# Avaliação da confiabilidade de sistemas compostos de geração-transmissão de energia elétrica em computadores paralelos

Carmen L. T. Borges[1] Djalma M. Falcão[2] João C.O. Mello[3] Albert C.G. Melo[3]

[1] COPPE/UFRJ-Programa de Engenharia de Sistemas, EE/UFRJ-Departamento de Eletrotécnica, carmen@coep.ufrj.br.
[2] COPPE/UFRJ-Programa de Engenharia Elétrica, falcao@coep.ufrj.br.
[3] CEPEL-Centro de Pesquisa de Energia Elétrica, albert@fund.cepel.br, joao@fund.cepel.br.

**Resumo**

Este trabalho apresenta métodos de análise de confiabilidade de sistemas elétricos de potência compostos de geração/transmissão usando simulação Monte Carlo em computadores paralelos. São discutidas questões relativas à distribuição da carga computacional entre processadores. São propostas três implementações paralelas diferentes, variando principalmente no grafo de tarefas e no controle da convergência paralela. O desempenho de algoritmos síncronos e assíncronos é estudado sobre três sistemas elétricos distintos. Os resultados demonstram ótima eficiência obtida no computador paralelo de memória distribuida IBM RS/6000 SP.

**Abstract**

This paper presents methods for the reliability evaluation of electric power systems composed of generation/transmission sub-systems using Monte Carlo simulation on parallel computers. Questions concerning the distribution of computational load among processors are discussed. Three different parallel implementations are proposed, varying mainly on task graph and parallel convergence controll. The performance of both synchronous and assynchronous algorithms is studied on three distinct electric test systems. The results demonstrate very good eficiency on the IBM RS/6000 SP distributed memory parallel computer.

## 1. INTRODUÇÃO

Os sistemas de geração, transmissão e distribuição de energia elétrica constituem elemento básico no desenvolvimento econômico e social das sociedades modernas. Por razões técnicas e econômicas, esses sistemas evoluíram de um conjunto de pequenos sistemas isolados para grandes e complexos sistemas interligados com dimensões nacionais ou, até mesmo, continentais. Por esta razão, falhas em determinados componentes do sistema podem desencadear perturbações capazes de afetar um grande número de consumidores. Por outro lado, devido à sofisticação dos equipamentos elétricos e eletrônicos utilizados pelos consumidores, as exigência em termos da

confiabilidade do suprimento de energia elétrica tem aumentado consideravelmente. Mais recentemente, mudanças institucionais no setor de energia elétrica, tais como aquelas provocadas por políticas de desregulamentação, privatizações, restrições ambientais, etc., têm forçado a operação de tais sistemas mais próximos de seus limites aumentando a necessidade de avaliar, de forma mais precisa, os riscos de interrupção ou degradação da qualidade do suprimento de energia elétrica.

Modelos probabilísticos têm sido utilizados cada vez mais na avaliação do desempenho de sistemas de energia elétrica. A partir de informações relativas a falhas de componentes do sistemas, esses modelos permitem estabelecer índices de desempenho do sistema os quais podem ser utilizados para auxiliar a tomada de decisões relativas a novos investimentos, políticas operativas e para balizar transações no mercado de energia elétrica. Esse tipo de estudo recebe o nome genérico de Avaliação da Confiabilidade [1] e pode ser realizado nos níveis de geração, transmissão e distribuição ou, ainda, combinando os vários níveis. Neste último caso teríamos os estudos de confiabilidade composta dos quais o mais utilizado na prática é a avaliação da confiabilidade de sistemas compostos de geração-transmissão, objeto deste trabalho.

O objetivo básico da avaliação da confiabilidade em sistemas de geração-transmissão de energia elétrica é avaliar a capacidade do mesmo em satisfazer a demanda de energia elétrica nos seus principais pontos de consumo de energia. Para tanto, considera-se a possibilidade de falhas em componentes do sistema de geração e de transmissão e avalia-se o impacto dessas falhas no suprimento de energia. Existem duas abordagens possíveis para avaliação da confiabilidade: técnicas analíticas e simulação estocástica (simulação Monte Carlo [2,3], por exemplo). No caso de sistemas de grande porte, com condições operativas complexas e grande número de eventos severos, a simulação Monte Carlo é, geralmente, preferida tendo em vista a facilidade de modelar fenômenos complexos.

A avaliação da confiabilidade baseada na simulação Monte Carlo exige a análise de uma quantidade muito elevada de estados operativos do sistema. Esta análise, em geral, inclui o cálculo de fluxo de potência, análise estática de contingências, reprogramação da geração, corte de carga, etc. Em muitos casos esta análise deve ser repetida para diferentes cenários de carga e configuração da rede elétrica. No caso de redes elétricas com dimensões elevadas, este tipo de avaliação pode necessitar de horas de processamento em estações de trabalho de alto desempenho. A maior parte deste esforço computacional concentra-se na análise dos estados operativos. Esta análise pode ser realizada de forma independente para cada estado operativo o que sugere a utilização de computação paralela para redução do tempo total de computação.

Neste artigo são descritos resultados preliminares obtidos através de uma metodologia em desenvolvimento para análise da confiabilidade de sistemas compostos de geração-transmissão de energia elétrica em computadores paralelos com arquitetura computacional do tipo memória distribuída e comunicação por troca de mensagens. A metodologia está sendo desenvolvida usando-se como elemento de referência um programa de análise de confiabilidade utilizado por empresas de energia elétrica brasileiras [4][#] .

[#] O programa NH2 desenvolvido pelo CEPEL.

## 2. CONFIABILIDADE DE SISTEMAS DE GERAÇÃO-TRANSMISSÃO

A avaliação da confiabilidade de sistemas compostos de geração-transmissão consiste na avaliação de vários índices de desempenho tais como a probabilidade de perda de carga (LOLP), expectância de potência não suprida (EPNS), freqüência de perda de carga, etc., usando um modelo estocástico da operação do sistema de energia elétrica. Um algoritmo conceitual para esta avaliação é dado a seguir [3]:

1. Selecione um cenário operativo $x$ correspondente a um nível de carga, disponibilidade de componentes, condições de operação, etc.
2. Calcule o valor de uma função de avaliação $F(x)$ a qual quantifica o efeito de violações nos limites operativos neste cenário específico. O efeito de ações corretivas tais como reprogramação da geração, corte de carga, etc., pode ser incluído nesta avaliação.
3. Atualize o valor esperado dos índices de confiabilidade baseado no resultado obtido no passo 2.
4. Se a precisão das estimativas é aceitável, termine o processo. Em caso contrário, retorne ao passo 2.

Suponhamos um sistema com $n$ componentes (linhas de transmissão, transformadores, valores de carga, etc.). Um cenário operativo para esse sistema é dado pelo vetor aleatório $x = ( x_1, x_2, \ldots , x_n )$ onde $x_i$ é o estado do *i-ésimo* componente. O conjunto de todos os possíveis estados operativos, resultante de todas as combinações possíveis de estados dos componentes, é chamado de espaço de estados e representado por $X$. Na SMC, o passo 1 do algoritmo acima consiste em obter uma amostra do vetor $x \in X$, através do sorteio de variáveis aleatórias correspondentes aos estados operativos utilizando-se um gerador de números aleatórios.

O passo 2 do algoritmo acima exige a simulação da condição de operação do sistema nos respectivos estados operativos. Esta simulação exige a solução de um problema de fluxo de potência e, eventualmente, de um problema de fluxo de potência ótimo para simular o redespacho de geração e/ou o corte de carga mínimo. No caso de sistemas de grande porte, essas simulações exigem esforço computacional elevado em relação àquele necessário nos demais passos do algoritmo [5].

## 3. PARALELIZAÇÃO DO ALGORITMO

A análise de confiabilidade pode ser resumida em três grandes módulos: seleção de estados, análise de desempenho do sistema em contingência e o cálculo dos índices de confiabilidade. Conforme descrito, para cada estado selecionado por sorteio na simulação Monte Carlo, é feita a análise de desempenho ou análise de contingência, isto é, a verificação da possibilidade do atendimento à demanda de potência sem violação dos limites operativos. O algoritmo é inerentemente paralelo com um elevado grau de desacoplamento [6]. A análises das contingências são independentes entre si e a comunicação só se faz necessário em três situações distintas: 1) Para a distribuição inicial dos dados de entrada do problema, idênticos para todos os processadores e executado uma vez durante todo o processamento; 2) Para o agrupamento final dos resultados

parciais calculados em cada processador, também executado uma única vez; e 3) Para o controle da convergência global, quando os dados de convergência local dos processadores precisam ser agrupados em um deles para que a convergência global seja verificada. Necessita ser executado várias vezes durante o processo iterativo, com frequência que obedeça a algum critério de controle, que pode ser por tempo de simulação, número de contingências analisadas, convergência local ou outros.

A configuração básica para paralelização deste problema é a mestre-escravos, onde um processo, denominado mestre, é responsável por adquirir os dados de entrada, distribuí-los para os escravos, controlar a convergência global, receber os resultados parciais de cada processo, calcular os índices de confiabilidade e gerar relatórios. Os processos escravos, por sua vez, são responsáveis por analisar as contingências a eles alocadas, enviar para o mestre a sua convergência local e, ao final do processo iterativo, enviar também os seus resultados parciais. É importante salientar que, em arquiteturas onde os processadores têm capacidades de processamento equivalentes, o processo mestre também deve atuar como escravo a fim de melhorar o desempenho do algoritmo. Para fins deste trabalho, cada processo é alocado a um processador diferente, passando daqui por diante a ser referenciado diretamente por processador.

O principais pontos que necessitam ser solucionados na paralelização do algoritmo são:

1. Identificação dos dados que serão enviados como mensagem entre mestre e escravos, bem como das variáveis que deverão ser replicadas em todos os processadores e mantidas como locais;
2. Distribuição das contingências entre os processadores de forma a que todos realizem processamento útil para a precisão estatística do resultado final;
3. Estabelecimento do critério de controle da convergência global de forma a permitir um bom balanceamento de carga;
4. Cálculo dos índices de confiabilidade globais do sistema a partir dos índices parciais calculados em cada processador.

## 4. DISTRIBUIÇÃO DAS CONTINGÊNCIAS

Na implementação de métodos de simulação Monte Carlo em paralelo, o problema principal a ser evitado é a existência de correlações entre as sequências de números randômicos gerados em processadores diferentes. Se as sequências forem correlatas de alguma forma, as informações produzidas pelos diferentes processadores serão redundantes e não ajudarão a aumentar a precisão estatística da computação, além de piorar o desempenho do algoritmo. Em algumas aplicações de Monte Carlo, as correlações introduzem interferências que produzem resultados incorretos. Inicializar o gerador de números randômicos com sementes diferentes para cada processador não é uma boa técnica, pois pode gerar sequências correlatas entre si [7].

Em ambiente sequencial o problema da correlação também existe e não é trivial, porém já é bem estudado e conhecido. Dessa forma, reduzir o problema paralelo a algo equivalente ao problema sequencial seria interessante. Um enfoque prático é desenvolver um método paralelo que imite exatamente o comportamento do método sequencial, com a vantagem adicional de facilitar a depuração de códigos paralelos complexos.

Neste trabalho, o algoritmo de geração dos números randômicos não foi paralelizado. Todos os processadores recebem a mesma semente. Todos os processadores executam os mesmos sorteios. Cada processador, no entanto, começa a analisar a contingência de número igual ao seu *rank* na computação paralela e passa a analisar as próximas contingências em passo igual ao número de processadores envolvidos na computação. Supondo que o número de processadores disponíveis seja 4, o processador 1 analisa as contingências 1, 5, 9,..., o processador 2 as contingências 2, 6, 10, ... e assim sucessivamente. Desta forma, exatamente as mesmas contingências que seriam analisadas sequencialmente o são em paralelo e cada processador fica responsável pelas mesmas contingências, independente do número máximo de sorteios ou da tolerância de convergência especificados.

## 5. IMPLEMENTAÇÃO PARALELA

Foram implementadas três versões distintas de paralelização do problema, variando sobre o grafo de tarefas, o critério de distribuição das tarefas entre os processadores e o critério de controle da convergência global.

Nas considerações que se seguem, considere $p$ o número de processadores alocados para a execução. Cada processador possui um *rank* na computação paralela, variando de *0* a *(p-1)*. O processador *0* é o mestre e os demais processadores *(1* a *p-1)* são os escravos. Nos grafos de tarefas apresentados, cada círculo representa uma tarefa executada por um processador. As tarefas necessárias para a solução deste problema podem ser classificadas em 5 tipos: **I** - **I**nicialização, **A** - **A**nálise de Contingências, **C** - Verificação da **C**onvergência, **P** - **P**arada do Processo Iterativo e **F** - **F**inalização (cálculo dos índices e geração de relatórios). Um subíndice *i* associado a uma tarefa *T* significa que a tarefa *T* está alocada ao processador *i*. Um superíndice *j* associado à $T_i$ significa a j-ésima execução da tarefa *T* pelo processador *I*.

### 5.1 Versão 1

A primeira versão é a que apresenta implementação mais simples. O grafo de tarefas para esta versão está apresentado na figura 1.

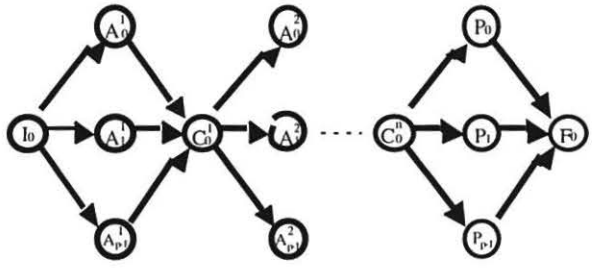


FIGURA 1 - Grafo de Tarefas das versões 1 e 2

A inicialização do problema, que consiste na obtenção dos dados de entrada e alguma computação inicial, é executada pelo processador mestre. Em seguida, o mestre faz um *broadcast* destes dados para todos os processadores escravos. A partir de então entra-se em um processo cíclico que consiste em, tanto escravos como o mestre, executarem a análise de um número fixo e idêntico de contingências e, ao final, comunicarem ao mestre seus dados de convergência local. De posse destes dados, o processador mestre verifica a convergência global, calcula o número total de contingências analisadas e em seguida faz um *broadcast* destes resultados. Se a convergência global ou o tamanho máximo da amostra não tivererm sido atingidos, todos os processadores analisam outro lote fixo de contingências. Esse processo continua até que um dos dois critérios de parada seja atendido. Neste caso, os escravos enviam para o mestre os seus resultados parciais calculados e o mestre então calcula os índices de confiabilidade e gera relatórios.

Essa versão possui como maior vantagem a facilidade de implementação, já que a adaptação do programa sequencial para o paralelo exige menor esforço. No entanto, o seu desempenho fica prejudicado pelos seguintes fatores:

1. Desbalanceamento de Carga: Como as análises das contingências requerem tempos de processamento distintos, pois envolvem diferentes graus de complexidade na solução do sistema elétrico, a divisão do trabalho em lotes de mesmo número de contingências pode implicar em grande desbalancemanto de carga;
2. Tempo de Sincronização: O tempo de sincronização, que é o tempo gasto pelos processadores enquanto esperam os outros completarem as suas tarefas [8], pode ser elevado se o balanceamento de carga for ruim. Se na solução de seu lote de contingências um processador acabar bem antes dos demais, ou por ser de maior capacidade de processamento ou por ter resolvido casos de mais simples solução, ele precisará ficar esperando pelos demais. Se esse processador for um escravo, ficará esperando a decisão do mestre se deve continuar a processar ou não e o mestre, por sua vez, só poderá decidir após ter recebido as informações de todos os escravos. Se for o mestre a terminar primeiro, precisará ficar esperando que todos os escravos terminem para poder decidir.

Isso fica claro na análise do grafo de precedência mostrado na figura 2. No grafo de precedência de eventos [9], as linhas horizontais são os eixos de tempo locais dos processadores, aqui representados apenas o mestre e um escravo. Os arcos horizontais representam os estados sucessivos por que passa um mesmo processador. Os nós representam os eventos de envio e recebimento de mensagens. As mensagens são representadas pelos arcos ligando linhas horizontais associadas a processadores distintos.

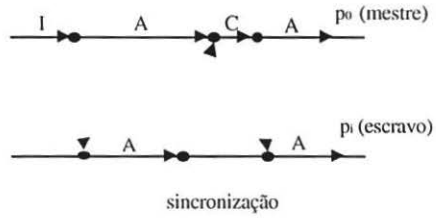


FIGURA 2 - Grafo de Precedência da versão 1

### 5.2 Versão 2

O objetivo principal desta versão é reduzir o tempo de sincronização provocado pelo desbalanceamento de carga da versão anterior. O grafo de tarefas é o mesmo da figura 1. A diferença básica está em que a distribuição de tarefas entre os processadores se dá por tempo de processamento e não mais por número de contingências. Após receber os dados de inicialização do mestre, cada processador (inclusive o próprio mestre) passa a executar análises de contingências por um intervalo de tempo fixo $\Delta t$ . Só após decorrido este intervalo, os escravos comunicam ao mestre as suas convergências locais, independente do número de contingências que analisaram. O processo prossegue de maneira análoga à versão anterior, implicando em um novo intervalo de tempo $\Delta t$ de análise de contingências para cada processador se a convergência global não for atingida.

Essa versão ainda mantém a facilidade de implementação da anterior, com a vantagem adicional de reduzir os possíveis desbalanceamentos de carga e, consequentemente, tempo de sincronização, haja visto que:

1. Balanceamento de Carga: Dentro de cada intervalo de tempo $\Delta t$, o processador analisa o número de contingências que conseguir em função da sua capacidade de processamento e da complexidade dos casos sob sua responsabilidade.
2. Tempo de Sincronização: O tempo de sincronização deve-se na sua maioria ao tempo gasto para o mestre receber os dados relativos às convergências locais dos escravos e verificar a convergência global. Durante este tempo os escravos ficam aguardando para saber se continuam a processar ou param. Isso pode ser verificado no grafo de precedência da figura 3.

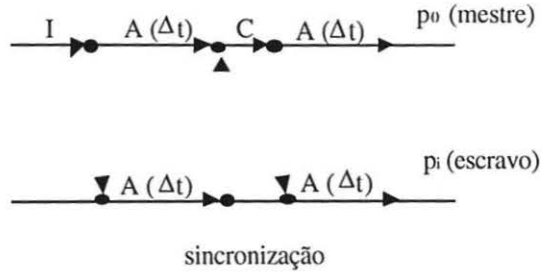


FIGURA 3 - Grafo de Precedência da versão 2

### 5.3 Versão 3

Essa é uma versão totalmente assíncrona de implementação da solução do problema. O grafo de tarefas desta versão é apresentado na figura 4.

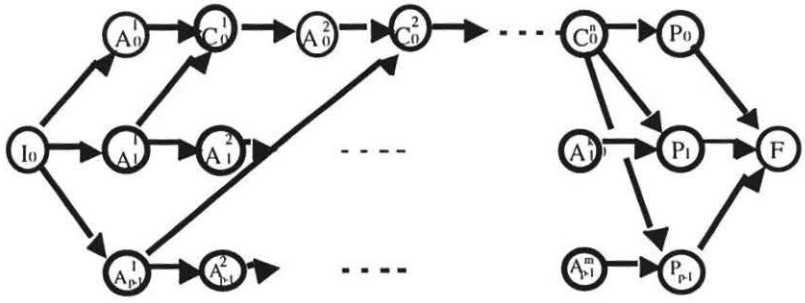

FIGURA 4 - Grafo de Tarefas da versão 3

A inicialização da computação é feita de maneira idêntica às versões anteriores, pelo processador mestre, seguida de um *broadcast* dos dados iniciais para todos os escravos. De posse destes dados, todos os processadores (inclusive o mestre) passam à fase de análise de contingências, sendo que aqui há diferenças entre o comportamento do mestre e o dos escravos. Os escravos permanecem na fase de análise por um intervalo de tempo fixo $\Delta t$. Após decorrido este intervalo, enviam para o mestre os dados relativos às suas convergências locais, independente do número de contingências que analisaram. Imediatamente a seguir, voltam a analisar contingências por outro intervalo de tempo $\Delta t$, findo o qual tornam a enviar dados de convergência para o mestre. Ou seja, enquanto o mestre verifica a convergência global, os escravos estão executando trabalho útil. O mestre, por sua vez, está continuamente checando a chegada de mensagem enviada por algum escravo. Ao receber uma mensagem, para de analisar contingências e verifica a convergência global. Se esta não ocorrer, volta a analisar contingências até receber nova mensagem quando verifica novamente a convergência global. Quando o mestre determina que um dos critérios de parada foi atingido, ele envia uma mensagem para cada escravo ordenando a interrupção da análise de contingências. Cada escravo, então, envia ao mestre seus resultados parciais calculados e termina seu processamento. O mestre, após receber todos os resultados parciais, calcula os índices de confiabilidade, gera relatórios e termina.

Esta implementação é a que oferece melhor aproveitamento tanto da arquitetura do computador paralelo como do grau de paralelismo do algoritmo. Isso porque não há qualquer espécie de sincronização durante o processo iterativo e o balanceamento de carga é o ideal, estabelecido pela capacidade dos processadores e complexidade das contigências em análise nos diferentes processadores. Por outro lado, a codificação e a depuração se mostram mais complicadas que das versões anteriores. O grafo de precedência está mostrado na figura 5.

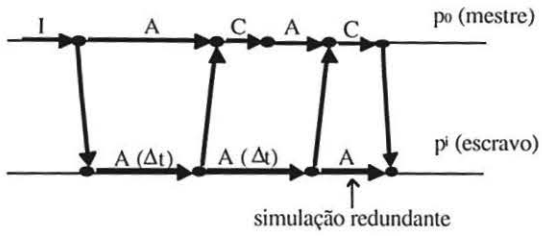


FIGURA 5 - Grafo de Precedência da versão 3

Uma consideração adicional introduzida é a simulação redundante. Durante o último estágio do processo iterativo, os processadores escravos executam algumas análises que não são necessárias para atingir a convergência, um vez que esta é determinada baseando-se nos valores enviados na última mensagem. Entre o envio da última mensagem e o recebimento da mensagem informando da convergência, os processadores escravos estão executando simulação que está além do necessário. Isso, no entanto, não implica em perda de tempo significativa para a computação como um todo. Essa simulação redundante é aproveitada no cálculo final dos índices de confiabilidade, gerando índices ainda mais precisos, já que em simulação Monte Carlo a incerteza da estimativa é inversamente proporcional ao número de sorteios analisados.

## 6. MESSAGE PASSING INTERFACE (MPI)

MPI é um sistema de troca de mensagem padrão e portátil desenvolvido para operar em uma grande variedade de computadores paralelos [10]. O padrão define a sintaxe e a semântica das subrotinas que integram a biblioteca. Existem várias implementações de MPI bastante eficientes, disponíveis para diferentes arquiteturas. Neste trabalho foi utilizada a implementação desenvolvida pela IBM para AIX que obedece à versão 1.1 do padrão MPI.

Troca de mensagens é um paradigma de programação largamente utilizado em computadores paralelos, especialmente nos computadores paralelos escaláveis com memória distribuída e em redes de estações de trabalho. O conceito básico consiste em processos alocados de forma distribuída comunicarem-se através de mensagens a fim de cooperarem na realização da computação. O mecanismo elementar de comunicação do MPI é a transmissão de dados entre um par de processos, um lado enviando e o outro recebendo, o que é chamado de comunicação ponto a ponto. Existem dois tipos de comunicação possíveis: bloqueante e não-bloqueante [11]. Um *send* bloqueante implica em um bloqueio do processo emissor até que o *buffer* de emissão possa ser reutilizado pelo programa. Analogamente, um *receive* bloqueante bloqueia o processo receptor até que o *buffer* de recepção realmente contenha a mensagem recebida. Na comunicação não-bloqueante, é possível sobrepor a transmissão da mensagem com a computação ou mesmo sobrepor a transmissão de múltiplas mensagens entre si, aumentando o desempenho do sistema. Operações não-bloqueantes são compostas de duas partes: a função de postagem, que inicia a operação e a função de teste por término, que permite à

aplicação verificar se a operação foi completada. Com hardware adequado, a transferência de dados do emissor pode ocorrer concorrentemente com outra computação executada no emissor, após o *send* ter iniciado e antes de ter terminado.

Dois aspectos que merecem especial atenção são a semântica das primitivas de comunicação e o protocolo que as implementa. Questões de semântica do tipo : "Quando um *send* completou, pode-se garantir que o *receive* correspondente completou ou até mesmo se já iniciou?", estão relacionadas com o protocolo que implementa estas operações. O padrão MPI oferece quatro modos para comunicação ponto a ponto, tanto bloqueante como não-bloqueante, que permite escolher a semântica da operação de *send* e, com efeito, influenciar no protocolo de trasferência de dados.

1. Modo *Buffered*: Uma operação de *send* pode ser iniciada quer o *receive* correspondente tenha sido postado ou não. Pode até mesmo completar antes que o *receive* seja postado. Desta forma, possui semântica de término local, pois seu término não depende da ocorrência do *receive* correspondente em outro processo. Um certo espaço de *buffer* está disponível, sendo necessário que o programa de aplicação o defina explicitamente. Na semântica que usa *buffer*, o protocolo bloqueia o processo pelo tempo mínimo, porém necessita efetuar mais cópias de dados.
2. Modo *Synchronous*: Um *send* pode ser iniciado quer o *receive* correspondente tenha sido postado ou não. No entanto, ele só termina se o *receive* tiver sido postado e a operação de recebimento da mensagem iniciada. A semântica *rendezvous* entre emissor e receptor é usada, ou seja, o término do *send* implica que o *receive* no mínimo já começou. A semântica de término é não local. Neste modo, a cópia de dados é minimizada, porém bloqueia o processo por um tempo maior.
3. Modo *Standard*: O término do *send* não significa necessariamente que o *receive* correspondente já começou e nenhuma consideração pode ser feita no programa de aplicação sobre se a mensagem a ser enviada será colocada no *buffer* pelo MPI. Ou seja, cabe ao MPI decidir se as mensagens enviadas usarão *buffer* ou não, baseado em otimização de espaço e/ou desempenho. A semântica de término é então não local.
4. Modo *Ready*: Um *send* só pode ser iniciado se o *receive* correspondente tiver sido postado, cabendo ao usuário assegurar que isso ocorra no programa. Esse modo permite ao usuário explorar o conhecimento que tem da aplicação para simplificar o protocolo e potencialmente melhorar o seu desempenho. A semântica de término é não local.

## 7. RESULTADOS

### 7.1 Sistemas Teste

Foram utilizados três sistemas elétricos para teste das versões. O primeiro, IEEE-RTS [12] (Reliability Test System), é um sistema padrão do IEEE para teste de confiabilidade. Possui 24 barras, 38 circuitos e 2 áreas interligadas. O segundo sistema, CICRÉ-NBS [13] (New Brunswick Power System), foi proposto no âmbito da CIGRÉ como sistema teste para comparação de modelos para cálculo de confiabilidade composta. Possui 89 barras, 126 circuitos e 4 áreas interligadas. O terceiro sistema utilizado foi o

NNE, uma representação do sistema interligado brasileiro da região norte-nordeste com as mesmas características elétricas de um sistema real. Suas dimensões são 89 barras, 170 circuitos e 6 áreas interligadas. Para todos os sistemas foi adotada uma tolerância para a convergência de 5% tanto na LOLP como EPNS.

### 7.2 Computador Paralelo

IBM RS/6000 SP Scalable POWERparallel System [14] - Cada nó desta máquina paralela é uma estação de trabalho completa, com sua própria CPU, memória RAM, disco e interface de rede. Os processadores são de arquitetura POWER2/PowerPC 604 ou Processador Simétrico (SMP) PowerPC. Os nós são interligados por um *switch* de alta velocidade, dedicado exclusivamente para a execução de programas paralelos. Este *switch* pode estabelecer conexões diretas entre quaisquer pares de nós, podendo existir simultaneamente até uma conexão *full-duplex* por nó. O sistema pode escalar até 16 nós por gabinete e quando interligados, esses gabinetes formam um sistema de até 512 nós.

O computador paralelo que serviu de plataforma para implementação deste trabalho foi o IBM RS/6000 SP instalado no NACAD/COPPE-UFRJ. Possui 4 nós de arquitetura POWER2 interligados por um *switch* com largura de banda de 40 MB/s *full-duplex* . Os nós não são todos iguais. Embora o desempenho de pico para operações em ponto flutuante seja de 266 MFLOPS para todos os nós, existem diferenças de tipo de processador, barramento, memória *cache* e memória RAM que podem ser mais ou menos significativas , dependendo das características do programa em execução.

### 7.3 Análise dos Resultados

Os *speedups* atingidos pelas três versões para os sistemas teste RTS, NBS e NNE estão mostrados nas tabelas 1. Os tempos de execução da versão sequencial do algoritmo para os três casos testes são 35,982 seg, 24,58 min e 17,58 min, respectivamente.

| Casos | Versão 1 | | | Versão 2 | | | Versão 3 | | |
|---|---|---|---|---|---|---|---|---|---|
| | p = 2 | p = 3 | p = 4 | p = 2 | p = 3 | p = 4 | p = 2 | p = 3 | p = 4 |
| RTS | 1,84 | 2,48 | 3,27 | 1,86 | 2,70 | 3,51 | 1,87 | 2,70 | 3,52 |
| NBS | 1,91 | 2,76 | 3,57 | 1,95 | 2,87 | 3,78 | 1,96 | 2,94 | 3,89 |
| NNE | 1,85 | 2,59 | 3,24 | 1,91 | 2,69 | 3,41 | 1,99 | 2,99 | 3,98 |

TABELA 1 - *Speedups*

A versão 3 sempre apresenta o melhor desempenho, independente do número de processadores e do tamanho e complexidade do sistema teste. Isso se deve ao fato da versão ser totalmente assíncrona, possuir balanceamento de carga ideal e tempo de sincronização desprezível. A versão 1 é a que apresenta desempenho inferior, como já esperado, devido ao maior tempo de sincronização envolvido.

Em todas as versões, os tempos de *broadcast* dos dados iniciais e agrupamento dos resultados finais são desprezíveis em relação ao tempo total da simulação. A quase

totalidade do tempo de computação se deve ao processamento das análises de contingências. As comunicações que têm efeito significativo no desempenho são as necessárias para verificação da convergência, variando com a versão e a complexidade do sistema sob análise. O tempo gasto nesta verificação é nitidamente maior na versão 1, reduzindo na versão 2 e atingindo um valor mínimo na versão 3.

A tabela 2 resume a eficiência alcançada pelas três versões para os três casos testes. Todas as versões apresentam boa eficiência, e em especial na versão 3, a eficiência é ótima. Em termos de escalabilidade do algoritmo em função do número de processadores (p), de novo a versão 3 apresenta melhor resultado, mantendo uma eficiência praticamente constante para os casos de maior porte.

| Casos | Versão 1 | | | Versão 2 | | | Versão 3 | | |
|---|---|---|---|---|---|---|---|---|---|
| | p = 2 | p = 3 | p = 4 | p = 2 | p = 3 | p = 4 | p = 2 | p = 3 | p = 4 |
| RTS | 91,98 | 82,56 | 81,68 | 93,23 | 90,07 | 87,84 | 93,68 | 90,07 | 88,07 |
| NBS | 95,50 | 92,15 | 89,38 | 97,65 | 95,64 | 94,47 | 98,10 | 97,90 | 97,28 |
| NNE | 92,59 | 86,33 | 80,96 | 95,44 | 89,91 | 85,28 | 99,75 | 99,63 | 99,44 |

TABELA 2 - Eficiência

## 8. CONCLUSÕES

Todas as versões paralelas necessitam de um ajuste fino do critério de controle da convergência global. Na versão sequencial, a cada nova contingência analisada, a convergência é verificada e tão logo a tolerância especificada seja satisfeita, o processo iterativo é interrompido.

Nas versões paralelas, a verificação da convergência é feita em lotes de contingências. Na versão 1, este lote é determinado pelo número de contingências e nas versões 2 e 3, por um intervalo de tempo de computação. O número de contingências analisadas quando a convergência global é detectada depende do tamanho destes lotes e não é exatamente aquele que ocorre na versão sequencial. Se o tamanho do lote for grande, existe a possibilidade de se ultrapassar em muito o número realmente necessário para atingir a convergência, o que resulta em computação desnecessária e redução do desempenho do algoritmo paralelo. Por outro lado, se for muito pequeno, a comunicação passa a ser mais frequente, o que também é ruim para o desempenho.

Outra consequência deste fato é que o problema paralelo não converge exatamente para o mesmo resultado do problema sequencial. A solução paralela é ainda mais precisa que a sequencial e tão válida quanto.

Em arquiteturas que executam mais que um job em modo *time sharing*, um desbalanceamento de carga adicional pode ser introduzido por tarefas de outros usuários executando concorrentemente nos mesmos processadores. Essa perturbação prejudica o desempenho das três versões implementadas, porém não inviabiliza a execução de nenhuma delas. Para fins de levantamento do desempenho do algoritmo, só deve ser considerado o tempo gasto com a computação em questão e o fato dos processadores

estarem carregados com outros processos deve ser encarado como uma redução no número de processadores disponíveis.

Quando a estratégia de controle da convergência é por intervalo de tempo de computação e o algoritmo é síncrono, o tempo máximo de sincronização em cada processador é o equivalente à análise de uma contingência por intervalo de tempo. O teste do momento de comunicar a convergência local é feito ao final de cada análise de contingência. Dependendo da carga em cada processador, pode ocorrer do intervalo de tempo se esgotar imediatamente após o teste em um processador, o que o obriga à execução da análise de mais outra contingência. Dessa forma, sistemas elétricos onde as contingências são de difícil análise podem apresentar tempos de sincronização elevados.

## REFERÊNCIAS


1. BILLINTON and R.N. ALLAN, *Reliability Assessment of Large Electric Power Systems*, Kluwer, Boston, 1988.
2. BILLINTON and W. LI, *Reliability Assessment of Electric Power Systems Using Monte Carlo Methods*, Plenum Press, New York, 1994.
3. PEREIRA and N.J. BALU, "Composite Generation/Transmission System Reliability Evaluation", *Proceedings of IEEE*, vol. 80, no. 4, pp. 470-491, April 1992.
4. *Sistema Computacional NH2 para Análise de Confiabilidade de Sistemas de Geração/ Transmissão de Grande Porte - Manual de Metodologia*, Relatório CEPEL no. DPP/POL-137/93, 1993.
5. MELO, J.C.O. MELLO, S.P. ROMÉRO, G.C. OLIVEIRA, R.N. FONTOURA F°, "Avaliação Probabilística do Desempenho do Sistema Interligado Brasileiro", *Anais do IV Simpósio de Especialistas em Planejamento da Operação e Expansão Elétrica*, 1994.
6. GUBBALA, C. SINGH, "Models and Considerations for Parallel Implementation of Monte Carlo Simulation Methods for Power System Reliability Evaluation", *IEEE Transactions on Power Systems*, Vol. 10, no. 2, pp. 779-787, May 1995.
7. FOX, M. JOHNSON, G. LYZENGA, S. OTTO, J. SALMON, D. WALKER, *Solving Problems on Concurrent Processors*, Vol.1, Prentice Hall, New Jersey, 1988.
8. CHAUDY, J. Misra, *Parallel Program Design: a Foundation*, Addison-Wesley, 1988.
9. BARBOSA, *An Intoduction to Distributed Algorithms*, The MIT Press, Cambridge, Massachusetts, 1996.
10. SNIR, S. OTTO, S. HUSS-LEDERMAN, D. WALKER, J. DONGARRA, *MPI: The Complete Reference*, The MIT Press, Cambrige, Massachusetts, 1996.
11. KITAJIMA, "Programação Paralela utilizando Mensagens", *Anais do XV Congresso da Sociedade Brasileira de Computação*, Canela, RS, Brasil, 1995.
12. IEEE APM Subcommittee, "IEEE Reliability Test System", *IEEE Transaction on Power Apparatus and Systems*, Vol. PAS-98, pp. 2047-2054, Nov/Dec 1979.
13. CIGRÉ Task Force 38-03-10, *Power System Reliability Analysis - Volume 2 - Composite Power Reliability Evaluation*, 1992.
14. *IBM POWERparallel Technology Briefing: Interconnection Technologies for High-Performance Computing (RS/6000 SP)*, RS/6000 Division, IBM Corporation, 1997.