ÂMBAR: Um Ambiente de Paralelização de Programas

Bruno Müller Jr.*
Eduardo Voigt†
Fábio Carneiro Mokarzel‡
Jairo Panetta§
Walter Luiz Caram Saliba¶

Instituto de Estudos Avançados (IEAv-CTA)
Rodovia dos Tamoios Km 5,5 Caixa Postal 6044
CEP 12231 São José dos Campos - SP
Tel: (0123) 413033 Ramais: 367 e 348
e-mail: CTA@BRFAPESP.BITNET

RESUMO

Este artigo descreve o ÂMBAR, um ambiente de paralelização de programas que expõe o processo de compilação ao usuário, permitindo sua intervenção direta em cada fase da compilação. O objetivo deste ambiente é auxiliar a paralelização de programas de origem seqüencial, tanto para usuários leigos em processamento paralelo quanto para os experientes.

ABSTRACT

This article describes ÂMBAR, a parallelization programming environment that exposes the compilation process to the user, allowing direct intervention at each compilation phase. The central goal of this environment is to assist the parallelization of sequentially originated software, for a wide range of users, from naive to expert in parallel processing.

*MsC, UNICAMP.
†MsC, UNICAMP.
‡MsC, ITA.
§PhD, Purdue University.
¶MsC, ITA.

1 Introdução

Processamento paralelo é um recurso largamente explorado por sistemas computacionais voltados para alto desempenho. Quer implícito em instruções vetoriais e no esquema de execução de máquinas RISC, quer explícito em máquinas MIMD ou SIMD, processamento paralelo amplifica, embora em graus distintos, a dificuldade de programação inerente a máquinas seqüenciais.

Este fato é particularmente agravado quando há necessidade de extrair alto desempenho de programas originalmente seqüenciais. Há fortes indícios que a quantidade de paralelismo existente nesses programas é muito superior à que se consegue extrair na prática [24]. Mesmo assim, extrair este baixo grau de paralelismo requer grandes esforços, obrigando o usuário típico a se contentar com desempenhos ainda mais modestos.

Este trabalho descreve o ÂMBAR, um ambiente de paralelização de programas que auxilia usuários a extrair alto desempenho de máquinas paralelas. Voltado para programas escritos em FORTRAN 77 ANSI [2], o ambiente apresenta ferramentas que possibilitam a paralelização automática, acelerando a obtenção e a avaliação de uma primeira versão paralela. A obtenção de versões progressivamente mais eficientes é auxiliada por ferramentas manuais de paralelização, que permitem a interferência direta do usuário nas diversas fases da compilação. Esta estrutura atinge tanto o usuário iniciante em paralelismo quanto o experiente.

O processo de extração de desempenho é descrito, detalhadamente, nas seções 2 e 3. Em seguida, apresenta-se a estrutura proposta para o ÂMBAR, a descrição de seus módulos componentes e o estágio atual de implementação. Uma breve crítica da proposta encerra este trabalho.

2 A Extração de Desempenho - O Papel do Compilador

A forma atual de extração de desempenho de um programa baseia-se em um processo de tentativa e erro, com direções determinadas pela literatura técnica e pela experiência prévia. Tipicamente, visualiza-se uma direção de paralelização, altera-se correspondentemente o programa e mede-se experimentalmente o ganho obtido. A análise dos resultados gera novas direções, realimentando o processo. Este ciclo é interrompido ao atingir-se o resultado desejado ou esgotarem-se os recursos disponíveis.

Quando a extração de desempenho é aplicada à execução de um programa seqüencial em um processador paralelo, grande esforço é despreendido na interação com o compilador. Por exemplo, em laços com iterações sabidamente independentes que não são paralelizados por detalhes de programação. Ou então em repetidas invocações a um mesmo procedimento, sobre dados independentes, porém agrupados em uma única estrutura de dados.

As mensagens emitidas pelo compilador, em casos similares a estes, pouco auxiliam o usuário, por serem genéricas. Por exemplo, "recorrência envolvendo as variáveis X Y e Z impede a paralelização". Nestes casos, o programa fonte é usualmente distorcido por um novo processo de tentativa e erro até a remoção do empecilho. Este novo ciclo, interno ao processo de extração de alto desempenho, é indesejável a tal processo e estafante por si só.

Situação semelhante ocorre quando o compilador escolhe, automaticamente, qual laço paralelizar em um aninhamento de laços sabidamente paralelos. Muitas vezes a escolha não é a desejada. Nesses casos, o usuário é obrigado a alterar o ordem dos laços, no programa fonte,

até obter o resultado desejado. Abre-se, novamente, um ciclo interno ao processo de extração de desempenho.

Este "ciclo dentro de um ciclo" pode ser evitado pela ação direta do usuário sobre o compilador e não sobre o programa fonte. As "diretivas de compilação" dos compiladores atuais são um exemplo deste mecanismo. Trata-se, entretanto, de mecanismo inadequado de comunicação, pela própria linguagem restrita que pode empregar.

O ÂMBAR propõe expor o processo de compilação ao usuário, permitindo sua intervenção direta em cada passo da compilação. Para tanto, o compilador paralelizador é particionado na forma abaixo:

1. Análise léxica, sintática e semântica;
2. Análise de dependências;
3. Detecção de paralelismo;
4. Exploração de paralelismo;
5. Geração de código.

O papel de cada fase, bem como o grau de interação com o usuário, são discutidos a seguir. Conclusões que definem a forma do ÂMBAR são apresentadas.

2.1 Análise Léxica, Sintática e Semântica

Esta fase é similar à existente em compiladores tradicionais, exceto que dados usualmente descartados devem ser mantidos. Na compilação tradicional, esta fase produz uma forma intermediária do programa que armazena apenas as informações necessárias para a geração de código. Em compiladores otimizadores, algumas destas informações são mantidas para a análise de dependências. Já compiladores paralelizadores requerem ainda mais informações sintáticas e semânticas, necessárias para a detecção e a exploração de paralelismo. Conseqüentemente, a forma intermediária do programa é mais rica neste tipo de informação que nos casos anteriores. No ÂMBAR, esta estrutura é denominada *grafo do programa* e sua especificação encontra-se na referência [29].

Esta fase comunica-se com o usuário pelo relato convencional de erros léxicos, sintáticos e semânticos. Não há comunicação reversa.

2.2 Análise de Dependências

Dois comandos podem ser executados simultaneamente quando forem independentes entre si, ou seja, quando a execução de um deles não está subordinada à execução do outro. As diversas formas de dependência entre comandos estão fartamente documentadas na literatura técnica [4, 3, 14]. Os resultados desta análise são armazenados em um *grafo de dependências* do programa, com vértices representando comandos e arestas representando as relações de dependência entre eles.

Esta fase é fundamental para o sucesso do paralelizador. Infelizmente, é indecidível determinar, durante a compilação, se dois comandos são independentes. Por exemplo, no fragmento abaixo,

```
      DO 10 i = 1, n
         read *, b(i), c(i)
         a(b(i)) = a(c(i))
10    CONTINUE
```

não se conhece, durante a compilação, quais elementos de a serão alterados durante a execução. Em casos similares, o algoritmo de análise considera dependentes todas as instâncias da atribuição, sequencializando a execução do laço.

A existência de fortes mecanismos de comunicação entre o usuário e esta fase é essencial. Ele deve poder observar e alterar as dependências geradas, eliminando o conservadorismo imposto pelas deficiências da análise tradicional.

2.3 Detecção de Paralelismo

Detecção de paralelismo é a tarefa de aplicar ao programa uma série de técnicas para determinar e explicitar paralelismo escondido pela sua formulação seqüencial. Para tanto, o texto do programa fonte é alterado por métodos específicos, como por exemplo, a expansão de variáveis escalares. Múltiplos outros métodos são descritos na literatura especializada [14, 32, 19, 23, 34, 25].

Tais alterações visam produzir novas versões do programa original, quer com menor número de dependências, quer com dependências situadas em posições mais favoráveis. Entretanto, há múltiplas técnicas, algumas conflitantes, que podem ser aplicadas em um trecho de programa.

A ordem de aplicação das técnicas é, no caso geral, um problema em aberto. Paralelizadores automáticos implementam heurísticas de resultados duvidosos e não há, na literatura técnica, estudos que indiquem vantagens de uma heurística sobre outras. Por isso, é desejável que o usuário possa dirigir a aplicação das técnicas desta fase.

2.4 Exploração de Paralelismo

Nem todo o paralelismo detectado na fase anterior pode ser aproveitado pelo computador alvo. Este certamente apresenta limitações ao processamento paralelo, como por exemplo, no número de processadores, no tamanho de registradores vetoriais, na velocidade e simultaneidade nos acessos à memória, etc. Conseqüentemente, a máquina alvo não aproveita todo o paralelismo detectado.

Tal critério é, novamente, heurístico, o que conduz a insatisfações. Por exemplo, na escolha de um dentre múltiplos laços de um aninhamento a ser paralelizado, na alocação das variáveis na hierarquia de memória e no mapeamento das iterações de laços e das tarefas de grande porte nos processadores [34, 25]. Novamente, faz-se necessário forte interação entre o usuário e a fase.

2.5 Geração de Código

Esta fase inclui a otimização e a geração de código comuns a compiladores otimizadores. Em particular, é perfeitamente possível que ganhos obtidos pelas fases anteriores sejam eliminados

pelas otimizações desta fase. Por exemplo, a interação entre métodos de detecção de paralelismo e os métodos de reordenação de código para máquinas RISC é virtualmente desconhecida. A necessidade de intervenção do usuário ainda é motivo de pesquisa.

2.6 Conclusões

É inegável que usuários experientes serão beneficiados por mecanismos específicos de comunicação com diversas fases do paralelizador. Por outro lado, é irreal imaginar que usuários inexperientes conheçam as diversas técnicas existentes (por exemplo, "frente de onda" e "auto-escalonamento"). Conseqüentemente, o ambiente deve prover facilidades tanto para ser orientado quanto para orientar o usuário.

Estas conclusões indicam a necessidade da confecção de grande volume de software, em excesso ao requerido por um compilador paralelizador. Não é assim. O núcleo de um paralelizador pode ser entendido como um conjunto de heurísticas que governa a aplicação de um elenco de métodos. Logo, cada um dos métodos deve estar implementado; basta torná-los disponíveis, individualmente, ao usuário.

O ÂMBAR propicia estes dois acessos. As fases centrais são utilizadas por meio de ferramentas automáticas ou manuais. As manuais permitem a escolha, de forma interativa, de uma técnica e, em alguns casos, de um trecho do programa para aplicá-la. As automáticas são heurísticas que fornecem a ordem de aplicação das técnicas.

3 A Extração de Desempenho - Demais Características

Abstraídas as dificuldades de compilação, o processo de extração de desempenho ainda requer a capacidade de alterar programas, verificar resultados, medir tempos de execução e, caso tais tempos sejam insatisfatórios, determinar se há características de arquitetura mal aproveitadas.

Logo, um bom compilador não é suficiente. Necessita-se, no mínimo, de ferramentas para a edição e a avaliação de desempenho. As ferramentas de edição parecem, a princípio, não serem afetadas pelo processamento paralelo. Entretanto, há melhorias de desempenho advindas de alterações não representáveis no texto do programa (por exemplo, eliminação manual de dependências). Logo, se um programa assim tratado for submetido a um editor de texto usual, será difícil evitar a perda de trabalho anterior.

A solução proposta no ÂMBAR é um editor de texto que considera e atualiza o grafo do programa, conforme o texto seja alterado. Para tanto, o editor faz análises léxica e sintática em conjunto com as tarefas usuais de edição. Mantém-se, portanto, a correspondência entre as duas representações do programa (texto e grafo), possibilitando a convivência de informações não conflitantes oriundas de qualquer uma das duas fontes.

Uma outra ferramenta primordial é o analisador de desempenho. O sistema descrito até este ponto orienta o processo de compilação na direção de um determinado código de máquina, mas não garante que tal código será executado eficientemente e nem indica as origens de eventuais ineficiências.

Determinar os motivos de um fraco desempenho envolve ainda mais variáveis que a paralelização automática. Pode ser originado por um algoritmo computacionalmente caro ou ainda

inadequado à máquina disponível. Neste último caso, detalhes da máquina podem ser preponderantes (o mecanismo de tradução de endereços virtuais para reais, por exemplo). Uma ferramenta que considere computadores a este nível não possui, internacionalmente, formato e características definidas, embora existam propostas interessantes [12]. Um destes esboços, proposto para o ÂMBAR, encontra-se em [21].

4 Estrutura e Componentes do ÂMBAR

A figura 1 esquematiza a estrutura geral do ÂMBAR. As figuras 2 e 3 detalham os "middle-end" interativos e automáticos. Descreve-se a seguir diversos de seus componentes, o mecanismo de comunicação com o usuário e o estágio atual de implementação.

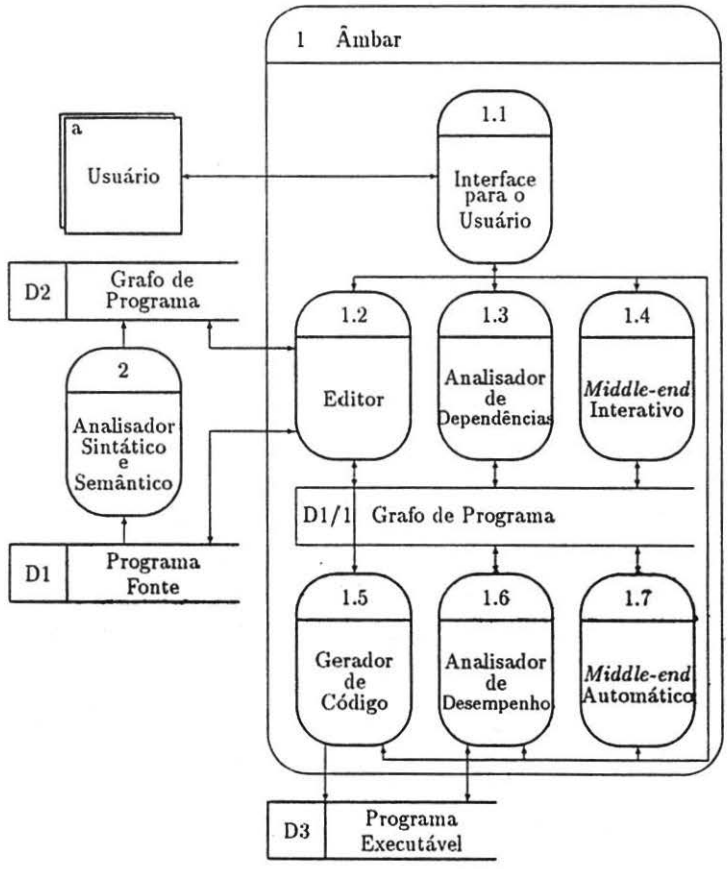

Figura 1: Estrutura interna do Âmbar.

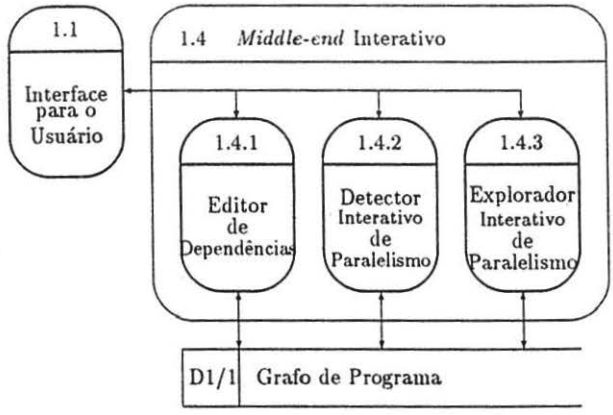

Figura 2: Detalhamento do *middle-end* interativo.

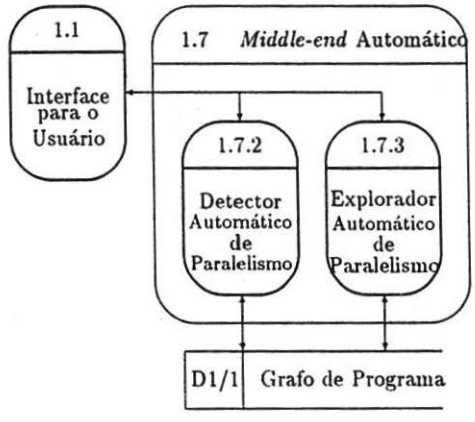

Figura 3: Detalhamento do *middle-end* automático.

4.1 Interface Gráfica

A interface gráfica do ÂMBAR possui dois objetivos:

- Padronizar a interação do usuário com as ferramentas do ambiente;
- Simplificar a confecção da interface das ferramentas, pela utilização de primitivas gráficas padronizadas.

A interface provê mecanismos que simplificam a construção de janelas, menus, *scrollbars*, além de facilitar a interação via texto e permitir a abstração de programas.

Este último item é uma característica marcante da interface. Ele permite ao usuário o exame de uma representação de seu programa sob a forma de grafos dirigidos. Numa das possíveis formas, os vértices representam os comandos do programa enquanto que as arestas representam o fluxo de execução entre eles. Além destas duas entidades, também é possível representar as dependências, de tal forma que o usuário pode indicar quais aquelas que devem ser mantidas e quais podem ser omitidas, possivelmente melhorando o desempenho do programa.

Uma versão inicial desta ferramenta encontra-se implementada. Sua descrição detalhada está em [21].

4.2 Grafo de Programa

As ferramentas componentes do ÂMBAR atuam, preferencialmente, sobre o grafo do programa. Na sua forma mais restrita, o grafo sintetiza as informações léxicas, sintáticas e semânticas necessárias para reproduzir o texto original do programa (exceto por brancos, parênteses em excesso e comentários). A especificação formal dessa forma restrita do grafo encontra-se em [29].

O Analisador de Dependências acrescenta, ao grafo do programa, o grafo de dependências. O Detector de Paralelismo modifica o grafo quer pela anotação de vértices que encabeçam estruturas paralelas (por exemplo, na transformação de nós DO em nós DOALL), quer pelas alterações feitas por seus métodos de paralelização. O Explorador de Paralelismo modifica novamente o grafo, anotando quais laços serão executados em paralelo, posicionando variáveis na hierarquia de memória, etc.

Percebe-se, portanto, que o grafo do programa precisa estar preparado para representar todos os comandos FORTRAN e também as formas de paralelismo, o escalonamento de trechos de programa a processadores e a alocação de variáveis, dentre outros. Isto é efetuado por modificações nos nós do grafo denominadas *anotações paralelas*.

A transformação do texto do programa no grafo pode ser feita por duas ferramentas: o Analisador Sintático e Semântico (vide seção 4.4) e o Editor (vide seção 4.3). O grafo criado por estas ferramentas encontra-se despojado de informações sobre dependências e paralelismo.

4.3 Editor

O Editor é uma via de entrada de programas no ÂMBAR. Trata-se de um editor de textos comum (MicroEMACS) ao qual foi acoplado o *front-end* de um compilador incremental para FORTRAN 77. Portanto, além da capacidade de geração incremental do grafo correspondente

aos programas editados, o editor também é capaz de detectar erros léxicos, sintáticos e, até o momento, alguns erros semânticos.

A existência do Editor permite alterações no texto do programa internamente ao ÂMBAR. Conforme menção anterior, esta funcionalidade é fundamental para manter informações adicionadas por outras ferramentas, no caso de re-edição. Na implementação atual [28], tais informações são descartadas.

4.4 Analisador Sintático e Semântico Automático

O Analisador Sintático e Semântico é a segunda forma de entrada de programas no ÂMBAR: ele converte o texto do programa na forma mais despojada do grafo. Trata-se, portanto do *front-end* de um compilador para FORTRAN 77 (analisadores léxico, sintático e semântico), ao qual foram acrescidos procedimentos para criar o grafo do programa.

O analisador automático compartilha larga porção de código com o *front-end* incremental. Este é o primeiro exemplo implementado de uma das características centrais do ÂMBAR: ferramentas automáticas construidas sobre o código das ferramentas manuais.

4.5 Analisador de Dependências

O analisador de dependências implementa os métodos tradicionais de análise de fluxo de dados [1, 9] para a determinação das dependências entre variáveis escalares, além de métodos específicos para variáveis indexadas. Projeta-se claramente a necessidade de métodos de análise interprocedurais para a detecção de dependências.

Dentre os testes para variáveis indexadas, estão incluidos o "constante", "mdc", "raiz real", "exato", "tudo igual" e "lambda", além do cálculo dos vetores de direção [4, 3, 34, 18, 16].

Estão em fase de implementação a análise de fluxo de dados e os testes "constante", "mdc", "exato" e o cálculo dos vetores de direção .

Não se prevê, no momento, a interferência do usuário na seqüencia de aplicação das técnicas. O Analisador de Desempenho impõe uma seqüencia única. O usuário, entretanto, terá a seu dispor o Editor de Dependências (bloco 1.4.1 da figura 2), que possibilita a eliminação interativa de dependências indevidas. Os resultados são visíveis no grafo do programa.

4.6 Detector de Paralelismo

O Detector de Paralelismo do ÂMBAR é composto pelo seguinte elenco de métodos: eliminação de blocos desestruturados [33, 30], uso de novos identificadores para variáveis escalares [14, 20], expansão escalar unidimensional [14, 20], reordenação de laços [34, 10], reordenação de comandos [19, 25], fissão de comandos [14, 23], fusão de laços [34], retrocesso de ciclos [25], paralelização total e parcial de laços DO [34, 32], frente de onda [22, 15], paralelização de laços WHILE [34, 18] e linearização de laços [25]. Estão em fase de implantação os quatro primeiros métodos.

Há duas formas de comunicação entre o usuário e o Detector de Paralelismo: a automática e a interativa. O Detector Automático de Paralelismo (bloco 1.7.2 da figura 3) impõe uma seqüencia fixa de aplicação das técnicas de paralelismo, enquanto o Detector Interativo (bloco 1.4.2 da figura

2) permite ao usuário a escolha do método e do trecho do programa que sofrerá a alteração. Os resultados são visíveis no grafo do programa.

4.7 Explorador de Paralelismo

As fases de exploração e detecção de paralelismo são usualmente reunidas em um contexto comum de reestruturação. Com o objetivo de aumentar a flexibilidade do usuário na transformação do programa, o ÂMBAR separa as atividades de detecção das de exploração. Esta última estará munida de um conjunto de técnicas, tais como:

- Escolher, dentre laços paralelos aninhados, qual será executado em paralelo;
- Escalonar[1] as iterações de laços mais internos, envolvendo técnicas de alocação estática (pré-escalonamento) e alocação dinâmica (auto-escalonamento, escalonamento por blocos e escalonamento auto-guiado[2]) [25];
- Alocar os dados na hierarquia de memória, diferenciando dados globais de locais e permitindo um maior controle na utilização de *caches*.

Esta ferramenta não está sendo implementada, no momento.

Novamente, há duas formas de interação com o usuário. O Explorador Automático de Paralelismo (bloco 1.7.3 da figura 3) impõe uma ordem fixa de aplicação das técnicas, enquanto o Explorador Interativo (bloco 1.4.3 da figura 2) permite que o usuário escolha a técnica a ser aplicada e o trecho do programa que sofrerá alteração.

4.8 Gerador de Código

A geração de código ocorrerá em duas fases. Para gerar código seqüencial, necessário a cada processador, será utilizado o *back-end* do *GNU C Compiler* (GCC) [31] da Free Software Foundation. Ele será acoplado ao restante do sistema por meio de um filtro que transforma trechos do grafo do programa na linguagem intermediária (RTL) utilizada pelo GCC. Observe-se que o *back-end* do GCC gera código para diversos processadores, como o MIPS R3000, IBM RS6000, Intel i860, dentre outros.

A segunda fase compreeende a geração de código paralelo. Esta fase será implantada através de um *conversor de paralelismo*[17], que atuará como um pré-processador ao filtro acima descrito. O conversor transformará as anotações paralelas existentes no grafo em código envolvendo as primitivas de sincronização e rotinas de divisão de trabalho entre processadores.

Atualmente, apenas o filtro está em fase de implementação.

5 Discussão

A obtenção automática de paralelismo a partir de programas seqüenciais é uma realidade industrial. Todos os fabricantes de supercomputadores tradicionais ofertam compiladores FORTRAN com capacidade automática de paralelização. Entretanto, o desempenho obtido por este

[1] schedule

[2] guided-self scheduling

método encontra-se muito aquém do desejado. Ressalte-se, novamente, a existência de fortes indícios que programas seqüenciais apresentam paralelismo intrínseco em nível muito superior ao obtido automaticamente [24].

Por outro lado, programas seqüenciais implementam algoritmos seqüenciais. Consequentemente, a efetividade da paralelização de tais programas parece estar limitada a máqinas com poucos processadores, nas quais um ambiente como o aqui exposto pode ser bastante efetivo. Em máquinas maciçamente paralelas, ou a aplicação possui paralelismo trivial, ou será necessário utilizar algoritmos paralelos. Aplicações no primeiro caso são beneficiadas por ambientes como o ÂMBAR. Aplicações no segundo caso deverão ser, obviamente, re-escritas.

O ÂMBAR está sendo projetado para máquinas com memória central e número modesto de processadores. Isto é refletido na escolha das técnicas para a detecção e exploração de paralelismo. Entretanto, nada impede a inserção de técnicas mais adequadas à memória distribuida e ao processamento paralelo maciço [13].

A pesquisa internacional na área retrata a atualidade do tema [8, 27, 35]. Há múltiplas propostas de linguagens inovativas para expressar paralelismo; há até evidências que linguagens funcionais podem apresentar desempenho compatível com o atingido por programas FORTRAN em máquinas paralelas [7]. Entretanto, não há um claro vencedor. Enquanto isto, a comunidade científica continua programando em linguagens seqüenciais, alvo de ambientes como o ÂMBAR.

Como oportunidade de pesquisa, o ÂMBAR possui flexibilidade incomum. Há campo para diversos temas clássicos de processamento paralelo, como a alocação de processadores, o controle da hierarquia de memória e a representação de paralelismo. Permite, obviamente, a pesquisa de novas técnicas para a paralelização de programas, bem como o confronto de heurísticas e, finalmente, nivela-se a projetos de pesquisa semelhantes no exterior [5, 6, 11, 26].

Referências

[1] Aho, A. V. e Sethi, R. e Ullman, J. D. - *Compilers - Principles, Techniques and Tools*, Addison-Wesley, 1986.

[2] American National Standards Institute - *American National Standard Programming Language FORTRAN*, X3.9 - 1978.

[3] Banerjee, U.- *Speedup of Ordinary Programs*, Ph.D. Thesis, University of Illinois, outubro de 1979.

[4] Banerjee, U.- *Dependence Analysis for Supercomputing*, Kluwer Academic Publishers, 1988.

[5] Callahan, D. et alli - *Parallel Programming Support in ParaScope*, Computer Science Technical Report COMP TR87-59, Rice University, julho de 1987.

[6] Callahan, D. et alli - *ParaScope: A Parallel Programming Environment*, Computer Science Technical Report COMP TR88-77, Rice University, Setembro de 1988.

[7] Cann, D. - *Retire FORTRAN? A Debate Rekindled*, Communications of the ACM, Vol. 35, No. 8, agosto 1992, pg 81–89.

[8] Chase, C., Cheung, A., Reeves, A. e Smith, M. - *Paragon: A parallel programming environment for scientific applications using communication structures*, Proceedings of the 1991 International Conference on Parallel Processing, agosto 1991, IEEE.

[9] Cunha, P. A. G. - *Testes de Dependências em Tempo de Compilação*, Trabalho de Graduação em andamento, ITA, 1992.

[10] Faria, R. A. B.- *Técnicas de Tratamento de Ciclos de Dependências*, Trabalho de Iniciação Científica em andamento, ITA, 1992.

[11] Gannon, D. et alli - *A Software Tool for Building Supercomputer Applications*, Technical Report 224, Indiana University, agosto de 1987.

[12] Gannon, D. et alli - *Programming Environments for Parallel Computation: Performance Debugging of Parallel Algorithms*, CSRD Technical Report, 1988.

[13] Hiranandani, S., Kennedy, K., e Tseng, C.W. - *Compiling FORTRAN D for MIMD Distributed-Memory Machines* Communications of the ACM, Vol. 35, No. 8, agosto 1992, pg 66-80.

[14] Kuck, D. J. et alli - *Dependence Graphs and Compiler Optimizations*, Proceedings of the 8th ACM Symposium on Principles of Programming Languages, pg. 207-218, Williamsburg, VA, janeiro de 1981.

[15] Lamport, L.- *The Parallel Execution of DO loops*, Communications of the ACM, 17(2), pg 83-93, fevereiro de 1974.

[16] Li, Z. e Yew, P. C. e Zhu, C. Q. - *An Efficient Data Dependence Analysis for Parallelizing Compilers*, IEEE Transactions on Parallel and Distributed Systems, 1(1), pg 26-34, janeiro de 1990.

[17] Maciel, F. B.- *Um Ambiente para Reestruturação e Compilação de Programas para Máquinas Paralelas*, Dissertação de Mestrado, ITA, 1990.

[18] Mokarzel, F. C. *Compilador de Programas Seqüencias para Multiprocessamento: Análise e Metodologia para sua Implementação* , Dissertação de Mestrado, ITA, 1984.

[19] Mokarzel, F. M. e Panetta, J.- *Reestruturação Automática de Programas Seqênciais para Processamento Paralelo*, II Simpósio Brasileiro de Arquitetura de Computadores - Processamento Paralelo, Águas de Lindóia, SP, Setembro de 1988.

[20] Morizawa, R. K.- *Técnicas Preparatórias de Paralelização de Laços FORTRAN*, Trabalho de Graduação em andamento, ITA, 1992.

[21] Müller Jr., B.- *Uma Interface de Comunicação para um Ambiente de Reestruturação de Programas*, Dissertação de Mestrado, UNICAMP, 1991.

[22] Muraoka, Y.- *Parallelism Exposure and Exploitataion in Programs*, Ph.D. Thesis, University of Illinois, fevereiro de 1971.

[23] Padua, D. A. e Wolfe, M. J.- *Advanced Compiler Optimizations for Supercomputers*, Communications of the ACM, 29(12), pg. 1184-1201, dezembro de 1986.

[24] Polychronopoulos, C. D. e Banerjee, U. - *Allocation for Horizontal and Vertical Parallelism and Related Speedup Bounds"*, IEEE Transaction on Computers, Vol C-36, No 4, abril 1987.

[25] Polychronopoulos, C. D.- *Parallel Programming and Compilers*, Kluwer Academic Press, 1988.

[26] Polychronopoulos, C. D. et alli - *Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing and Scheduling Programs on Multiprocessors*, International Journal of High Speed Computing, Vol 1, No 1, pg. 45-72, 1989.

[27] Pugh, W. - *A Practical Algorithm for Exact Array Dependence Analysys*, Communications of the ACM, Vol. 35, No. 8, agosto 1992, pg 102–115.

[28] Saliba, W. L. C.- *Um Editor Orientado a FORTRAN 77*, Dissertação de Mestrado, ITA, 1992.

[29] Saliba, W. L. C. et alli - *Documento de Especificação da Forma Interna de Programas FORTRAN 77 no Ambiente de Reestruturação do Projeto Computação Científica*, Publicação Interna do IEAv, fevereiro de 1992.

[30] Spadinger, B. A.- *Linearização Automática de Laços FORTRAN*, Trabalho de Graduação em andamento, ITA, 1992.

[31] Stallman, Richard M. - *Using and Porting GNU CC*, Free Software Foundation, 1992.

[32] Voigt, E.- *Paralelismo e Sincronização em Laços*, Dissertação de Mestrado, UNICAMP, 1991.

[33] Williams, M. H. e Ossher, H. L.- *Conversion of Unstructured Flow Diagrams to Structured*, The Computer Journal, 21(2), pg. 161-167, fevereiro de 1978.

[34] Wolfe, M. J.- *Optimizing Supercompilers for Supercomputers*, MIT Press, 1989.

[35] Zima, H., Bast, J. e Gerndt, M. - *SUPERB: A tool for semi-automatic MIMD/SIMD parallelization*, Parallel Computing Vol. 6, 1989.