIX Association and O'Reilly & Associates, Inc., Sebastopol: USA, 1994, chap. 18.
32. GNU-Project, *gprof: The GNU execution profiler*; Free Software Foundation: http://gcc.gnu.org/software/binutils/binutils.html, Cambridge: EUA, 1995.
33. Martins, L. R.; *Dissertação de Mestrado*, Universidade Estadual de Campinas, Brasil, 2000 (http://biq.iqm.unicamp.br/arquivos/teses/teses online.html).
34. Skaf, M. S.; comunicação pessoal.
35. Gomes, A. S. P.; Braga, A. A. C.; Vazquez, P. A. M.*; X Simpósio Brasileiro de Química Teórica*, Caxambu, Brasil, 1999, http://minmei.iqm.unicamp.br/gamess98/.
36. USENIX; O'Reilly, *4.4BSD Programmer's Supplementary Documents*, The USENIX Association and O'Reilly & Associates, Inc., Sebastopol: USA, 1994.