Um Esquema Produtor-Consumidor para Resolução Paralela do Problema de Fluxo de Potência Ótimo com Restrições de Segurança

Alcir J. Monticelli, Marcelo Rodrigues, Osvaldo R. Saavedra
DSEE-FEE-UNICAMP
CAMPINAS-SP-BRASIL
CP 6101 -CEP 13081

1.0 Introdução

Os avanços recentes na engenharia de hardware têm levado a novos dispositivos e arquiteturas enquanto que a engenharia de software defronta-se com novos paradigmas de desenvolvimento buscando extrair ao máximo esta capacidade do hardware. Assim o desenvolvimento de aplicações tende a ficar cada vez mais especializado para uma determinada arquitetura de hardware dificultando a migração das aplicações de uma arquitetura para outra. A proposta principal deste trabalho é apresentar um paradigma de resolução de uma família de problemas que facilite a migração de um ambiente para outro. Este paradigma baseia-se no esquema produtor - consumidor, aplicado aqui para resolver um problema importante da área de controle em tempo real de sistemas de energia elétrica, que é o problema do cálculo de fluxo de potência ótimo com restrições de segurança (FPORS), problema cuja formulação se presta naturalmente para processamento paralelo [1]. As chamadas às bibliotecas de comunicação devem então obedecer a padrões que sejam transparentes para a aplicação. E assim sendo esta biblioteca mudará de um ambiente para outro, devido as diferenças de protocolos, primitivas, etc, porém sem que este fato seja notado pela aplicação.

O cálculo do fluxo de potência ótimo com restrições de segurança é um problema de otimização de grande porte com m variáveis de controle e $(nc + 1)$ restrições; nc é o número de cenários ou configurações que são resultado da ocorrência de contingências, sejam elas simples ou compostas; para sistemas de potência de grande porte o número total de restrições está na ordem de vários milhões.

Utilizando a filosofia produtor-consumidor estruturou-se um eficiente algoritmo de resolução do problema de fluxo de potência ótimo com restrições de segurança, que apresenta as seguintes propriedades mais destacáveis:

- Considerável grau de portabilidade de um ambiente paralelo para outro.
- Assincronismo: As subtarefas executadas pelos processadores não são interrompidas. Todos eles operam com a informação mais recente.
- Balanço dinâmico de carga entre processadores.
- Robustez: O programa mestre pode invocar as subtarefas "escravas", quando as respostas requeridas dos outros processadores atrasarem ou, em caso extremo, não chegarem (falha).
- O mesmo modelo de programação (paradigma) é válido para resolver o problema quando são levadas em conta as capacidades corretivas do sistema (remanejamento pós-contingência).

Apresenta-se resultados de teste com um sistema brasileiro de 725 barras, 1212 linhas e 76 geradores controláveis, o que corresponde a um problema de otimização de 76 variáveis de controle e 1212*(1212+1) restrições, isto é, mais de *um milhão* de restrições. A biblioteca para programação paralela foi desenvolvida utilizando linguagem C e na aplicação utilizou-se Fortran.

A plataforma de desenvolvimento e testes foi um computador de memória compartilhada (desenvolvido pelo CPqD-Telebras) formada por nove processadores ligados a um barramento comum [2]. Presentemente está sendo feita a implementação em um sistema de memória distribuída, constituído de uma rede de estações SUN que é utilizada como se fosse uma máquina paralela virtual.

2.0 Despacho Econômico com restrições de Segurança

O objetivo do despacho econômico é determinar o ponto de operação de custo mínimo, respeitando as restrições operacionais do sistema. Já o despacho econômico com restrições de segurança tem o mesmo objetivo, porém a solução é restrita a não ter violações nos cenários pós-contingência em um universo de contingências consideradas prováveis. Este universo pode envolver milhões de restrições que devem ser levadas em conta no processo de otimização.

Seja x o vetor das variáveis de controle, c o vetor de custos correspondente. O problema do despacho econômico com restrições de segurança pode ser formulado assim:

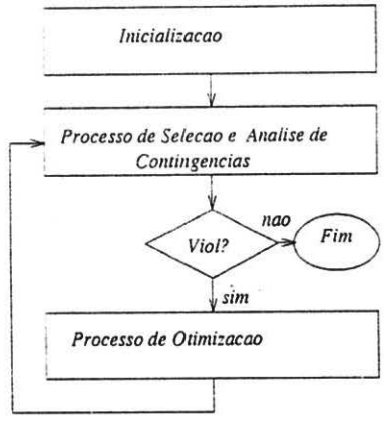
$$Min\ f = c^t x \qquad (1.a)$$

$$Ax \leq b \qquad (1.b)$$

$$A_i x \leq b_i \quad i = 1, ..., nc \qquad (1.c)$$

$$x_{min} \leq x \leq x_{max} \qquad (1.d)$$

O algoritmo paralelo utilizado neste trabalho é uma extensão do algoritmo originalmente sugerido em [8] para a solução sequencial para o problema de fluxo de carga ótimo com restrições de segurança. Na figura (1) apresenta-se a algoritmo sequêncial clássico para o problema de despacho econômico.



Fig(1): Algoritmo sequencial

3.0 Formulação para Processamento Paralelo: Decomposição em subtarefas

Para sua formulação para processamento paralelo. é necessário desacoplar esse algoritmo em subtarefas, o que é feito a seguir. O problema é dividido em dois estágios. Num primeiro estágio é resolvido o problema (1.a) sujeito à (1.b) e (1.c), que representa as restrições operacionais do caso base. Para isto utilizou-se o modelo incremental compacto formulado em [3]. O segundo estágio consiste na resolução de todo o conjunto de restrições de contingências, isto é eq. (1.c); a solução obtida no caso base é gradualmente melhorada a medida que novas restrições que são calculadas e analisadas são incluídas.

A granularidade das aplicações, e do hardware, é fundamental para a eficiência global do processo de paralelização. A idéia básica é de se ter "grãos" relativamente grandes de tal forma a reduzir o overhead de comunicações. A escolha do "grão" paralelo (escolha das subtarefas que serão executadas em paralelo) é um aspecto de grande importância pois ela influenciará diretamente os resultados do desempenho do algoritmo. Assim, é desejável que o tempo utilizado no processo para sincronização e troca de mensagens seja muito menor que o tempo utilizado pela computação de um tarefa paralela. A determinação dos "grãos" passa pela identificação das subtarefas que sejam fracamente acopladas, isto é, o fluxo de informação entre elas é muito menor do que os processos que elas executam.

O segundo estágio é apropriado para uma resolução em paralelo do algoritmo por duas razões principais: ela trabalha com uma alta granularidade (que é o problema de análise de contingência) e que as contingências calculadas são quase independente de estados passados. Quando mencionamos quase independentes é porque a alteração do ponto calculado no caso base pode alterar o resultado na análise das contingências. Assim um esquema assíncrono eficiente pode ser utilizado para a resolução do problema proposto.

O problema de Despacho Econômico com Restrições de Segurança é decomposto basicamente em quatro subtarefas:

- 1) Otimização.
- 2) Cálculo do índice de severidade.
- 3) Ordenação da lista de contingências.

- 4) Análise de contingências e geração de restrições.

A subtarefa (1) calcula uma versão relaxada do problema dado pela equação (1), de acordo com o método compacto [3]. Inicialmente todas as restrições são relaxadas e a solução do caso base é obtida através do subproblema dado pelas equações (1..a) e (1.b).

A subtarefa (2) calcula o índice de severidade de uma contingência [4]. Esta subtarefa executa uma rápida análise da contingência afim de obter o fluxo de potência no estado pós-contingência para cada caso.

A subtarefa (3) ordena a lista de contingências de acordo com o índice de severidade produzido pela subtarefa (2) e seleciona um subconjunto crítico de contingências (as outras são temporariamente deixadas de lado).

A subtarefa (4) verifica se a contingência leva a uma violação dos limites operacionais. Se acontecer, gera uma restrição que será incorporada ao processo de otimização pela subtarefa (1). Esta subtarefa executa um cálculo similar ao utilizado pela subtarefa (3) para a análise da contingência.

Fig(2): Sub-tarefas do modelo produtor-consumidor.

A interdependência entre as subtarefas descritas introduz sincronismo no processo global, podendo levar a uma queda na eficiência do esquema paralelo [1][4]. Para contornar isto, é necessário que cada estágio utilize a informação **mais recente** recebida do outro, e que essa informação seja aproveitada no máximo antes de ser descartada. Na seção seguinte descreve-se as caracteristicas básicas do algoritmo.

4.0 Algoritmo Proposto

O algoritmo proposto aqui fundamenta-se na filosofia **produtor-consumidor** (figura (2)) onde o problema de FPORS a visto como um conjunto de subtarefas que geram e/ou consomem produtos, conforme o estágio do processo de resolução. Assim, a subtarefa (1) é consumidora de restrições e gera novos estados; a subtarefa (2) recebe casos para analisar e fornece o índice desempenho correspondente e a subtarefa (4) "consome" casos (contingências) e "estados" (valor mais recente do vetor de estado do sistema), gerando como produto restrições de contingências. A subtarefa (3) é utilizada somente

durante um primeiro estágio e ela recebe os índices de desempenho e a contingência, produzindo uma lista de contingências ordenadas. Além disso tem-se que:

- O "consumo" e "produção" em cada bloco é feito de maneira assíncrona.
- Todas as subtarefas operam com os "produtos" mais atuais disponíveis. Por outro lado, subtarefas podem gerar "produtos" mais rapidamente do que eles são consumidos. Isto é estocado em filas - estrutura *fifo* - para sua utilização, pois essa informação pode ainda ser valiosa para o processo global.
- O programa Mestre dispõe de **todas** as subtarefas descritas na seção anterior. Isto significa que ele tem a capacidade de assumir todas as subtarefas, seja para ajudar aos processadores restantes, ou para assumir o processo todo de maneira autônoma.

5.0 Alocação de Tarefas nos Processadores

Na seção anterior foram identificadas as subtarefas produtoras/consumidoras em que a aplicação foi decomposta. A seguir essas subtarefas serão mapeadas nos programas Mestre e Escravo.

O programa **Mestre** é responsável pela iniciação e carga dos processadores e contém todas as subtarefas descritas anteriormente;o programa **escravo** é responsável pelas subtarefas (2) e (4).

Os pseudo códigos para os programas Mestre e Escravo são mostrados a seguir:

```
*Leitura de dados
*Otimiza caso base
do while(caso < nc )
  if(Fifo não vazia)
     *recebe casos dos escravos
  else
     *calcula casos
  endif
enddo
*Ordena casos
Do while(existam casos não marcados)
  if(Fifo não vazia)
     *Otimização com contingências
     *Remite novo ponto de operação
  elseif(Fifo não vazia)
     *Processo de análise de contingências
     if(caso causa violação)
        *Otimização com contingências
        *Remite novo ponto de operação
     endif
  endif
```

```
enddo
*Stop
```

Fig(3): Pseudo-código do programa mestre.

```
do while(caso < nc )
  *Calcula casos
  *Coloca na Fifo
enddo
do while(Existam casos não marcados)
  *Processo de análise de contingências
  *Coloca na Fifo
enddo
```

Fig(4): Pseudo-código do programa escravo.

O processo começa com o primeiro estágio que consiste na ordenação das contingências após otimizado do caso base (restrições de segurança relaxadas). O programa mestre ativa a subtarefa (2) nos processadores escravos para que calculem os índices de sobrecarga (índice de severidade da contingência [4]). Eles, então "adquirirão" casos e produzirão índices que são colocados numa fila para que por sua vez sejam consumidos pelo mestre. Se o mestre achar a fila vazia, então ele assume a subtarefa (2), e passa a calcular índices da mesma forma que os escravos.

Quando todos os índices tiverem sido calculados, então o programa mestre ordena a lista de casos por grau de severidade dado pelo índice (subtarefa (3)).

No segundo estágio o programa mestre ativa a subtarefa (3) nos processadores escravos e no seu processador é ativada opcionalmente ou a tarefa (2) ou (4): Os escravos retiram (consomem) um caso da lista e lêem o valor atual do vetor de estado, então realizam uma análise de contingência do caso; se detectada violação montam (produzem) uma restrição linear que é colocada em uma fila para o mestre retirar. No processador mestre é acionada a subtarefa (2), para o qual precisa "consumir" restrições da fila. Se a fila estiver vazia, então é ativado a subtarefa (4), assumindo temporariamente o papel de escravo.

Note-se que apesar de uma restrição ter sido calculada para um ponto ótimo passado, ela ainda pode ser útil. O mestre verifica, através de um teste rápido, a validade das restrições que estão chegando via fila. Este teste consiste na avaliação da restrição para o valor atual das variáveis de controle (disponíveis no modelo compacto de otimização [3]), e compara com o limite correspondente. O algoritmo visa manter sempre todos os processadores estejam trabalhando; apenas no final a carga é desbalanceada, porém por um período curto de tempo.

Note-se também que uma propriedade do algoritmo proposto é que uma falha nos processadores escravos não interrompe o processo global de resolução. No caso extremo, quando o processador mestre perde todo contato com os escravos, ele assume todas as subtarefas, executando o processo todo até o fim, se necessário. Isto é especialmente adequado para arquiteturas com processadores remotos onde a comunicação pode ver-se comprometida, seja por tráfego devido a outros processos ou por problemas de hardware na rede.

Critério de Parada

Por ser um esquema altamente assíncrono, o critério de parada deve receber um tratamento cuidadoso, para que a otimalidade não seja afetada. Na mesma implementação, utilizou-se uma filosofia de marcas. O caso da lista de contingências testado para o estado mais recente é marcado quando não produz quaisquer violações para esse estado. Assim, a solução ótima é atingida quando todos os casos da lista tenham sido marcados para o último estado produzido pelo processo de otimização.

6.0 Ferramentas para Processamento Paralelo [7]

O desenvolvimento de uma aplicação em ambiente paralelo requer cuidados adicionais que devem ser tomados pelo programador, pois novos conceitos são introduzidos como sincronização e troca de informações entre programas que executam em processadores diferentes. Existem diferentes maneiras de se implementar a cooperação entre os programas e normalmente a escolha de uma metodologia se faz com relação a arquitetura de hardware que irá ser utilizado. Esta porém, não é a melhor opção, pois uma eventual evolução da plataforma de hardware implicará em novo desenvolvimento da aplicação. Assim uma idéia é fornecer para aplicação um conjunto de funções que tornem transparente a arquitetura envolvida. Estas funções ficam, então, responsáveis por fornecer serviços variados para sincronização e troca de informações entre os programas. À aplicação cabe a tarefa de avaliar e de levar em conta os tempos usados para sincronismo e ou troca de informações.

6.1 Mapeamento do Problema no ambiente de memória comum

A partir da figura (5) procuramos agora mostrar como está implementada a interface de comunicação e como fica a base de dados. No modelo de memória comum [7], n processadores utilizam como meio de comunicação o barramento que permite o acesso a uma região de memória, a qual é chamada de memória comum. Assim a base dos dados fica alocada nesta região. Para o acesso exclusivo a esta base de dados, está implementado um conjunto de funções que mascaram problemas de sincronismo entre programas paralelos. Estas funções fazem parte de uma biblioteca que é invocada pelos programas mestre e escravo.

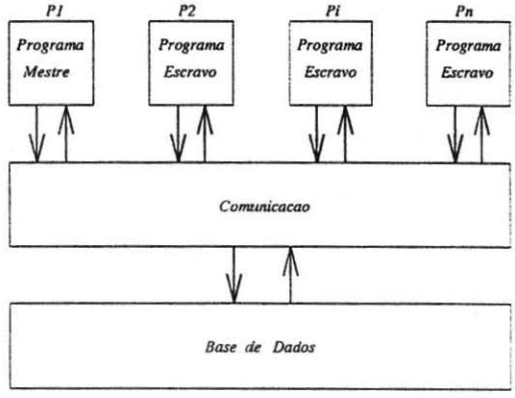


Fig (5): Esquema de multiprocessamento

6.2 Breve Descrição da Biblioteca

O mapeamento de um algoritmo em uma arquitetura paralela real, requer a existência de certas ferramentas de programação. A seguir apresenta-se duas destas (Getsub e Fifo) usados na comunicação dos programas paralelos. Uma estrutura básica chamada semáforo, a qual é usada na implementação destas rotinas também é brevemente discutida. As rotinas são usadas para o gerenciamento do acesso aos dados na área comum. Cada rotina é composta de variáveis locais que ficam alocadas na memória comum, e subfunções que coordenam a sincronização e troca de mensagens nas áreas definidas. Outro aspecto importante é que as funções podem trabalhar sobre áreas distintas da memória comum, de tal forma que cada área possua uma identidade própria conhecida pela tarefa que irá a utilizá-la. Para este problema utilizam-se dois tipos de funções:

6.2.1 Semáforos

A sincronização para comunicação entre programas paralelos é feita com mecanismos tipo semáforos. O semáforo é um conceito importado da computação concorrente [6], sendo definido como uma váriavel positiva na qual são realizadas as operações de "bloqueio" e "liberação". A operação de bloqueio é feita quando um processo deseja utilizar de um recurso de maneira exclusiva e a operação de "liberação" é feita para permitir outros processos a utilizarem este recurso. Os semáforos são relativamente simples de serem implementados e podem ser utilizados diretamente pela aplicação ou fazerem parte de rotinas de mais complexas. O uso dessas rotinas, pela aplicação faz que o acesso exclusivo a dados comuns, fique transparente, o que simplifica a codificação e facilita a portabilidade.

6.2.2 Função de tarefas Getsub

O estilo de programação assíncrona adotado neste trabalho ajuda no balanceamento da carga computacional sobre os processadores envolvidos no cálculo do problema. Para suporte deste estilo, a execução de uma subtarefa deve ser feita de maneira flexível e independente. Uma tarefa pode ser obtida através de um índice pertencente a uma lista, por exemplo, neste caso, uma contingência. Uma subtarefa (por exemplo: o cálculo do índice de severidade) pode estar executando em qualquer processador e requisitar uma tarefa simplesmente chamando a função Getsub. A rotina Getsub é um pedaço do código paralelo que pode ser executada de qualquer processador (mestre ou escravo), ela assegura a exclusividade do índice retornado (número da tarefa). A sua implementação é baseada em semáforos e o índice da tarefa a ser executada fica residente na memória comum.

6.2.3 Função Fifo

Durante a solução do problema, na medida que as restrições são analisadas e casos com violações são encontrados, eles podem eventualmente ser incluídos na lista de contingências pelo programa mestre. De acordo com o mecanismo assíncrono proposto neste trabalho, estas contingências são colocadas em uma fila (first-in-first-out) a qual é gerenciada por esta função de Fifo, que é chamada tanto pelo mestre (que atua como consumidor de recursos - contingências e violações) como pelos escravos (produtores). Note que poderia ser adotado uma metodologia síncrona na qual o mestre ficaria esperando (por exemplo em uma barreira) a ocorrência de uma violação. Na implementação assíncrona as violações são enfileiradas de forma que nem mestre nem escravo fique parado. Para manipulação da Fifo existem quatro sub funções responsáveis pela inicialização da fila (FifoInit), produção (FifoPut), consumo (FifoGet) e estado da fila (FifoStat). A fila fica residente em memória comum e é:

Programa Mestre

Lê dados
Inicializa FIFO e GETSUB
Cria os programas paralelos
Executa o subrotina principal
Imprime resultados

Fig(6): Programa Mestre

7.0 Resultados Experimentais

O algoritmo proposto foi testado utilizando um sistema de transmissão brasileiro operando em carga média e formado por 725 barras, 1212 linhas e 76 geradores controláveis. O número de cenários foi fixado em 50, 100 e 900 casos. Os "speed-up" para essas variantes são ilustrados no gráfico da figura (7). Como é de esperar a eficiência do algoritmo cresce com o números cenários que são envolvidos no processo de otimização (para alguns casos obtem-se "speep-up" acima do ideal). Na prática a lista de casos é grande pois envolve a saída de outros dispositivos e combinação deles.

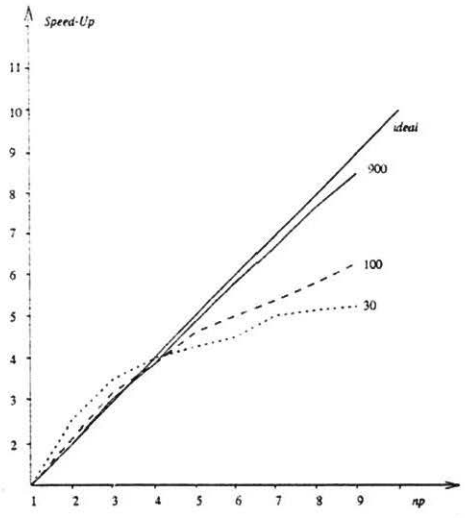
8.0 Comentários

Os resultados mostram a eficiência do algoritmo proposto e a sua versatilidade. Sua característica assíncrona faz que de maneira automática tire proveito máximo da capacidade de processamento disponível alocando outras subtarefas obtendo-se um natural balançamento da carga entre os processadores.

Deve-se notar também que a mesma estrutura é válida para resolver a família de problemas do tipo fluxo de potência ótimo-seguro. Assim, por exemplo, se o problema a resolver é apenas análise de contingências, a subtarefa (1) não é realizada e o processador mestre coopera com os processadores escravos realizando a tarefa (3) simplificada (sem criação de restrição). Por outro lado, quando são levadas em conta as capacidades corretivas do sistema (pós-despacho) tem-se uma granularidade ainda maior para a subtarefa (3), pois além do processo de análise de contingências tem-se um processo de resolução de um problema de otimização (subproblema da operação)[1][5].

A subtarefa de otimização trabalha inicialmente com um conjunto crítico de casos que é facilmente identificável utilizando-se um critério simples de tolerância, que é suficiente para a maioria das aplicações práticas. No caso estudado, os casos considerados dentro do conjunto crítico foram aqueles com índice de sobrecarga maior do que 0.001 p.u..

Note-se que no caso de falha de um (ou todos) processador escravo, o processo de resolução não é interrompido; a única consequência será o natural aumento do tempo global de processamento.



Fig(7): Speed-up para tres diferentes numeros. de casos.

9.0 Conclusões

Neste trabalho apresentou-se um algoritmo para a resolução do problema do cálculo do Fluxo de Potência Ótimo com Restrições de Segurança utilizando processamento paralelo baseado na abstração produtor-consumidor. A visualização desse problema utilizando este paradigma permite a concepção de um algoritmo eficiente e portável, no nível da aplicação, de uma arquitetura computacional para outra. O mesmo algoritmo pode ser utilizado para outros problemas do mesmo tipo, por exemplo quando no problema tratado são consideradas las capacidades corretivas do sistema de potência frente a um distúrbio. O desenvolvimento foi feito utilizando como plataforma de trabalho um computador paralelo de memória compartilhada com barramento comum.

10.0 Agradecimentos

Os autores agradecem o apoio financeiro da Fundação de Amparo à Pesquisa do estado de São Paulo - FAPESP - no desenvolvimento deste trabalho.

11.0 Referências

[1] A.Monticelli, M. V. F. Pereira, " A New Generation of Energy Management Systems" Proc. PSCC, pp 99-104, Portugal, 1987.

[2] Cavalli E., Zbeu M.C., "Sistema hardware para processamento paralelo", Anais I SBACC, maio 1987.

[3] B. Stott, J.L. Marinho, "Linear Programming for Power System Network Security Applications" , IEEE Trans. Power App. Syst., Vol PAS-98, pp 837-848, may/june 1979.

[4] B. Stott, O. Alsac, A. Monticelli, "Security Analysis and Optimization", Proc. of the IEEE, Special Issue on Computers in Power System Operations, Vol 75, pp 1623-1644, Dec 1987.

[5] M.J. Texeira, H.J.C.P. Pinto, M.V.F. Pereira and M.F. McCoy, "Developing Concurrent Processing Applications to Power System Planning and Operations", Sixteenth PICA Conf., Seattle, 1989.

[6] Dijsktra E.W., "Cooperating Sequencial Processes", In F. Genuys, Programming Languages, Academic Press NY 1968.

[7] Marcelo Rodrigues - Tese de Mestrado Unicamp; em preparação.

[8] O. Alsac and B. Stott, "Optimal Load Flow with Steady-State Security", IEEE Transactions on Power Apparatus and Systems, Vol.93, pp.745-751, May/June 1974.

Biografias

Alcir J. Monticelli , graduou-se em Engenharia Elétrica pelo ITA em 1970, obteve o título de mestre na UFPb em 1972, e concluiu seu doutoramento na UNICAMP em 1975, onde atualmente é professor titular junto ao Depto. de Sistemas de Energia Elétrica. De setembro de 1982 a agosto de 1985 ele foi professor visitante do Depto. de Engenharia Elétrica e Ciências da Computaçaõ, Universidade da Califórnia, Berkeley, e de setembro de 1990 a agosto 1991 pesquisador visitante no Laboratório de Sistemas Industriais, Mitsubishi Elec. Co., Japão.(e-mail: *alcir@dsee.fee.unicamp.br).*

Marcelo Rodrigues , graduou-se em Engenharia Elétrica em 1984 na UNICAMP Brasil. De 1986 à 1989 ocupou a diretoria de desenvolvimento de software na HST Ltda. Atualmente está concluindo mestrado na UNICAMP. Suas principais áreas de pesquisa são software básico em ferramentas de programação paralela.(e-mail: *marcelo@dsee.fee.unicamp.br).*

Osvaldo R. Saavedra concluiu sua graduação em Engenharia Elétrica em 1981 na Universidade do Norte, Chile, e seu mestrado em 1988, na UNICAMP, Brasil. De 1981 até 1985 foi professor tempo parcial do Depto. de Eletrônica, Universidade de Tarapacá-Chile. De 1983 até 1986 ocupou a diretoria de Inecom Engenheiros Ltda-Chile. Atualmente está concluindo seu doutoramento em Engenharia Elétrica na UNICAMP. Suas principais áreas de pesquisa são planejamento e controle de sistemas elétricos de potência. (e-mail: *osvaldo@dsee.fee.unicamp.br).*