UNIVERSIDADE ESTADUAL DE SANTA CRUZ

PAULO DA SILVA SILVEIRA

REESTRUTURAÇÃO E PARALELIZAÇÃO DO ESPAÇO DE DADOS E DOS MÉTODOS DE ENTRADA/SAÍDA DO SOFTWARE CIENTÍFICO DE ANTENNA CICLOTRÔNICA IÔNICA (ICANT).

ILHÉUS – BAHIA

2008

PAULO DA SILVA SILVEIRA

REESTRUTURAÇÃO E PARALELIZAÇÃO DO ESPAÇO DE DADOS E DOS MÉTODOS DE ENTRADA/SAÍDA DO SOFTWARE CIENTÍFICO DE ANTENNA CICLOTRÔNICA IÔNICA (ICANT).

Monografia apresentada, para a obtenção do título de bacharel em Ciência da Computação, à disciplina Estágio Supervisionado, à Universidade Estadual de Santa Cruz.

Área de concentração: Ciência da Computação.

Orientador: Prof. Gesil Sampaio Amarante Segundo.

ILHÉUS – BAHIA

2008

PAULO DA SILVA SILVEIRA

REESTRUTURAÇÃO E PARALELIZAÇÃO DO ESPAÇO DE DADOS E DOS MÉTODOS DE ENTRADA/SAÍDA DO SOFTWARE CIENTÍFICO DE ANTENNA CICLOTRÔNICA IÔNICA (ICANT).

Ilhéus-BA, 04/07/2008

_____

PhD. Gesil Sampaio Amarante Segundo
UESC/DCET
(Orientador)

_____

MSc. Flavio Shigeo Yamamoto
UESC/DCET

_____

PhD. Martha Ximena Torres Delgado
UESC/DCET

DEDICATÓRIA

Á minha família e amigos que me apoiaram e não mediram esforços para que eu chegasse até essa etapa de minha vida, assim como a meus professores e orientadores que sempre me incentivaram, dedico.

AGRADECIMENTOS

Aos meus professores da formação básica, pelo ensino e disciplina que me foi passado desde os tempos de infância.

Á Universidade Estadual de Santa Cruz (UESC) que através do Departamento de Ciências Exatas e Tecnológicas (DCET) me concedeu a oportunidade da realização do curso.

Ao Fundo de Amparo à Pesquisa do Estado da Bahia (Fapesb) que, por 3 vezes, concedeu-me bolsa de Iniciação Científica incentivando-me a contribuir com a sociedade científica da região.

Aos meus orientadores José Weyne Nunes Marcelino, Marta Magda Dorneles Bertoldi e Martha Ximena Torres Delgado que acreditaram e desenvolveram o meu potencial na pesquisa científica e me incentivaram a permanecer nessa área da Universidade.

Especialmente ao professor Doutor em Física de Plasmas Gesil Sampaio Amarante Segundo que foi meu professor e orientador em dois projetos de iniciação científica, além de orientar-me nesse projeto de fim de curso.

Aos amigos Rodrigo Tripodi Calumby e Cristianno Martins Vieira pelo companheirismo nas horas de descontração e de estudo, nas viagens para os diversos eventos e apoio nas disciplinas mais exigentes durante o curso.

Aos amigos Marcos Vinícios Vieira Souza e Fabio Luiz Lessa pelo companheirismo e ajuda nos momentos finais desta etapa extraordinária da minha vida.

Á minha família pelo imenso apoio, ajuda e cooperação. Especialmente ao meu pai.

E finalmente ao meu Deus por me ter dado o dom da vida e me ter permitido tudo o que tenho conseguido até este momento.

"Uma pessoa inteligente resolve um problema, um sábio o previne."

Albert Einstein

REESTRUTURAÇÃO E PARALELIZAÇÃO DO ESPAÇO DE DADOS E DOS MÉTODOS DE ENTRADA/SAÍDA DO SOFTWARE CIENTÍFICO DE ANTENNA CICLOTRÔNICA IÔNICA (ICANT).

RESUMO

O Plasma (gás ionizado, conhecido como o quarto estado da matéria) é o estado mais comum da matéria no universo, mas é relativamente raro na Terra. O desafio para obtenção de energia através da Fusão Termonuclear Controlada, para muitos a fonte de energia do futuro, depende do aquecimento de plasmas a temperaturas em torno de 10.000.000 Kelvin.

A otimização da deposição de energia em plasmas pelo aquecimento destes por ondas de radiofreqüência RF em máquinas chamadas *tokamaks* tem sido objeto de intensa pesquisa e modelagem numérica, sendo o programa de Antena Ciclotrônica Iônica ICANT um software científico criado especificamente para esse fim.

Os métodos numéricos e estratégias usadas nessa modelagem requerem muito poder de processamento e grandes espaços de memória para serem realizados, tornando-se um trabalho moroso que pode exigir poderosos recursos computacionais.

Neste trabalho, é apresentada a reestruturação e paralelização do código ICANT com enfoque principal na divisão e otimização das necessidades de memória (item diretamente proporcional a qualidade da modelagem e calculo dos elementos de antena) e na Entrada/Saída.

Palavras-chave: Física de Plasmas; Fortran; Message Passing Interface; Processamento Paralelo.

SUMARIO

1. INTRODUÇÃO

No atual estágio do desenvolvimento humano, a necessidade de grandes volumes de energia (ver Figura 1) tornou-se ponto crítico para a estabilidade econômico-financeira mundial, pois o principal modelo de energia de que depende o mundo hoje que é a energia petrolífera, segundo alguns estudos, poderá se esgotar em aproximadamente 50 anos (CRUZ, 2006) e vem a cada dia batendo recordes de preço, além de ser um dos principais agentes causadores da poluição e contaminação dos ecossistemas e do ambiente global, contribuindo em muito para o avanço do efeito estufa, um dos problemas que mais preocupam nossa sociedade.

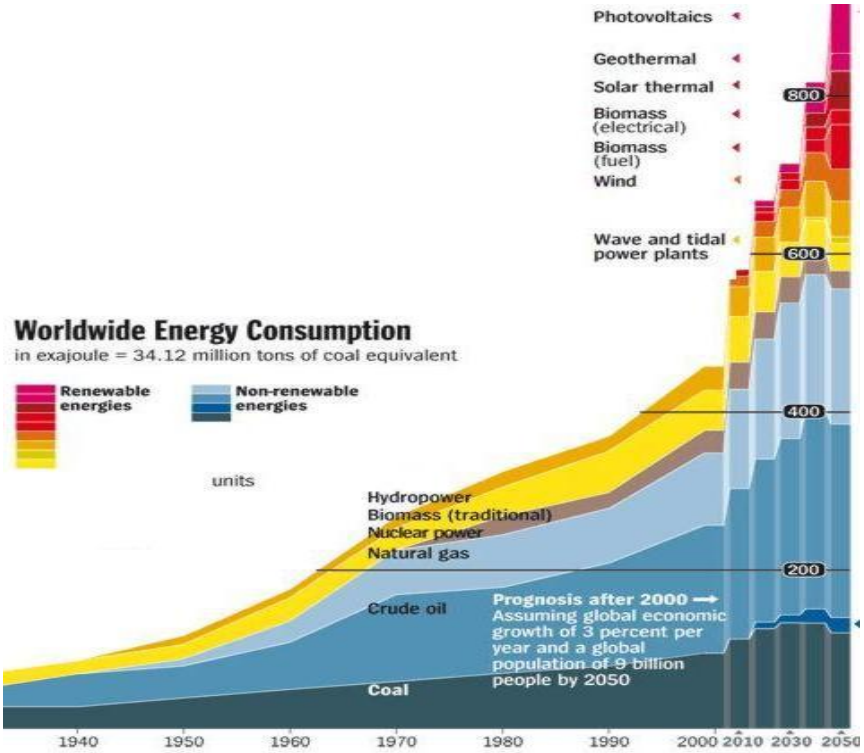

Figura 1. Consumo Global de Energia. Pode-se perceber a grande predominância da dependência e o crescente uso de energias fósseis (não renováveis) em relação às energias renováveis. Retirada de (FOLLATH, 2006).

Existem esforços em diversas direções para a futura substituição do petróleo como, por exemplo, as energias eólica, solar, hidroelétrica, termoelétrica, biomassa e outras, porém nenhuma ainda com possibilidades de ser uma alternativa factível, sendo a viabilidade econômica e em alguns casos tecnológica, as principais causas disso (YOUNGQUIST, 1999).

A busca por fontes de energia abundantes e não poluentes requer opções absolutamente inovadoras e a Física de Plasmas tem apresentado uma opção limpa e poderosa, a mesma que mantém as estrelas: a Fusão Termonuclear Controlada (FTC). Ainda são necessários esforços para vencer obstáculos para a viabilização econômica e tecnológica da FTC, que já teve sua viabilidade científica comprovada (LAWSON, 1985) e a sua comunidade de pesquisa tem feito grandes progressos, especialmente nas últimas duas décadas.

Esta pesquisa é feita principalmente através de equipamentos complexos chamados *tokamaks* (ver Figura 2) do acrônimo russo para Armadilha Magnética Toroidal, onde um plasma de núcleos leves (Hidrogênio ou seus isótopos Deutério e Trítio) é aquecido e estes colidem com energia suficiente para vencer a repulsão coulombiana e fundem-se em núcleos de Hélio. Neste processo, nêutrons e grandes quantidades de energia são liberados (GALVÃO, 2008).

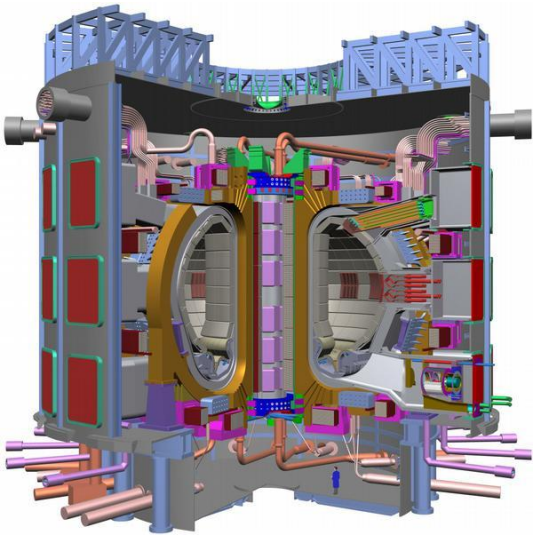

Figura 2. Modelagem 3D de como será o *tokamak* ITER. Em detalhe pode-se ver o desenho de uma pessoa (ponto azul na parte de baixo da figura) para se ter uma idéia das dimensões do ITER. Figura retirada de (ITER, 2008).

Nos *tokamaks*, correntes pulsadas geram campos magnéticos intensos dentro da câmara toroidal para manter o plasma confinado em uma configuração estável e aquecê-lo até a temperatura necessária para se obter a fusão. O maior *tokamak* existente é o *Joint European Torus* (JET), na Inglaterra, mas operado por equipes de cientistas de toda a Europa, da forma como deve ser operado o novo maior *tokamak* do mundo em construção na França, num empreendimento multinacional de 5 bilhões de Euros, o *International Thermonuclear Experimental Reactor* (ITER), (também “o caminho” em Latim) - http://www.iter.org/ -, que está destinado a provar

a viabilidade técnica da fusão como fonte de energia e deve ser concluído em 2016. Algumas partes do ITER estão ainda em projeto (GALVÃO, 2008).

Melhorias em várias tecnologias são necessárias, o que tem empregado intensa modelagem computacional. Existem vários códigos científicos voltados para a melhoria e otimização destes processos, onde se pode citar os códigos comerciais *Mafia*^{TM}, *Microwave Studio*® (CST MICROWAVE STUDIO, 2004) e os códigos acadêmicos *Topica* e o *Ion Cyclotron Antennas* ICANT (AMARANTE-SEGUNDO *et al.,* 2005). Este último tem significativa importância nesse tema, sendo usado na modelagem das estruturas de antenas de Radio Freqüência usadas para aquecer plasmas no interior de *tokamaks*.

O código ICANT, idealizado pelo Dr. Raymond Koch (KOCH; BHATNAGAR; MESSIAEN, 1986), foi escrito em FORTRAN 77 pelo Dr. Serge Pécoul na Universidade Henri Poincaré em Nancy, França em seu doutorado (PECOUL, 1998) para o modelamento tridimensional de antenas de RF (ver Figura 3) para aquecimento de plasmas em *tokamaks* via onda ciclotrônica iônica ICRH (SWANSON, 1989), sendo largamente utilizado e modificado pelo Dr. Gesil Amarante quando de seu período de pós-doutoramento no exterior (AMARANTE-SEGUNDO *et al.,* 2002).

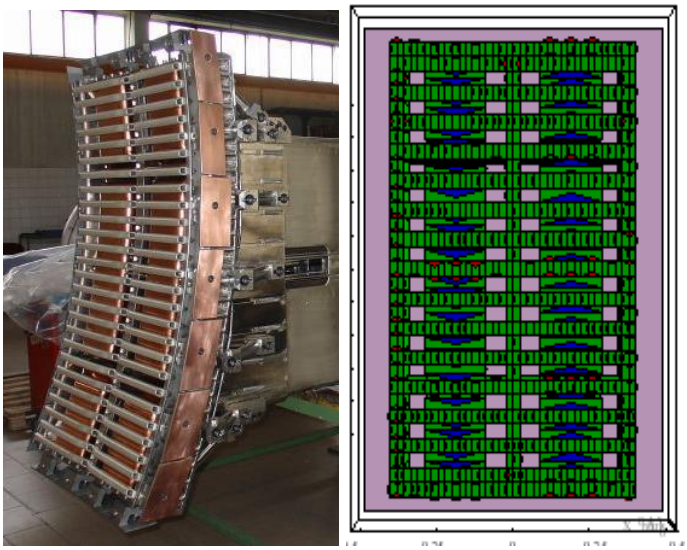

Figura 3. Modelo de antena real (á esquerda) e modelagem feita pelo ICANT da geometria e correntes da antena (á direita) gerada pelo software Matemática®. Em detalhe pode-se ver as direções e a intensidade de corrente representada pelas setas azuis e vermelhas. Retirado de (AMARANTE-SEGUNDO *et al.,* 2005).

O ICANT foi alvo também de otimizações e modernização em nível

algorítmico[1] e de legibilidade e modularidade realizado por (SOUZA; AMARANTE-SEGUNDO, 2007), tendo sua linguagem atualizada para Fortran 90, alocação dinâmica[2] implementada e tempo de cálculo reduzido em mais de 30% com a somatória das otimizações impetradas pelos mesmos. Algumas estruturas de dados foram reescritas e o modelo de compartilhamento de variáveis globais (entre o programa principal e as sub-rotinas) atualizado para a forma da linguagem Fortran 90 através da declaração de módulos.

O código o ICANT calcula as densidades de correntes para os elementos de um modelo tridimensional de uma antena (ver Figura 3) de radiofreqüência (RF), e sua vantagem principal é, ao mesmo tempo, fornecer as amplitudes das correntes nos elementos da antena (ver Figura 4), levando em conta a presença da coluna de plasma cujos parâmetros de densidade e de composição podem ser escolhidos, permitindo-se estimar a eficiência do acoplamento onda-plasma para diferentes desenhos de uma antena e seus regimes de operação.

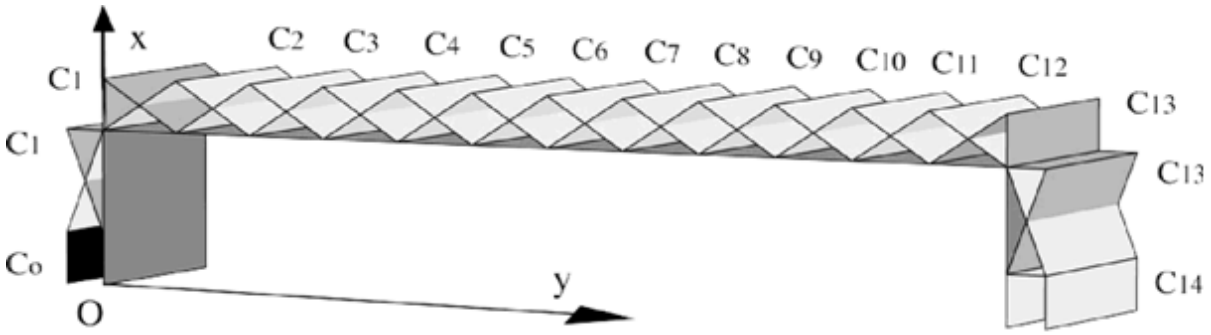

Figura 4: Elementos de corrente para antena simples com uma corrente unidimensional. O elemento ativo C_0 é mostrado em preto. Retirada de (AMARANTE-SEGUNDO, 2005).

[1] Reestruturação ou otimização das instruções de um algoritmo.
[2] Método de alocação onde o programa, no decorrer, de sua execução adquire, utiliza e libera espaços de memória por quantas necessitar. Esse método é interessante, pois permite menos desperdício e mais dinamismo no uso de memória.

1.1. Identificação do Problema

O código científico ICANT calcula as correntes dos elementos de antena por meio de um método matemático denominado Transformadas Inversas de Fourier[3], utilizando-se de elementos de antena arranjados em uma matriz que expressa a influência desses elementos entre si, levando em conta que alguns elementos têm seus valores fixados previamente (os elementos ativos, geralmente nos pontos de alimentação das antenas), e a grande maioria, os elementos passivos, terão suas correntes calculadas.

Estes cálculos, influenciados principalmente pela forma como foram programados sem maiores preocupações com legibilidade, portabilidade ou mesmo algum tipo de estratégia que primasse pelo desempenho, proporcionam uma natureza quadrática ao crescimento de tempo de solução do problema e exigem grande poder de processamento, demandando tempos de resposta excessivamente longos e tornando a obtenção alguns poucos resultados um trabalho de dias.

Quase todo tempo de execução é gasto compilando a matriz de impedância *R*, cuja inversão resulta num vetor de valores de densidade de corrente, através da expressão $P = \frac{I^* \cdot R \cdot I}{2}$, onde *P* é a energia total depositada no plasma, *R* é a matriz *N* por *N* e *I* o vetor das correntes dos elementos ativos (AMARANTE-SEGUNDO, 2005).

Outro ponto crítico está na necessidade de grandes espaços de memória, pois à medida que se necessita de uma modelagem mais detalhada que maximize a eficiência do acoplamento plasma-antena, cresce o número de elementos a serem calculados e, por conseguinte, aumenta-se quadraticamente o espaço de memória requerido pelo programa, pois, como já citado, é calculada a influência de cada elemento em todos os outros e estes são armazenados na matriz *R*, onde, se existem *N* elementos de corrente, ter-se-á uma matriz *N* por *N* armazenando números complexos de dupla precisão que ocupam 16 bytes cada. Ou seja, qualquer aumento no número de elementos implica em crescimento quadrático no

[3] Desenvolvida por Jean Fourier (1768-1830) em 1822, consiste na superposição de ondas de freqüências diferentes correspondentes, por exemplo, a diferentes velocidades do sangue. Em um sinal de ressonância nuclear magnética ou um sinal de radio, freqüência também consiste de diferentes freqüências e fases, que podem ser analisados diretamente por um espectro de freqüências, ou Transformada de Fourier.

uso dos recursos de memória.

Vale ressaltar que o limite de recursos de memória da maioria dos computadores (modelo PC) para cálculos científicos (normalmente 2 GB), já esta sendo atingido, implicando na impossibilidade da melhoria no detalhamento da modelagem das antenas (fato importantíssimo para a obtenção da deposição ótima de energia no plasma), pois mais elementos teriam que ser calculados e não haveria memória suficiente.

Por conseguinte a análise, reestruturação e paralelização do código ICANT torna-se imprescindível, especialmente da paralelização de memória, pois permitirá que modelos muito mais complexos e detalhados de antena sejam calculados sem maiores dificuldades e em tempos razoáveis através de máquinas com configuração modesta e de baixo custo, possibilitando assim melhorias significativas no processo da otimização da obtenção de energia pela Fusão Termonuclear Controlada.

1.2. Objetivos

1.2.1. Objetivos gerais

Este trabalho tem como finalidade a análise, a reestruturação de código e a paralelização de dados do software ICANT, mais especificamente em sua principal parte, chamada ICANTDO. Este visa também a proporcionar experiência no trato com programas computacionalmente exigentes e com o paradigma de processamento paralelo, requisitando o aprimoramento de estratégias de paralelização de softwares científicos para os mais diversos propósitos, finalizando-se através da compilação dessas estratégias em um manual.

1.2.2. Objetivos Específicos

- Identificação dos trechos críticos do código e as estruturas de dados mais exigentes em recursos de memória.
- Idealização das estratégias de paralelização para estes trechos e estruturas de dados.
- Implementação destas estratégias.
- Avaliação de desempenho, em clusters *beouwlf*[4].
- Aplicação da paralelização de Entrada/Saída no código.
- Documentação das estratégias adotadas e da montagem do ambiente de desenvolvimento.

[4] *Beowulf* é o nome de um projeto para aglomerados de computadores (ou *Clusters*) para computação paralela, usando computadores pessoais, não especializados e portanto mais baratos. O projeto foi criado por Donald Becker da NASA, e hoje são usados em todo mundo, principalmente para processamento de dados com finalidade científica. Definição retirada de http://pt.wikipedia.org/wiki/Aglomerado_Beowulf.

2. METODOLOGIA

No início foi realizada uma rápida revisão do código que seria alvo da paralelização, pois já havia conhecimento prévio das principais partes deste advindo de projeto de iniciação científica que em período anterior aplicou paralelização de controle no referido código (SILVEIRA *et al.,* 2007). Essa revisão consistiu também da análise de informações de tempos de execução da versão com paralelização de controle e da versão serial, informações que foram obtidas através da execução em clusters *beouwlf* e mostraram a real necessidade da paralelização de dados para o processamento de modelagens mais detalhadas da antena.

Com o *profilling*[5] do código, teve-se a oportunidade de observar o comportamento do programa durante todo o seu tempo de execução, permitindo também a tomada de decisões sobre as estratégias de paralelização e a obtenção de informações sobre as estruturas mais custosas em relação ao uso de processamento (ver Figura 5) e memória.

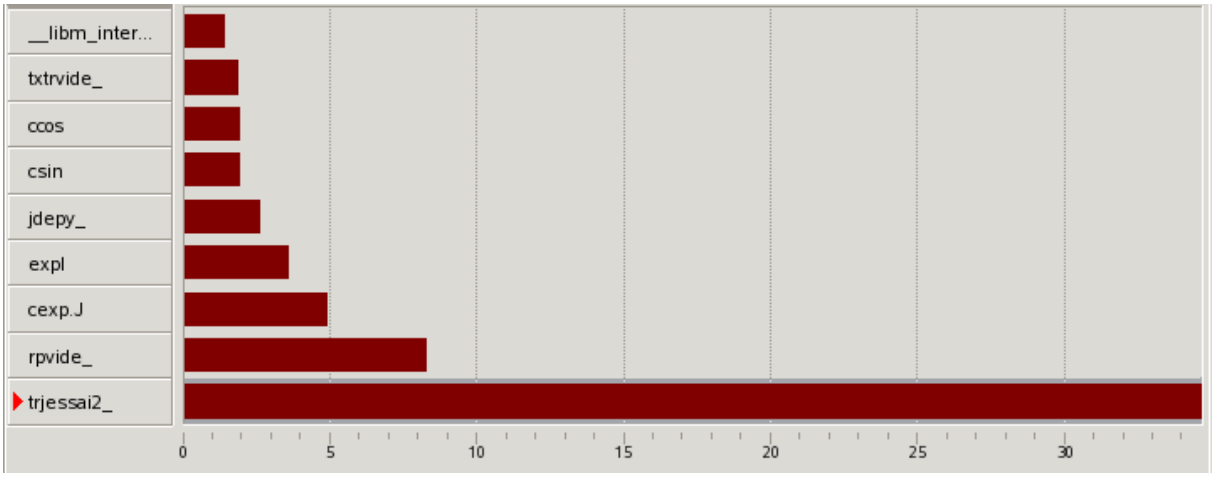

Figura 5. Profilling do código ICANT realizado com o software Intel® VTune™ Performance Analyzer em sua versão para a plataforma Linux. Sendo qeu a barra vermelha representa a participação de cada sub-rotina na composição do tempo total de execução, pode-se observar que a que mais consome recursos de processamento é a TRJessai2. Análise produzida por (SOUZA, 2007).

O ambiente montado para o desenvolvimento contou com ferramentas como

[5] Processo onde se estuda o comportamento de um programa tanto em relação ao uso de memória quando ao processamento que é exigido.

o *Palarell Tools Plataform* em cojunto com o *Parallel Language Development Tools* (PTP/PLDT), que são *plugins* para o *Integrated Development Environment* (IDE*) Eclipse PHOTRAN* (ver Figura 6) especialmente arquitetados para o desenvolvimento, *debugging*[6] e *profilling* de aplicações paralelas, possibilitando um desenvolvimento eficiente, facilitando muito a resolução de problemas relacionados à programação paralela. A utilização deste conjunto de ferramentas facilitou muito e reduziu drasticamente o tempo de desenvolvimento e testes dos algoritmos implementados.

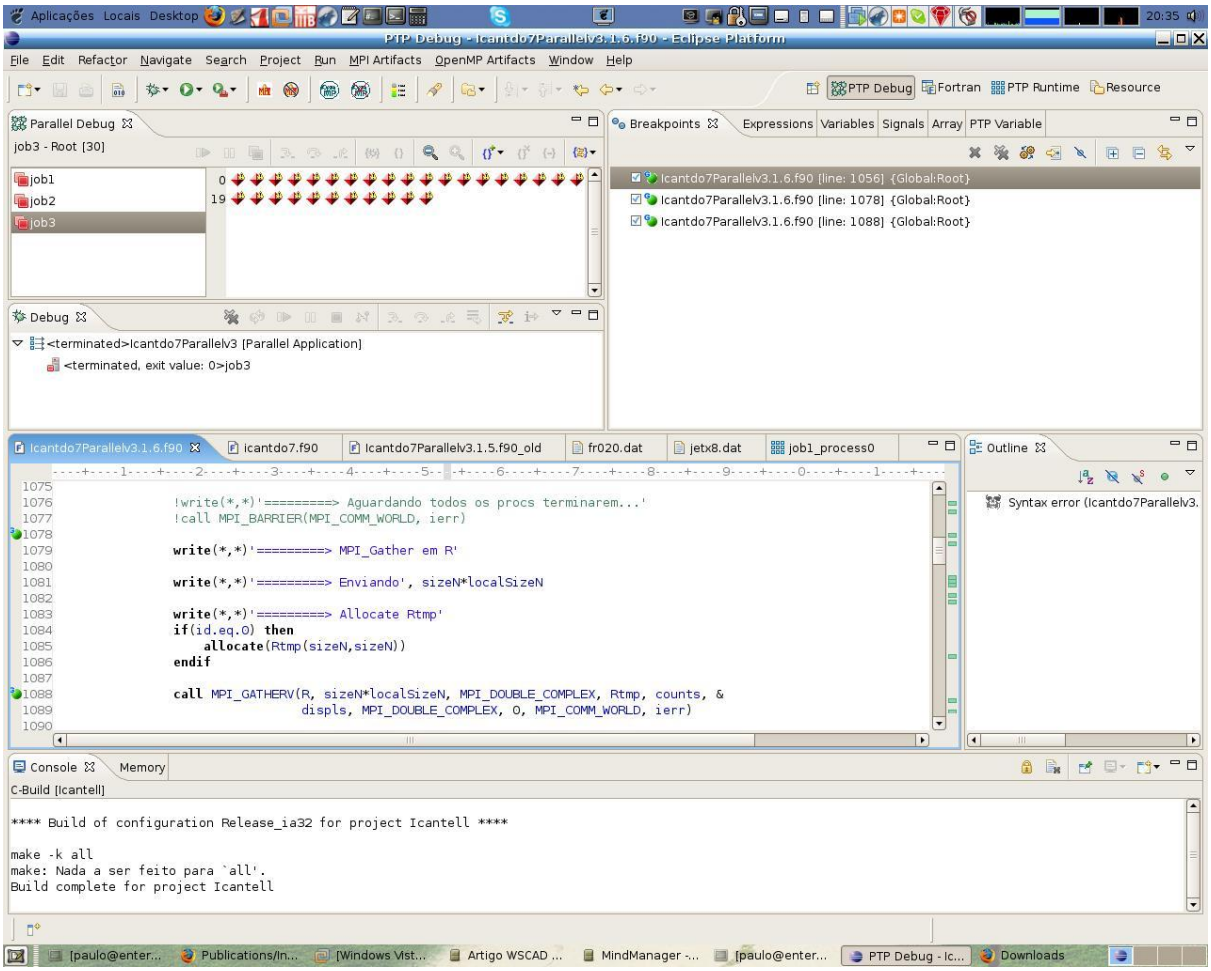

Figura 6. Janela típica do *plugin* PTP/PLDT no eclipse, mostrando a *view* "PTP Debug" que é responsável pelo debug de aplicações paralelas. No lado superior esquerdo tem-se os Jobs executados, ao lado, a representação dos nós que foram executados. No lado superior direito tem-se os *breakpoints*[7], acima botões para acesso a outras *views.* No meio o editor com o *highlighting* da linguagem Fortran 90 e abaixo o console que exibe informações de status de compilação e outras operações.

[6] Técnica em que, através de um programa especial chamado debugger, é possível a descoberta de problemas na execução do algoritmo de outro programa em tempo de execução, além do monitoramento das variáveis deste.

[7] Pontos do código onde deseja-se que a execução do programa seja interrompida para análise em tempo de execução.

Este projeto teve a sua disposição 2 máquinas Core 2 Duo com 2 gigabytes de memória RAM e disco de 300 gigabytes adquiridas com recursos do projeto PRODOC do orientador deste trabalho (AMARANTE-SEGUNDO, 2005) financiado com recursos da Fapesb, com Debian Linux 4.0 r1 instalado. Foi viabilizado também o acesso a clusters *beowulf* (ver Figura 7) como o cluster localizado no NBCGIB gerenciado pela Professora Martha Ximena, constituído de 5 nós com processador Intel Xeon 3.2 gigahertz *Hiper-Threading*, 2 gigabytes de memória RAM com rede Gigabit Ethernet e Debian Linux instalado e o cluster do CENAPAD-SP localizado na UNICAMP que possui 28 nós bi-processados AMD-Opteron 242, 1.6 gigahertz e 2 gigabytes de memória RAM com Gigabit Ethernet e Fedora Linux.

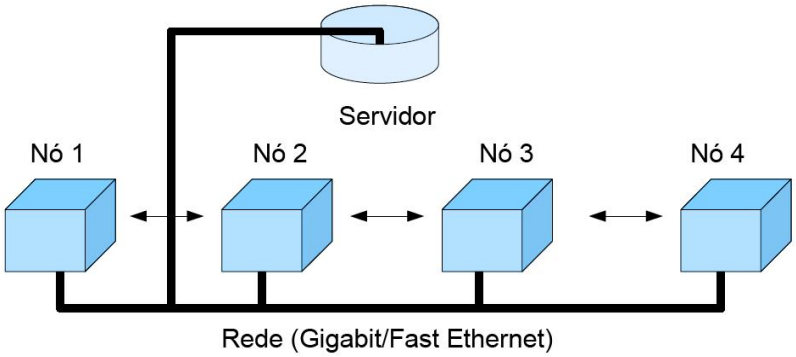

Figura 7. Exemplo de um cluster com 4 nós de processamento e um servidor interligados por tecnologia de redes de computadores que, entre outras, podem ser Gigabit Ethernet (1000 megabits por segundo) ou Fast Ethernet (100 Megabits por segundo).

Foi adotado o Message Passing Interface (MPI) como padrão de paralelização nas implementações MPICH1, utilizada no cluster do CENAPAD-SP, MPICH2, utilizada no cluster do NBCGIB, e OpenMPI utilizada principalmente em conjunto com o IDE Eclipse e os *plugins* PTP/PLDT. Ferramentas Linux como o ssh/putty (para conexão remota), scp (transferência de arquivos entre sistemas remotos), pico/nano (edição de arquivos texto), nohup (execução de programas remotos) foram essenciais para o desenvolvimento do trabalho remotamente.

Como compilador fortran foi escolhido o *Intel Fortran Compiler* for Linux (IFORT), pois este tinha várias otimizações diretamente ligadas ao tipo de processador dos computadores do cluster dó NBCGIB que são *Intel Xeon* e das máquinas de desenvolvimento que são *Intel Core2 Duo*. Foi também verificado que o tempo de execução de programas compilados pelo IFORT era cerca de 30% menor que o tempo de execução dos mesmos compilados por compiladores como o *Gfortran*, por exemplo.

2.1. Revisão de literatura

Um número crescente de códigos numéricos tem sido utilizado para o planejamento ou para se tentar explicar resultados das experiências em laboratórios de física. A comunidade de física de plasmas já tem uma certa tradição na utilização de códigos pesados e que requerem máquinas extremamente rápidas, freqüentemente multiprocessadas. Em casos, máquinas com centenas de processadores são usados na simulação de instabilidades e fenômenos de transporte em plasmas para a pesquisa de fusão nuclear.

Entre os vários tipos de códigos utilizados na área situam se alguns grandes grupos:

Códigos de Ondas Completas: O Aquecimento de plasmas por meio de ondas de RF (rádio-freqüência) pode ser visto de dois pontos de vista complementares. Do ponto de vista do plasma, é um processo de absorção da onda. Descobrir como a onda se comporta requer resolver a equação de onda pertinente,

$$\nabla \times \nabla \times \bar{E} / k_0^2 = \bar{\bar{K}}.\bar{E} + i\bar{J}_a / \omega\varepsilon_o = \bar{E} + i\{\bar{J}_a + \bar{J}_p\} / \omega\varepsilon_o .$$

Aqui, $\bar{E}$ é o campo elétrico, $k_o = \omega / c$, $\bar{\bar{K}}$ é o tensor dielétrico, □ é a freqüência da onda, $\bar{J}_a$ w $\bar{J}_p$ são as correntes da antena e do plasma.

Códigos Focker-Planck: Do ponto de vista das partículas, o aquecimento do plasma é o processo de estar sendo acelerado ou desacelerado por um campo elétrico. A difusão resultante das partículas é descrita pela Equação de Fokker-Planck,

$$\frac{\partial F_o}{\partial t} = Q + C + S + L ,$$

em que F_o é a função de distribuição, Q é o termo de difusão quase-linear de RF, C representa o efeitodas colisões de Coulomb, S e L são os termos de fonte e de sumidouro.

Desde que essas duas equações acima descrevem 2 aspectos do mesmo processo físico, deveriam ser resolvidos como um sistema acoplado de equações. Por causa da complexidade envolvida elas são, todavia, resolvidas separadamente na maioria das aplicações.

Códigos Monte Carlo e Códigos de Partículas: Uma integração num espaço

multidimensional pode ser feita eficientemente usando a técnica de Monte Carlo, que – diferente de usar uma grade regular – usa um conjunto aleatório de pontos uniformemente distribuídos. Adotando este procedimento, a integral de uma função no hiper espaço é predita com um erro associado da ordem de $N^{-1/2}$, onde N é o numero de posições geradas aleatoriamente, independente do número de dimensões (enquanto a precisão de uma predição feita numa grade uniforme varia com $1/N^{1/d}$, onde d é o número de dimensões).

2.1.1. As bases do código ICANT

O ICANT baseia-se no método de elementos de fronteira (KOCH; BHATNAGAR; MESSIAEN, 1986) e tem como finalidade o cálculo das características físicas de uma antena de ICRH (ion-cyclotron Ressonance Heating - Aquecimento por ressonância cyclotrônica iônica) descrita de forma a mais próxima da real possível. O código permite um cálculo das quantidades tais como a distribuição das correntes em toda a estrutura da antena, a constante de propagação para cada emissor (*strap*), a potência espectral[8], a potência radiada[9], a reatância[10], a matriz de acoplamento de vários *straps*, o Vetor de Poynting[11] em todos os pontos, *E*, *B*, etc.

A estrutura tridimensional da antena é aproximada por um conjunto de superfícies retangulares condutoras perfeitas[12] para as diferentes componentes da antena ou todos os elementos metálicos que podem influenciar o acoplamento antena-plasma. Cada superfície retangular está associada a um elemento de corrente em uma determinada direção. A constituição do modelo da antena é realizada através da parte do código chamado ICANTELL.

---

[8] Potência espectral é o gráfico que mostra qual a fração da potência dissipada que o foi através da combinação dos números de onda *ky e kz.*

[9] Potência é energia por unidade de tempo. Potência irradiada é a quantidade de energia depositada pela antena no plasma através das ondas de RF por unidade de tempo.

[10] Reatância elétrica é a oposição oferecida à passagem de corrente alternada por indutância ou capacitância num circuito. É dada em *Ohms* e constitui a componente da impedância de um circuito que não é devida à resistência pura. Imaginando o sistema antena-onda-plasma como um circuito, a parte resistiva da impedância representaria a deposição de energia no plasma.

[11] Vetor de Poyinting é o vetor que dá a potência irradiada pela onda na direção de propagação desta.

[12] Superfícies condutoras perfeitas são superfícies que não oferecem qualquer resistência à condução de corrente. Numa primeira aproximação ideal, superfícies metálicas pode ser tratadas como condutoras perfeitas.

O plasma é descrito através de uma matriz de impedância de superfície, o que exige um modelo plano do plasma. Este método permite, em princípio, levar em conta o acoplamento da antena com todos os modos de plasma (ondas rápidas, ondas lentas, modos de transporte de Bernstein, etc) para todos os tipos de plasma (com densidade em níveis abruptos – *step plasma*, plasma não homogêneo com uma ou várias espécies de íons). Uma vez definido o sistema antena-plasma, alguns estudos paramétricos podem ser feitos com ICANTDO.

O processo de modelagem do ICANT depende das geometrias da antena e do *tokamak*, do número de elementos com o qual o modelo da antena é construído, das propriedades do plasma como a composição, perfis de densidade e temperatura (entre outros fatores) e é extremamente complicado de operar. Deve-se ter certeza de que as ondas irão eventualmente encontrar uma zona de ressonância dentro do plasma, que irá absorver boa parte da energia carregada pelas mesmas (alguns megawatts em grandes *tokamaks*). O cálculo pode ser custoso em modelos detalhados de antena, especialmente porque em alguns casos ele deve ser feito várias vezes para diferentes freqüências e configurações do plasma, o que pode levar dias e justifica o uso de paralelização.

O ICANT calcula a densidade das correntes através da transformada inversa de Fourier dos elementos de corrente, organizados em uma matriz (*R*) que expressa a influência eletromagnética (impedância) de cada elemento em todos os outros, levando em conta que alguns elementos têm seus valores de corrente previamente fixados (são os elementos ativos, geralmente nos pontos de alimentação das antenas), onde a grande maioria, os elementos passivos, vão ter suas correntes calculadas pelo programa. Uma das características mais importantes do ICANT é que ele leva em conta a configuração do plasma e resolve de maneira auto consistente os problemas acoplados onda-plasma e antena-onda, isso também é feito pelo TOPICA (LANCELLOTTI *et al.,* 2006). Existem diversos códigos eletromagnéticos comerciais, mas eles não podem incluir o plasma corretamente e, portanto, nem calcular a quantidade de energia acoplada ao mesmo.

A corrente de superfície nos condutores *j(r)* é representada pela sobreposição de elementos

$$\mathbf{j}(\mathbf{r}) = \sum_{m=1}^{N} a_m \mathbf{T}_m(\mathbf{r}) \quad ,$$

com os elementos de corrente *Tm(r)* dos elementos correntes a_m coeficientes complexos.

A condição de auto consistência é uma versão fraca da condição de nulidade do campo elétrico tangencial na superfície de condutores ideais

$$\int_{V_A} \mathbf{T}_m^*(\mathbf{r}).\mathbf{E}(\mathbf{r})\, dV = 0, \quad (l = 1, \ldots, N_p)$$

e isso implica que a energia integral calculada pelo ICANT se limita à parte gerada pelos elementos ativos:

$$P = -\frac{1}{2}\int_{V_A} \mathbf{j}^* .\mathbf{E}\, dV = -\frac{1}{2}\sum_{m=1}^{N_a} a_m \int_{V_A} \mathbf{T}_m^*(\mathbf{r}).\mathbf{E}(\mathbf{r})\, dV \qquad (1)$$

O problema eletromagnético resolvido na região do vácuo é combinado com uma matriz de impedância da superfície calculada por um código de ondas completas como uma condição da interface vácuo-plasma. O simples vácuo também pode ser utilizado.

A maior parte do tempo de execução é gasto na construção da matriz R, cuja inversão dá origem aos valores da matriz de densidade da corrente, através do equivalente simplificado da expressão (1)

$$P = \frac{I^* \cdot R \cdot I}{2}$$

já vista no segundo parágrafo da seção “Identificação do Problema” onde P é a energia total depositada no plasma (energia em MW).

Para um modelo de antena de *N*-elementos, uma matriz *R* de *N*×*N* elementos deve ser construída. O cálculo dos elementos de *R* é feito através da transformada inversa de Fourier dos elementos, de forma que têm-se um somatório do espaço de comprimento de ondas para as direções poloidal (*y*) e toroidal (*z*) nas quais os *tokamaks* são periódicos (ver Figura 8), portanto cada elemento de matriz é construído com $Nk_z \times Nk_y$ componentes da integração discreta de Fourier. Na direção radial (*x*), as equações são resolvidas usando-se funções de Green.

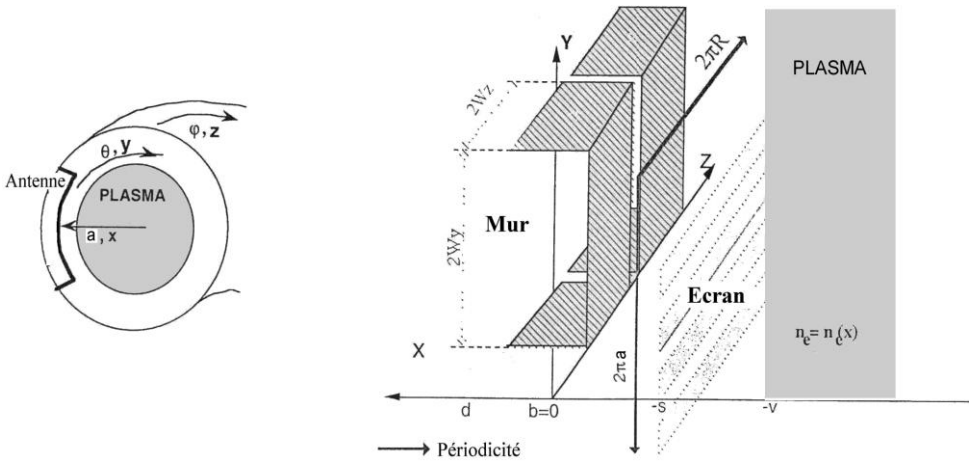

Figura 8. Disposição da antena e das coordenadas (eixo x, y e z). Fonte: (PECOUL, 1998).

O tamanho do *tokamak* e a menor dimensão de um elemento de corrente em cada uma dessas direções definem o número de elementos k_y (Nk_y) e k_y (Nk_y) necessários para atingir uma boa representação. É comum haver algumas centenas de componentes em cada direção.

No modelo de exemplo mostrado na Figura 9, para um tokamak como o JET, deve-se trabalhar com valores de 220 k_z (isto é, a integração vai de -220 à 220) e 110 em k_y (de -110 à 110). O código calcula cada um dos $N \times N$ elementos de matriz $(2k_y+1)\times(2kz+1)$ vezes. Neste exemplo, cada um dos 200704 elementos da matriz R é calculado com 97461 componentes de espectro. Além disso, a maioria dos coeficientes são números complexos de dupla precisão, o que significa, para um modelo razoável de antena, um cálculo longo e com alto consumo de memória.

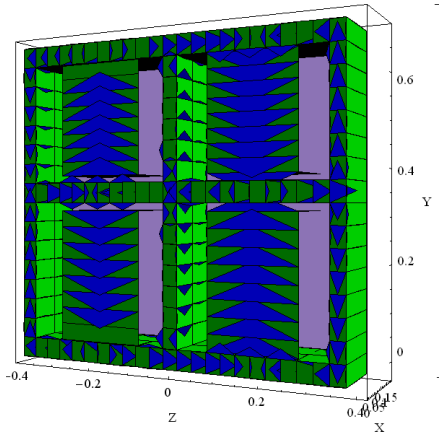

Figura 9. Modelo de antena de 448 elementos mostrando as densidades de correntes calculadas em setas azuis. Retirada de (SILVEIRA *et al.,* 2008).

2.1.2. ICANT: Funcionamento e uso

A execução do programa ICANT, necessita a priori dos arquivos *fr020.dat*, *fr021.dat*, *fr029.dat* e *fr030.dat* que contém as informações do conjunto *tokamak*–antena–plasma e que são submetidos ao módulo ICANTELL para a geração dos arquivos *fr013.dat*, *fr014.dat* (informações sobre os pontos/coordenadas de cada elemento da antena), *fr022.dat* e *xtrutcs.dat* (informações sobre os elementos das estruturas radiais da antena) que conterão informações mais detalhadas e específicas, sendo que os arquivos anteriores mais estes formam o conjunto de arquivos que constituem a entrada do módulo ICANTDO. Através desses arquivos é possível recriar, graficamente, as estruturas e uma antena e outras informações como densidade e direção das correntes nos elementos de antena como já se pode observar nas Figuras 3 e 8.

Durante a execução, o programa lê os arquivos que contém as informações sobre o modelo da antena, do *tokamak* e das características do plasma, organiza essas informações em vetores, estruturas e matrizes, sendo neste ponto que ele aloca a matriz *R* (*N*×*N* onde N é o número de elementos) inicializada com zeros. É chamada a sub-rotina ***TRJessai*** que calcula os elementos de *R* para uma pequena faixa do espaço de modos em k_y e k_z que vai de -5 a 5. Neste ponto são eliminadas as linhas da matriz de alguns elementos componentes das várias estruturas radiais que constituem um antena pela rotina ***IxElim***.

Agora que se tem uma matriz menor, é realizada a busca por simetrias físicas no modelo da antena, expressas por repetições na matriz, onde $R(e_1, e_2)$ será simétrico de $R(e_3, e_4)$ se sua diferença for inferior a um limite de tolerância (elementos simétricos não precisam ser calculados, economizando tempo de processamento). Em uma matriz auxiliar *TS*(*N*×*N*×*3)* serão marcadas todas as ocorrências de simetrias assim como um mapa de referências cruzadas de simetria, tarefa realizada pela sub-rotina ***FindSym***. Terminado este ponto, se tem uma matriz com cerca de 10 a 5% de itens $R(e_1, e_2)$ que serão efetivamente calculados em toda a faixa do espaço de modos em k_y e k_z que, no exemplo cita acima, vai de -220 a 220, pela sub-rotina ***TRJessai2***. Após o cálculo completo da matriz *R* são recolocados os elementos marcados como simétricos que não foram calculados utilizando-se a informação da matriz *TS*.

Uma vez que a matriz é calculada, serão repostas também as linhas dos elementos das estruturas radias pela sub-rotina ***IxBac***. Neste ponto são eliminados os elementos que têm continuidade[13] com o próximo (ex. se e_1 tem continuidade com e_2, então e_1 recebe a soma dele com e_2 e e_2 é eliminado temporariamente da matriz) pela sub-rotina ***Continui.*** Como foi diminuída novamente (linhas e colunas foram retiradas por *Continui*), os elementos ativos precisam ser reorganizados na matriz, o que é feito pela sub-rotina ***Separati***.

Terminado esse processo, inicia-se a parte do cálculo dos vetores de corrente. Antes, porém a matriz *R* precisa ser invertida pela sub-rotina ***Inverse***. Finalmente são calculados os vetores de corrente e é chamada a sub-rotina ***Puissanc*** que faz o cálculo final da potencia dissipada pela antena no plasma,

$$P = \frac{I^* \cdot R \cdot I}{2}$$

e quantidades a esta relacionadas.

Vale ressaltar que cópias de *R* e *TS* são escritas em disco dado que para outras execuções essas informações podem ser aproveitadas e um tempo muito grande será poupado.

Esse processo esta resumido pelas etapas exibidas na Figura 10.

[13] A Continuidade assegura que o valor da densidade de corrente no elemento i+1 será igual a do elemento i. Este elemento i+1 não precisará, portanto, ser calculado.

Figura 10. Representação do fluxo de processamento e das principais sub-rotinas executadas pelo módulo ICANTDO. As bandeirinhas verdes indicam os trechos do código que foram paralelizados neste trabalho, as em amarelo foram parcialmente paralelizadas, e as bandeirinhas vermelhas indicam os trechos que não foram, o relógio indica as sub-rotinas mais custosas em processamento, sendo realçada por tracejado a sub-rotina que abrange cerca de 95% do tempo de processamento de todo o programa. As setas tracejadas indicam que as sub-rotinas posteriores realizam operações diretamente ligadas as anteriores.

A compilação do código ICANT pode ser feita com os seguintes comandos utilizando o IFORT:

$ ifort -O3 -ipo -prefetch -o icantell icantell_x.for braffa3.for divpag.for nagimsl.for
$ ifort -O3 -ipo -prefetch -o Icantdo7 Icantdo7.f90

A execução do mesmo, verificada a presença dos arquivos necessários (citados no início desta seção) no mesmo diretório do executável, pode ser feita através dos comandos:

$ ./icantell
$ ./Icantdo7

Durante a execução várias informações são exibidas na saída padrão à medida que o processo avança. Informações como em qual rotina se encontra a execução no momento, quantos elementos de corrente realmente serão calculados e quantos foram marcados como simetrias, quantos tinham continuidade com o próximo são úteis para o acompanhamento da execução e posterior resolução de problemas às vezes triviais, mas que podem se tornar difíceis de serem descobertos e eliminados.

Ao final da execução do segundo comando existirão os seguintes arquivos: fr001.dat (valores que caracterizam a antena como a somatória das correntes dos elementos, potência dissipada no plasma, etc.), *fr002.dat*, *fr003.dat*, *fr004.dat*, *fr100.dat* (quantidades escalares[14]), *fr016.dat*, *fr017.dat* (potência espectral), e na saída padrão algumas informações que resumem os resultados (basicamente as informações constantes no arquivo fr001.dat são replicadas na tela).

2.2. A biblioteca de paralelização adotada

O *Message Passing Interface* (MPI Standard 1.1, 1995) é um padrão internacional bem consolidado que fornece um rico conjunto de rotinas otimizadas para suporte a escrita paralela de programas baseada na passagem de mensagens e dispõe de várias implementações de acordo com a plataforma hardware para as quais são direcionadas. Como exemplo se pode citar a implementação MPICH que é específica para clusters *beowulf* (GROPP; LUSK; SKJELLUM,1999).

No funcionamento do MPI, a paralelização se dá através da criação de várias cópias de um mesmo processo, sendo que estes passam a executar paralelamente

[14] Quantidades numéricas da antena como somatórias de correntes e outros números.

seja em processadores distintos ou no mesmo, cabendo ao programador definir os pontos de comunicação, a troca de mensagens (dados) e sincronização entre estes processos através de chamadas às respectivas sub-rotinas constantes no padrão. Essas solicitações são efetivadas através do envio de mensagens pela rede, para o caso de memória distribuída, ou pelo barramento do sistema (sistemas de memória compartilhada).

No processo de envio de mensagens, os dados (bytes) são colocados em buffers gerenciados pelo *daemon*[15] do MPICH para depois serem colocados no espaço de memória de outro processo em cada nó de processamento.

À medida que o programa paralelo em execução solicite a entrega ou recebimento destas mensagens, esse *daemon* envia as pela rede e coloca na memória do nó de destino do programa paralelo em execução. Existem mensagens que devem ser entregues no mesmo momento do envio (o nó que recebe fica em espera até que a comunicação seja efetivada), e mensagens que são enviadas para entrega posterior (o nó que envia prossegue sua execução). Neste trabalho foi usada comunicação bloqueante[16], sendo que os principais tipos de dados trafegados foram o tipo inteiro de 4 bits, referenciado pela constante MPI_INTEGER do padrão MPI, o inteiro de 2 bits referenciado pela constante personalizada[17] MPI_INTEGER_2 e o tipo complexo de dupla precisão ou 16 bits, tipo muito utilizado principais estruturas de dados do programa, referenciado pelo padrão MPI como MPI_DOUBLE_COMPLEX (MPI Standard 2.0, 1997).

[15] Processo que executa em background, realizando tarefas de coordenação e comunicação os processos do MPI dispersos pelos vários nós de execução.

[16] Comunicação onde os processos envolvidos ficaram bloqueados ate que esta finalize (processo receptor tenha recebido).

[17] O MPI permite a criação de tipos de dados personalizados. No Fortran é permitido se declarar variáveis especificando a quantidade de bytes que esta usará, o que prima pela economia de memória.

3. DESENVOLVIMENTO

O desenvolvimento da paralelização de memória no ICANT foi influenciado pelas experiências anteriores em paralelização de outros códigos e algoritmos e da paralelização de controle que foi realizada no mesmo (SILVEIRA *et al.,* 2007).

Um dos principais motivos do ganho que se obteve em qualidade e eficiência no desenvolvimento dessa paralelização mais complexa do ICANT, foi a utilização do ambiente montado com a IDE Eclipse e os *plugins* para desenvolvimento paralelo, pois houve a automatização de várias tarefas da programação dos algoritmos como a facilitação da navegação no código (existem algoritmos que realizam algo numa sub-rotina para depois desfazerem em outra), a visualização mais agradável do código (*high lighting* configurável da gramática da linguagem), compilação automatizada através da criação gerenciada de *makefiles*[18] onde os processos de inclusão de bibliotecas estáticas e inclusão de arquivos de cabeçalho[19] já estão predefinidos, uma vez configurados – fato importante pois o ambiente necessitava das bibliotecas e dos cabeçalhos da versão do MPI sendo utilizada (mpich2 por exemplo), do compilador de Fortran em uso e dos *plugins* do eclipse supra citados.

A manipulação de arquivos fonte e de arquivos de entrada/saída foi facilitada, tornando o processo de versionamento fácil (caso se quisesse retornar para uma versão anterior, bastaria renomear essa versão para a extensão f90 que seria automaticamente visto como o arquivo fonte atual). O versionamento foi feito à medida que o processo de paralelização progredia, sendo criadas versões para cada sub-rotina paralelizada ou cada passo considerável de paralelização que resultasse em uma versão paralela estável e funcional do programa ICANT.

Um ponto que merece destaque na utilização do ambiente Eclipse PTP/PLDT esta na divisão da saída padrão[20] de cada nó individualmente. Quando um programa serial grande e complexo (executa várias rotinas que precisam ser acompanhadas

[18] Arquivo que define as relações ou dependências entre aplicações funções, simplificando o processo de desenvolvimento pela realização automática de tarefas necessárias para a recompilação (rebuild) de uma aplicação quando você modifica o código.

[19] Arquivos onde se encontram definidas constantes e funções de uma biblioteca ou programa. A importação destes arquivos possibilita o uso, por outros programas, de funções disponibilizadas na biblioteca.

[20] Arquivo que representa para onde os comandos enviarão sua saída que normalmente é a tela do sistema. Quando o sistema escreve para o arquivo de saída padrão, a saída é coloca na tela, a menos que seja redirecionada para um arquivo específico.

durante a execução) que foi paralelizado é executado para vários nós através de linha de comando (*$ mpirun –n 15* por exemplo) a saída de cada nó é lançada na saída do nó zero o que mistura as saídas e dificulta a resolução de possíveis problemas relativos a nós específicos. Com a divisão das saídas proporcionada pelo ambiente de desenvolvimento a solução de problemas é maximizada, pois se tem saídas claras e individualizadas para cada nó.

Na pratica, a rotina de paralelização se deu da forma explicitada na Tabela 1 e apresentou bons resultados com relação ao tamanho da paralelização a ser implementada.

Tabela 1. Passos utilizados no desenvolvimento do paralelização do programa ICANT.

Etapa	Realização
1	Decidia-se o próximo bloco/sub-rotina a ser paralelizado, ou seja, ter seu algoritmo adaptado para lidar com a estrutura de memória de forma paralela, (pois as estruturas estavam igualitariamente dividias entre o número de nós).
2	Era feita uma pequena revisão de literatura relacionada com a sub-rotina em questão e perguntas como qual o propósito, como atua nas estruturas de memória eram respondidas para cada sub-rotina sob paralelização.
3	Seu algoritmo era estudado com foco principal nas partes que manipulavam as estruturas que foram eleitas para serem dividas/paralelizadas (tema do próximo tópico).
4	Uma estratégia de paralelização era idealizada para aquele caso que fosse compatível com as estratégias e algoritmos paralelos já implementados em sub-rotinas anteriores que estavam comprovadamente funcionando.
5	Era realizada a implementação, sendo após testada localmente (vários processos numa mesma máquina) para casos pequenos até se apresentasse estável e funcional.
6	Testes definitivos eram feitos em clusters, sendo que para 1, 3, 2, 4 nós no cluster no NBCGIB e para 1,2,4,8,12,16,20,24 nós no CENAPAD-SP.
7	Em caso de falha em qualquer um testes era feita uma análise da falha, sendo esta simulada algumas vezes para melhor entendimento e posterior correção do problema.

3.1. Alocação dinâmica de memória no ICANT

A linguagem Fortran 90, para qual o programa ICANT foi atualizado anteriormente, facilitou o trabalho de paralelização de memória por suas características mais flexíveis em relação a manipulação de espaços de memória, características essas que não existiam na linguagem Fortran 77 em que o programa foi escrito originalmente.

A constante alocação e liberação de espaços de memória é fato muito comum na programação paralela, pois vetores de dados de tamanhos variáveis precisam ser enviados entre os nós no decorrer da execução (a depender do tipo de paralelização) e quase nunca se sabe o tamanho destes que pode depender do montante de dados, dos algoritmos de balanceamento de carga[21], e ,concerteza, da configuração de número de nós que a cada execução pode mudar (dependendo do número de processadores disponíveis).

Portanto é imperativo que os espaços de memória sejam alocados depois que todas essas condições estão definidas e isso, é claro, é alcançado somente depois que o programa já esta em execução.

A alocação estática[22] de memória praticamente inviabiliza qualquer esforço de paralelização, dada a grande dinamicidade que é exigida na manipulação de espaços de memória pelo paralelismo, o que reitera a importância do trabalho realizado por (SOUZA, 2007) par que fosse possível a paralelização de dados no programa ICANT em seu módulo chamado ICANTDO.

As funções disponíveis na linguagem Fortran 90 que foram utilizadas para a manipulação dinâmica no programa foram:

```
complex*16,dimension(:,:),allocatable :: R   ! declara R
allocate(R(sizeN, sizeN))                    ! aloca R
deallocate(R)                                ! desaloca R
```

O compartilhamento de variáveis entre o programa principal e as diversas sub-rotinas é feita pela declaração de módulo pela diretiva *module*:

[21] Técnica para a divisão igualitária de trabalho entre computadores, muito usada em sistemas distribuídos e paralelos.
[22] Alocação de memória predefinida na compilação do programa, onde, no momento em que o programa é executado, o espaço de memória é alocado.

```
module trjM
    integer xtru, elim, elim_local
    integer, dimension(:,:), allocatable ::  xtruc
    integer*2, dimension(:,:,:), allocatable :: TS
    complex*16, dimension(:,:), allocatable :: R, TSp
end module trjM
```

seguida da importação em cada sub-rotina através da diretiva *use*:

```
SUBROUTINE ixbac2(cont, cont2, elmnt0, rang, Co, Nchps,
                  actlis, pitchB)
    use sizes
    use depxM
    use depyzM
    use trjM
    use mpidataM
    use mpiDataM2
    ...
```

3.2. Reestruturação do ICANT

O código ICANT foi projetado para solucionar diversos problemas relacionados a modelagem de antenas de ICRH, logo o mesmo apresenta em sua implementação diversos possíveis caminhos de execução (execução com plasma, em vácuo, com simetria, com spectro) e utilização de muitas diretivas de controle de fluxo (se, para, enquanto) e várias estruturas de memória (vetores, matrizes, structs). Isso torna o código confuso e de difícil entendimento além de dificultar sua possível extensão ou modificação, a depender do tipo de uso a que pode ser submetido.

No código ICANT foram feitas basicamente 3 grandes modificações além de diversas outras pequenas modificações para que se viabilizasse ou mesmo otimizasse o processo de paralelização, diminuindo número de comunicações via rede e o uso de memória do programa.

A primeira consistiu da unificação de 4 chamadas da mesma função (TRJessai). Essas chamadas em separado serviam para a obtenção de informações adicionais sobre o comportamento dos elementos da matriz que não influenciavam no curso normal de processamento do programa e nos resultados finais.

Durante o cálculo dos elementos de antena na execução da sub-rotina TRJessai2 a matriz R foi desalocada (por já existir uma lista com os elementos) o

que economiza memória e possibilita que durante este outros programas ou mesmo o sistema operacional tenha maior espaço pra suas demais operações, já que essa é a parte do código responsável por cerca 90% do tempo total de execução do programa e essa constituiu a segunda modificação.

Dado que o diferencial do programa ICANT é a possibilidade de realizar os cálculos em presença da coluna de plasma e vácuo, o cálculo do espetro a priori foi deixado, o que economizou muito em se tratando de recursos de memória, pois havia uma cópia da matriz R chamada R1 criada somente para esse propósito, além de outros testes e algoritmos auxiliares que foram eliminados, sendo esta a terceira reestruturação do código ICANT.

3.3. Paralelização do código

A paralelização do programa ICANT, foi iniciada pela implementação da divisão das estruturas de dados, sendo seguida pela posterior modificação dos algoritmos, processos e sub-rotinas em que essas estruturas eram manipuladas. As estratégias que foram utilizadas nesse desenvolvimento estão descritas nos tópicos que seguem e a compilação e execução do programa fica da seguinte forma para mpich2:

```
$ ifort -O3 -ipo -prefetch -o icantell icantell_x.for braffa3.for divpag.for nagimsl.for
$ mpif90 -O3 -ipo -prefetch -o Icantdo7Parallelv3.1.7 Icantdo7Parallelv3.1.7.f90
$ ./icantell
$ mpirun -machinefile maquinas -n 5 ./ Icantdo7Parallelv3.1.7 > arquivoSaida
```

3.3.1. Paralelização das estruturas de dados

O código ICANT, como já dito, utiliza as matrizes R (ver Figura 10) e TS para o armazenamento das informações relativas ao cálculo das correntes dos elementos de antena, sendo estas as estruturas mais exigentes com relação a quantidade de memória. Essas duas estruturas são manipuladas no decorrer do programa por várias sub-rotinas, principalmente a matriz R que sofre várias modificações ou contribuições em cada uma.

A divisão destas matrizes implicou diretamente na modificação e paralelização de quase todas as sub-rotinas do código, onde pode-se citar as sub-rotinas TRJessai, IxElim, FindSym, IxBac, Continui e Separati, além é claro de grandes modificações no programa principal e foi feita com o melhor balanceamento de carga possível levando em conta que elementos de uma mesma estrutura radial e elementos com continuidade deveriam permanecer unidos.

Vale ressaltar que primeiro foi pensado a divisão da matriz em linhas como na Figura 10, porém o Fortran armazena matrizes na memória coluna por coluna (sendo a matriz um grande vetor de colunas consecutivas) e não linha por linha como linguagens convencionais como C fazem, então decidiu-se por dividir as matrizes por colunas (Figura 11).

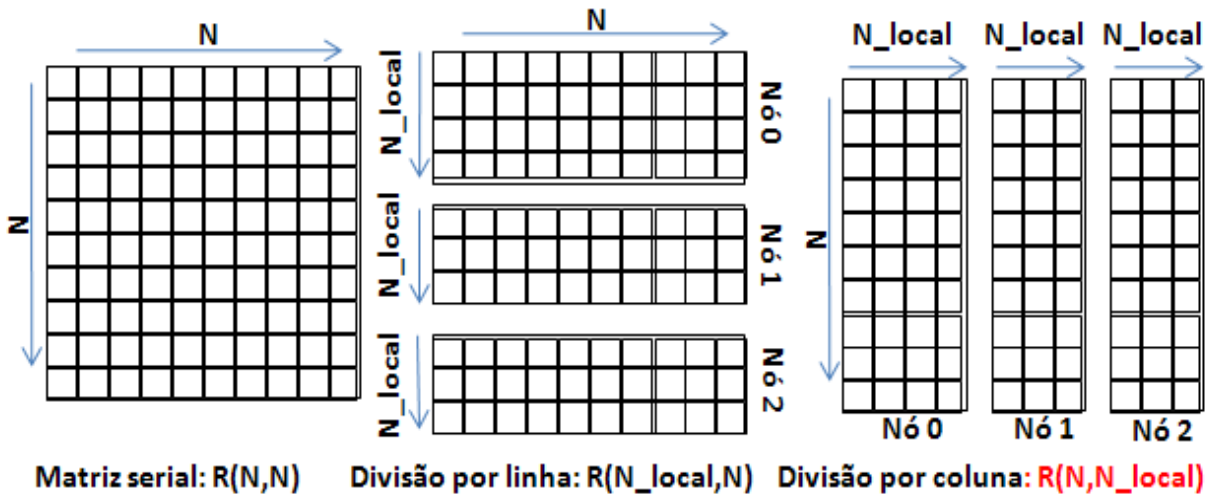

Figura 11. Do lado esquerdo tem-se a matriz R na sua forma serial, no meio tem-se a primeira idealização de paralelização de R que seria por linha, no lado direito tem-se a estrutura paralela da matriz R que foi adotada, a paralelização por colunas que foi feita também na matriz TS.

O algoritmo implementado foi desenvolvido por etapas e possui o seguinte funcionamento: primeiro é feito o balanceamento de carga ótimo, onde através das equações

$$n\_local = \frac{N}{numNós}$$

$$n\_resto = N - (n\_local * numNós)$$

é obtido o *n_local* inicial para cada nó. O *n_local* definitivo é encontrado distribuindo-se igualitariamente *n_resto* entre os nós (somando-se a cada *n_local*). Como exemplo, se temos *N=448* e *numNós=15*, encontraremos inicialmente *n_local=29* e *n_resto=13*, ou seja, todos os nós ficariam com 29 elementos para calcular. A

distribuição do resto em todos os nós quanto possível proporciona equilíbrio máximo ficando os 13 primeiros nós com 30 elementos e os 2 últimos com 29, ou seja, os 13 primeiros nós iriam alocar uma matriz R(448, 30) e os 2 últimos uma matriz R(448,29) que juntas seriam a matriz completa R(448,448).

As estruturas radiais (estruturas definidas no arquivo *xtructs.dat*) precisam estar contíguas em cada nó, ou seja, não haver fragmentação desta em mais de um nó, por conseguinte, quando a divisão do algoritmo anterior cai no meio de uma estrutura, o limite é movido para a extremidade da estrutura que estiver mais próxima (ver figura 12). Os elementos que tem continuidade como próximo (informação definida no arquivo fr014.dat) também não podem estar em nós distintos, executando-se o mesmo procedimento para preservá-los contíguos.

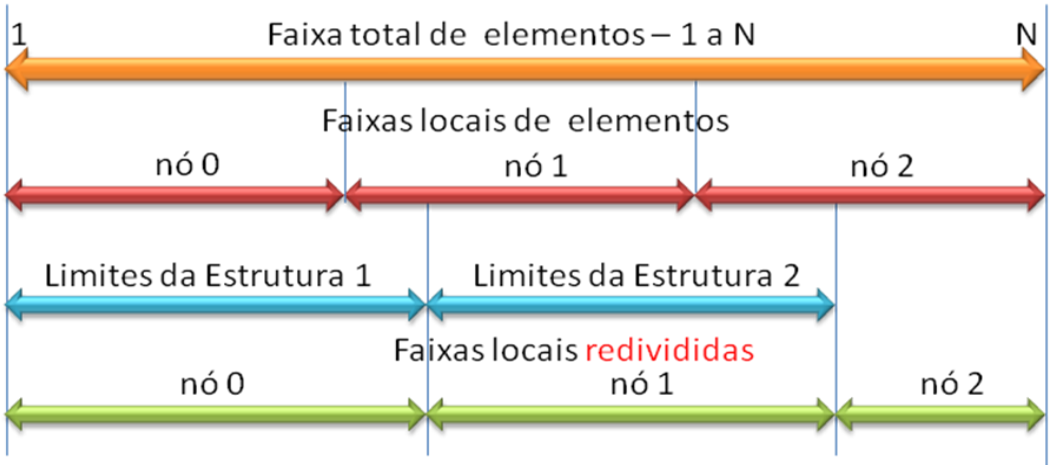

Figura 12. Na faixa laranja tem-se o total de elementos (1 até N), nas faixas vermelhas tem-se a divisão que é feita pelo algoritmo de divisão balanceada para cada nó. As faixas azuis representam os elementos que compõe cada estrutura radial e delimita estas. As faixas verdes representam a "redivisão" dos elementos a serem calculados por cada, dado que as estruturas não podem ficar separadas. Pode se pensar que o balanceamento seja comprometido com essa "redivisão", porém para casos médio-grandes de execução (4192 elementos, por exemplo) existem cerda de algumas centenas de estruturas radias de tamanho aproximadamente igual a 10 elementos cada, então a degradação do balanceamento causada pela movimentação desses limites é quase sempre insignificante.

O algoritmo que descreve esse processo foi colocado numa sub-rotina por ser usado em várias partes do código e constitui-se dos seguintes comandos:

```
1  subroutine divideBands(bandInf, bandSup, numNos, id, limInf, limSup)
2     use trjM
3     implicit none
4     integer resto,iter,desloc,range,i,j,k
5     integer bandInf, bandSup, numNos, id, limInf, limSup
6
7     iter=0
8     desloc=1
9     range = abs(bandInf -bandSup)
10
11    if((mod(range, numNos)).ne.0) then
12       resto = mod(range, numNos)
13       if (id <= resto) then
14          iter = id
15          desloc=0
16       else
17          iter = resto
18          desloc=1
19       endif
20    endif
21
22    limInf = (id * (range/numNos) + bandInf) + iter + desloc
23    limSup = ((id + 1) * (range/numNos) + bandInf) + iter
24
25    if(id.eq.0)then
26       limInf=bandInf
27    endif
28
29    if(id.eq.(numNos-1))then
30       limSup=bandSup
31    endif
32
33 end
```

Os seus parâmetros *bandInf, bandSup, numNos, id,* são de entrada (são respectivamente o número que inicia a faixa, o número que finaliza a faixa, o número total de nós e a identificação do nó que quer descobrir em qual número começa e em qual termina sua faixa) e os parâmetros *limInf, limSup* são de saída (exatamente os número onde começa e termina a faixa para o nó *id*) e fazendo *limSup – limInf,* obtém-se justamente a quantidade de colunas que o nó *id* precisa alocar (ver a Figura 13).

3.3.2. Paralelização do cálculo preliminar de R

O primeiro cálculo que o ICANTDO realiza é a compilação da matriz *R* para uma pequena faixa do espaço de modos K_y e K_z que vai de -5 a 5, realizada pela sub-rotina TRJessai. A modificação desse processo para que funcionasse com as partes da matriz distribuídas entre os nós mostrou-se relativamente fácil, pois o calculo de cada elemento *R(e1,e2)* da matriz é independente.

Nessa etapa da paralelização se fez necessário o cálculo para cada nó do elemento que inicia e termina a faixa de colunas da matriz R que serão calculadas pelo nó (ver Figura 13).

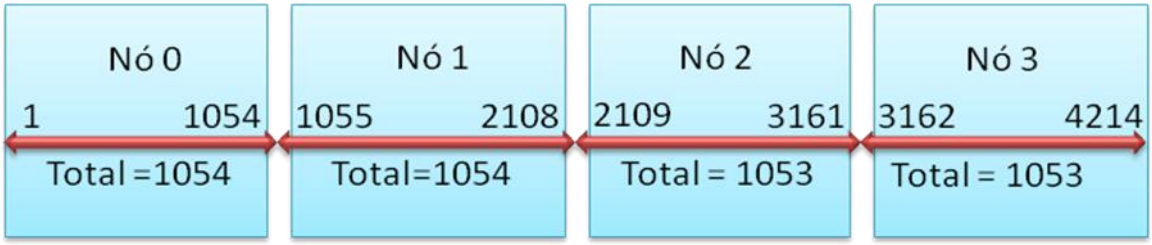

Figura 13. Distribuição balanceada da faixa total de elementos. No exemplo desta figura temos 4 nós sendo que o primeiro e o segundo calcularão 1054 elementos e o terceiro e o quarto 1053 elementos.

A sub-rotina citada no tópico anterior foi construída para que atendesse primeiro a essa necessidade, os seus parâmetros de saída *limSup* e *limInf* retornam justamente esse o elemento mínimo e máximo da faixa de colunas que o nó precisará calcular.

Vale ressaltar que até este ponto não foi necessária a utilização das rotinas de comunicação do MPI.

3.3.3. Paralelização da sub-rotina IxElim

Para o cálculo mais detalhado do comportamento da corrente nos elementos de antena que compõe as estruturas radias é necessária a composição destes por vários outros elementos de corrente, onde, por exemplo, se tem 4 elementos de antena radiais ter-se-á 10 elementos de corrente que na maioria das vezes são proporcionais entre si e não precisam ser todos calculados.

A retirada desses elementos (retirada de linhas e colunas da matriz R) do cálculo e a preservação das respectivas constantes de proporcionalidade para a

posterior recolocação é feita pela surotina IxElim, e constitui trabalho importante pois a dimensão da matriz fica bem reduzida.

O processo serial de retirada das linhas e colunas de R é feito do início para o fim da matriz onde primeiro se encontram os pares de linhas que são proporcionais e suas respectivas razões, depois se retiram uma das linhas de cada par, reescrevendo esta linha com a próxima linha da matriz e isso até o final. O que é feito pra linha da matriz é feito também para a coluna (eliminando-se a linha 5 deve-se também eliminar a coluna 5) e diminui o tamanho da matriz nas duas dimensões (ver Figura 14).

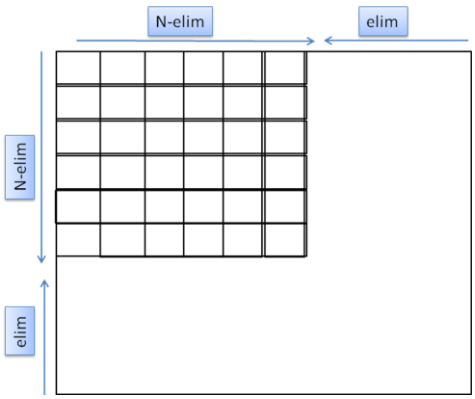

Figura 14. Matriz R depois de ter linhas e colunas proporcionais eliminadas pela sub-rotina IxElim em sua forma de execução serial. e*lim* representa o total de linhas eliminadas e N o total de linhas/colunas da matriz.

A paralelização desse processo constituiu basicamente da busca paralela pelos pares de linhas proporcionais e da retirada destas em cada nó. Foi necessária a utilização da rotina MPI_ALLGATHERV[23] no vetor *rlist_local* (informações de linhas e suas proporcionalidades) e na variável *elim_local* (quantidade de linhas eliminadas) para se colocar em todos os nós a informação sobre todas as linhas proporcionais e suas razões encontradas em cada nó, pois estes precisariam dessa informação para eliminar e, posteriormente, na sub-rotina IxBac, recolocar as linhas (dimensão da matriz R não paralelizada) e as colunas da sua faixa de colunas da matriz R (ver Figura 15).

[23] Essa rotina de comunicação do padrão MPI implementa as mesmas funções da MPI_GATHERV, só que o resultado é posto em todos os nós de processamento. Se tivermos um buffer em cada e precisamos que destes unidos e esta união precisa ser colocada em todos os nós então essa é a rotina a ser utilizada.

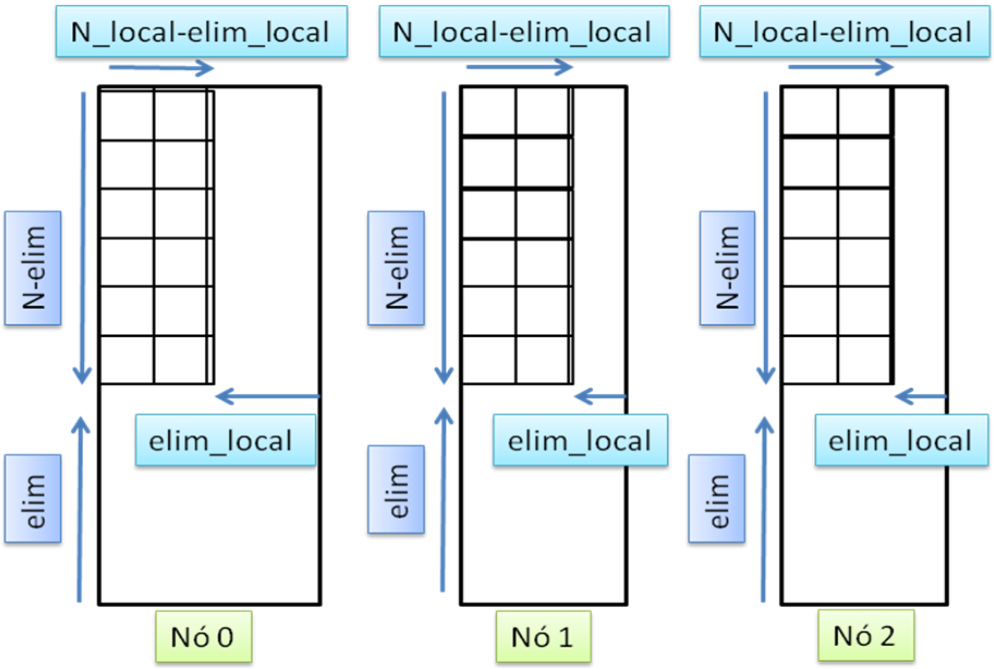

Figura 15. Matriz R distribuída ao longo de 3 nós após ser processada pela sub-rotina *IxElim* paralela. Foram retiradas 4 linhas em todos os nós (pois cada nó tem uma parte de cada linha), 2 colunas da primeira faixa (no nó zero) e 1 coluna da segunda e da terceira faixa o que totaliza a retirada de todas as 4 colunas. e*lim* representa o total de linhas eliminadas, *elim_local* representa o número de colunas eliminadas em cada faixa, *N* o número total de linhas/colunas e *N_local* o número total de colunas em cada faixa da matriz *R*.

3.3.4. Paralelização da sub-rotina FindSym

Esta sub-rotina, como mostrado antes, tem o papel relevante na diminuição da quantidade de elementos da matriz *R* a serem calculados. Após ter *R* sido calculado com um pequeno número de modos, cada elemento de *R* é comparado com todos os outros. Todos os elementos com os mesmos valores (até certa tolerância) são marcados como repetições, gravadas na matriz *TS,* construída a fim guiar a cópia dos valores após o fim do cálculo de TRJESSAI. Este procedimento reduz os tempos de cálculo por um fator de até 90% mas para matrizes maiores (alto *N*) o próprio cálculo das simetrias passa a ser custoso.

Como agora tem-se a matriz divida em faixas de colunas em cada nó, para se alcançar a comparação de todos os elementos com todos foi necessária a execução da busca em vários níveis, sendo o primeiro nível a busca interna em cada nó, o segundo nível a busca entre os nós com os elementos restantes (que não foram marcados simétricos no primeiro nível). Até este ponto todos os elementos simétricos encontrados pelo algoritmo serial também é encontrado pelo algoritmos

paralelo, só que da forma como foi feita a busca entre nós (nível 2), sempre sobravam mais elementos para calcular no nós iniciais e menos elementos para calcular nos nós de *rank*[24] maior, então foi feito o terceiro nível para equilibrar o número de elementos de R calculados em cada nós.

No nível 1 de paralelização de FindSym a busca é realizada pelo mesmo algoritmo serial que utiliza o método da força bruta[25], onde cada elemento é comparado por todos os outros, só que agora isso é feito na parte da matriz em cada nó (ver Figura 16).

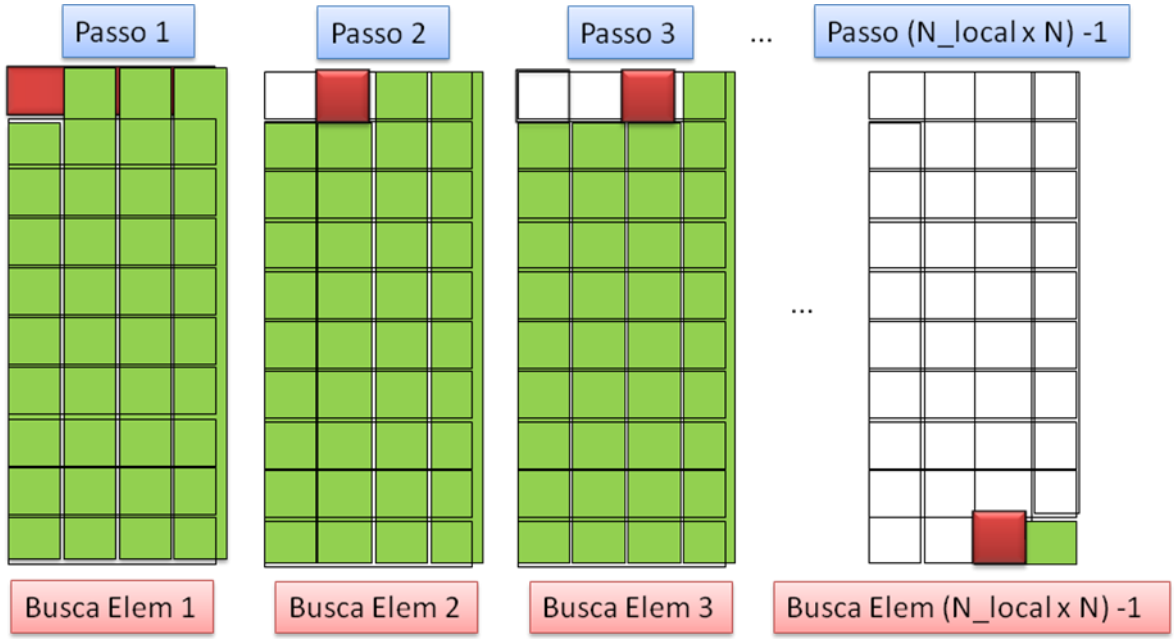

Figura 16. Busca de simetrias nas faixas de cada nó. No passo 1 é realizada a busca pelas simetrias do elemento 1 (em vermelho) da faixa de R com os próximos (em verde). No passo 2 é realizada a busca pelas simetrias do elemento 2 com os elementos em verde, no passo 3 as do elemento 3 com os elementos em verde e no último passo são comparados somente o penúltimo com o último elemento da faixa de R. Esse processo é feito para cada faixa de R em cada nó em paralelo. Após isso temos as simetrias locais (em cada nó) encontras.

No segundo nível de paralelização a busca é realizada entre todos os nós com os elementos que sobraram da busca interna em cada nó. Como a busca seria feita entre os nós, então, por exemplo, numa execução com 3 nós poderiam existir buscas paralelas do nó 0 com nó 1, nó 1 com nó 2 e do nó 2 com o 0 ao mesmo

[24] Variável do tipo inteiro que identifica um nós em um grupo de execução. À medida que se criem novos grupos, novos *ranks* são atribuídos aos componentes dos grupos, sendo que se inicia com zero e em cada grupo não se repete.

[25] Abordagem simplista para a resolução de um determinado problema, geralmente baseado diretamente na declaração do mesmo e nas definições dos conceitos envolvidos. Trata-se da técnica de projeto de algoritmos de mais fácil codificação, porém quase sempre de complexidade elevada (lento) e não otimizada.

tempo e no próximo passo seria o nó 0 com o 2, nó 1 com nó 3 e do nó 2 com o 1 e assim por diante. Porém poderia ocorrer de um elemento do início da matriz (nós com *rank* menor) ser simétrico de um no fim desta, o que transgride o princípio físico de que somente elementos do fim da matriz podem ser simétricos a elementos do início desta e não na direção inversa. Neste ponto foi perceptível que se fosse feito um algoritmo de busca entre nós que minimizasse o número de envio de mensagens[26], o montante de elementos presente em cada nó (minimização de memória) e maximizasse o paralelismo durante a busca resultaria num algoritmo bastante complexo e que não seguiria o princípio da direção de simetria, pois poderiam haver simetrias na direção inversa. Numa tentativa de se ter um algoritmo razoável, foi idealizado a criação de vários grupos de processamento onde o algoritmo se processaria da seguinte forma (sendo ***n*** o número de nós):

*Passo **1**:*	*1 Criar grupo com **todos os nós** (MPI_COMM_WORLD)* *2 Enviar elementos ainda não simétricos do **nó zero para todos** (MPI_BCAST, root=0)* *3 Os outros nós procuram simétricos em sua parte da matriz á elementos do nó **zero***
*Passo **2**:*	*1 Criar grupo dos nós **sem o nó zero (a partir do nó um)*** *2 Enviar elementos ainda não simétricos do **nó um para todos** (MPI_BCAST, root=1)* *3 Os outros nós procuram simétricos em sua parte da matriz á elementos do nó **um***
*Passo **3**:*	*1 Criar grupo dos nós **sem o nó zero e o nó um (a partir do nó dois)*** *2 Enviar elementos ainda não simétricos do **nó dois para todos** (MPI_BCAST, root=2)* *3 Os outros nós procuram simétricos em sua parte da matriz á elementos do nó **dois***
…	**…**
*Passo **n-1:***	*1 Criar grupo dos nós **n-2 e n-1*** *2 Enviar elementos não simétricos do **nó n-2 para n-1** (MPI_BCAST, root= **n-2 nó**)* *3 Os outros nós procuram simétricos em sua parte da matriz á elementos do nó n**-2***

Esse algoritmo de busca entre os nós possibilitou que se encontrassem todos os elementos simétricos encontrados pelo algoritmo serial através do uso da rotina de comunicação simples do MPI MPI_BCAST[27] ***n-1*** vezes e não quebra o princípio da direção de simetria, pois como se pode ver na Figura 17, é possível somente existirem simetrias de elementos posteriores a elemento anteriores.

[26] Neste contexto, trata-se do nome geral dado a comunicação realizada entre nós de processamento pelo padrão MPI, que consiste do envio de bytes de dados através comunicação TCP/IP (protocolos de redes padrão Ethernet). A minimização do envio destas mensagens é a premissa básica da programação paralela.

[27] Operação constante no padrão MPI que implementa uma operação onde o processo que a esta realizando, envia os mesmos dados (*buffer*) para todos os processos (inclusive ele próprio) no grupo de processamento.

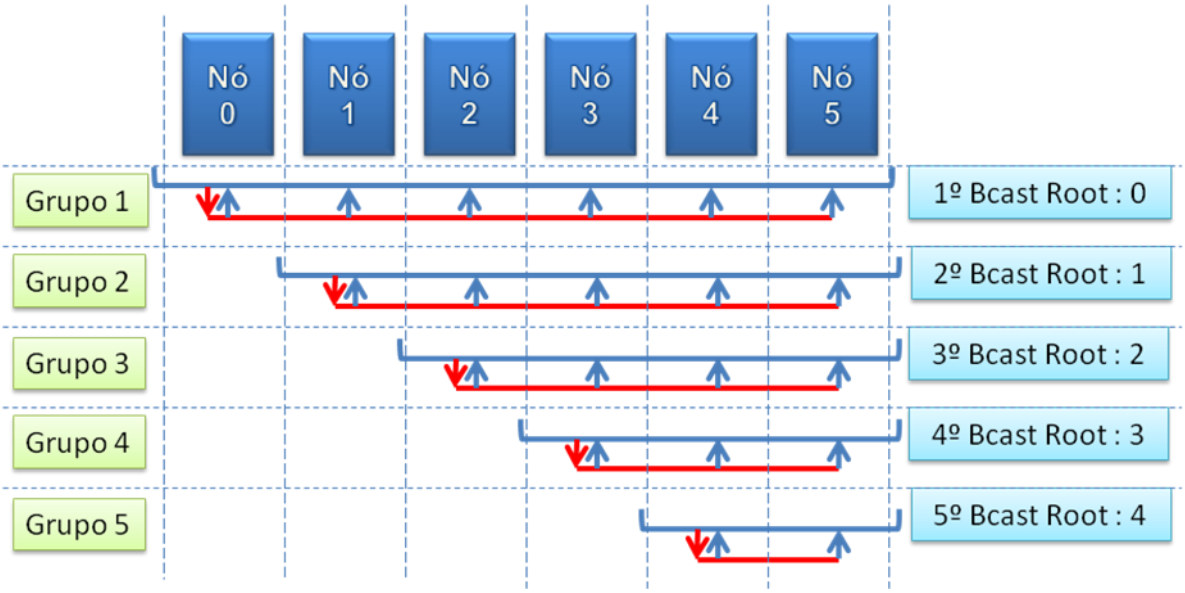

Figura 17. Nível 2 de busca por simetrias. Passo 1: É criado o grupo com todos os nós (MPI_COMM_WORLD) – Grupo 1; através de Bcast, o nó zero envia seus elementos para todos; o outros nós procuram simetrias dos elementos do nó zero com seus elementos e as armazenam em suas faixas de TS locais. Passo 2: É criado o Grupo 2; através de Bcast o nó um envia seus elementos para todos do Grupo 2; estes comparam os elementos do nó um recebidos com os seus e armazenam as simetrias encontradas. Este processo se repete até que o grupo somente contenha 2 nós, onde é finaliza a busca.

Apesar de atender aos requisitos, esse algoritmo tem o problema de deixar muito mais elementos para serem calculados nos nós de *rank* menor que nos de *rank* mais alto (ver Figura 18), o que afeta diretamente o balanceamento de carga e posteriormente poderia degradar o *speedup*[28] geral do programa.

Figura 18. Nesta figura o retângulo verde significa o montante de elemento que restaram para ser calculados em cada faixa depois do nível 2 de busca por simetrias e, como pode-se observar, estas quantidades não estão balanceadas. O fato de os últimos nós sempre terem menos elementos a serem calculadas existe por que estes sempre fazem comparações com mais nós e encontram mais simetrias do eu os primeiros nós. O último nó faz comparações com todos os outros. O nó zero não faz comparações com nenhum nó, somente envia para os outros seus elementos para serem comparados.

[28] Fator de redução do tempo de execução de um programa paralelizado em P processadores. Esse fator é calculado dividindo-se o tempo de execução do programa com 1 nó, pelo tempo de execução do programa em p nós de processamento. Existem duas modalidades de speedup ou aceleração sendo a primeira o speedup relativo onde o caçulo é feito utilizando-se o tempo de execução do programa paralelo em 1 processador como base, e o speedup absoluto que utiliza-se do tempo de execução do "melhor" programa seqüencial.

O nível 3 de paralelização desta subrotina soluciona este problema de forma relativamente simples: Todos os nós enviam pro nó zero os elementos de sua faixa de R que passaram no teste de simetrias (no total cerca de 5 a 10 % da matriz original) utilizando a rotina de comunicação MPI_GATHERV[29]; o nó zero recebe estes elementos e os envia de volta para todos os nós, só que agora de forma balanceada utilizando para tal o algoritmo de balanceamento mostrado no tópico Paralelização das estruturas de dados, sendo estes os elementos que serão calculados em cada nó de processamento. Estes dados são enviados de volta utilizando a rotina de comunicação MPI_SCATTERV[30], e nesse nível 3 quando se fala em enviar elementos, entenda-se, enviar os elementos que são do tipo MPI_DOUBLE_COMPLEX (16 bytes) e enviar as coordenadas de linha e coluna destes relativas a matriz que são inteiros de 2 bytes, ou seja, são enviados 3 vetores variantes de informações.

Os passos abaixo resumem o processo paralelização de FindSym:

Passo 1 - busca por simetrias em cada faixa de R em cada nó
Passo 2 - busca por simetrias entre as faixas em cada nó
Passo 3 - envio de todos os elementos não simétricos nos nós para o nó zero
Passo 4 - nó zero envia esses elementos de volta para todos os nós de forma balanceada

3.3.5. Paralelização da sub-rotina TRJessai2

O cálculo final da matriz de impedância R na versão paralela da sub-rotina TRJessai2 é feito utilizando-se da lista equitativamente distribuída pela sub-rotina paralela FindSym dos elementos e suas coordenadas relativas a sua posição na matriz R original, ou seja, da mesma forma com que essa sub-rotina calculava os elementos que escapavam da busca por simetrias e ficavam esparsos na matriz R, ela calcula essa lista de elementos.

[29] Sub-rotina constante no padrão MPI, que coloca os buffers a serem enviados de todos os nós em um buffer (de tamanho igual ou maior a soma de todos os outros a serem enviados) nó definido como *root*, obedecendo a ordem de *ranks,* podendo estes vetores serem de tamanhos variados.
[30] Tipo de comunicação do MPI que faz o trabalho inverso da comunicação *gather*, distribuindo entre todos os nós partes de um buffer presente no nó configurado como root, sendo que estas partes podem ter tamanhos diversos.

3.3.6. Paralelização da reconstrução de R

Após o cálculo da lista de elementos de *R* é necessária a reconstrução da respectiva matriz, para o prosseguimento dos cálculos. Essa reconstrução foi implementada no programa principal e utilizou-se das informações constantes na matriz *TS* para se realizar. Do mesmo modo que a retirada das simetrias na sub-rotina FindSym foi feita em níveis, a reconstrução também foi e o que permitiu isso foi a forma com que FindSym marcava as simetrias em cada nível, sendo 0 para os elementos que não tem simetria, 1 para as simetrias encontradas internamente (nível 1) e 3 para as simetrias encontradas na busca entre nós (nível 2).

O processo de reconstrução inicia-se com a realocação da matriz da forma *R(N, N_local)*, seguido da recolocação dos elementos da lista que foram calculados em TRJessai2. São recolocadas as simetrias encontradas no nível um, antes, porém é preciso que todos os nós tenham acesso à lista completa dos elementos que foi calculada em cada nó através de chamada a rotina de comunicação do MPI MPI_ALLGATHERV, para as três listas (elemento, coordenada de linha e coordenada de coluna deste na matriz *R*). Após a recolocação dos elementos simétricos de nível um, são recolocadas as simetrias pra o nível dois o que finaliza a reconstrução de *R*, lembrando que este procedimento é feito em cada nó para a sua faixa de *R*.

3.3.7. Paralelização da sub-rotina IxBac

O processo de recolocação dos elementos ou linhas retiradas por IxElim fica facilitado por já existir os dados para todas linhas eliminadas em todos os nós, pois na paralelização de IxElim foi usada a rotina MPI_ALL_GATHERV do MPI para disseminação da estrutura chamada *rlist*(N, 3), que na posição *rlist*(N, 1) marca primeira linha, na posição *rlist*(N, 2) a segunda linha e na posição *rlist*(N, 3) a razão entre as duas linhas.

O mesmo algoritmo serial oi usado para a reconstrução com a diferença que agora se estaria lidando com uma parte da matriz em não coma matriz completa, o

que obrigou a “localização” de informações como o número de linhas proporcionais em nós, e as próprias com suas proporções em cada nó. Essas informações foram facilmente obtidas já que ao final da busca local em IxElim tem-se justamente essas informações (como a busca é realizada localmente tem-se *elim_local* que guarda para cada nó o número de linhas proporcionais e *rlist_local* que guarda essas linhas e sua respectivas razões de proporcionalidade).

De posse destas informações bastou substituir no algoritmo serial as variáveis que antes eram para a matriz completa para as variáveis locais, ou seja, onde se tinha elim, se substituiu por elim_local e onde se tinha rlist por rlist_local, porém, é claro, havia substituição somente quando se tratava da dimensão de colunas da matriz, pois na dimensão das linhas o algoritmo serial permaneceu inalterado.

3.4. Paralelização de E/S

A obtenção de resultados com o programa ICANT implica em várias execuções do programa para um tipo de antena, *tokamak* e plasma, variando-se entre essas execuções alguns poucos parâmetros. Estruturas principais como a matriz *R* (em suas partes imaginária e real) e a matriz *TS* precisam ser obtidas apenas em uma das execuções, sendo que nas próximas elas serão reutilizadas. Esse processo de reutilização é possibilitado através da escrita destas em disco para posterior leitura e utilização.

Sabe-se que estas estruturas podem ser muito grandes a depender do caso em execução pelo programa. É sabido também que a velocidade de processos de E/S é muito inferior que a velocidade de processamento do sistema o que, a depender do caso, pode causar atrasos significativos no tempo de execução final. Outro problema decorrente de se ter um programa paralelo com a saída serial é o fato de que todos os processos vão escrever no mesmo arquivo de forma aleatória (em sistemas onde os sistemas de arquivos esta sob NFS[31]), onde a ordem dos elementos é perdida e o arquivo fica inutilizado para posterior recuperação (ver Figura 19).

[31] Serviço de rede muito comum em sistemas Linux que permite o compartilhamento transparente de sistemas de arquivos ou diretórios entre os nós de uma rede. Permite aos administradores criem sistemas de arquivo centralizados que facilitam tarefas de gerência tais como manutenção e suporte.

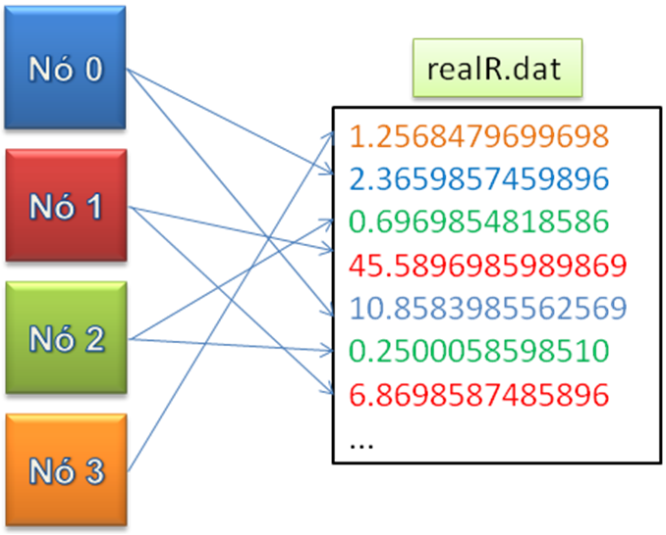

Figura 19. Vários nós executando em paralelo escrevendo no mesmo arquivo. Como pode-se observar, os valores são escritos fora da ordem, que deveria ser os valores do nó zero primeiro, depois os do nó um, depois, do dois e no fim do arquivo os valores do nó três.

Uma abordagem inicial para a solução desse problema seria o envio dos dados para o nó zero, onde este os escreveria para o sistema de arquivos (Figura 20). Isso é bom, pois grandes blocos de arquivos melhoram a performance, mas limita a escalabilidade e não faz sentido já que intentou-se fazer a paralelização de memória do ICANT justamente para não ter as partes dessas matrizes unidas em um único nó, o que acabaria acontecendo se estas fossem unidas em um nó para serem escritas.

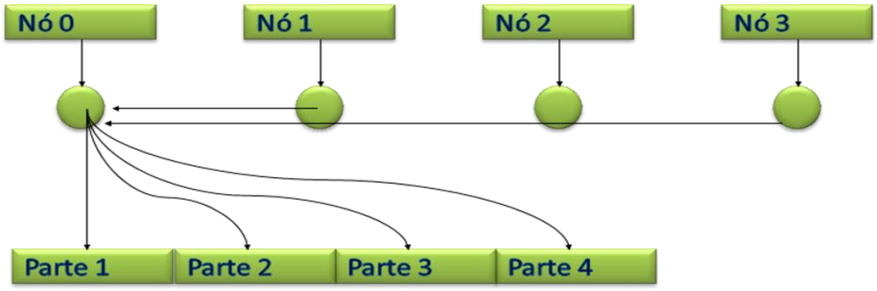

Figura 20. Todos os nós mandam os dados para o nós zero que os escreve para disco. Retirada de (THAKUR, 2005).

Outro caminho seria a escrita por cada nó em arquivos separados, ou seja, cada nó faria sua escrita para disco independentemente dos outros. Isso proporciona paralelismo e não desorganiza a ordem dos dados, porém tem-se vários pequenos arquivos para serem manipulados e existirá uma certa dificuldade de lê esses dados em vários arquivos para uma execução com configuração de número de nós diferente da que eles foram gerados.

A solução adotada nesta paralelização foi ter somente um arquivo sendo manipulado pelos diversos nós de execução do programa paralelo, sendo que cada nó poderá manipular a parte do arquivo que lhe é devida (ver Figura 21). Este método foi possibilitado pelo MPI através da implementação ROMIO do Argonne National Laboratory que dispõe de variadas rotinas para o tratamento paralelo de arquivos como as *views* que definem a parte do arquivo que o nó poderá manipular e podem se configuradas com MPI_File_set_view.

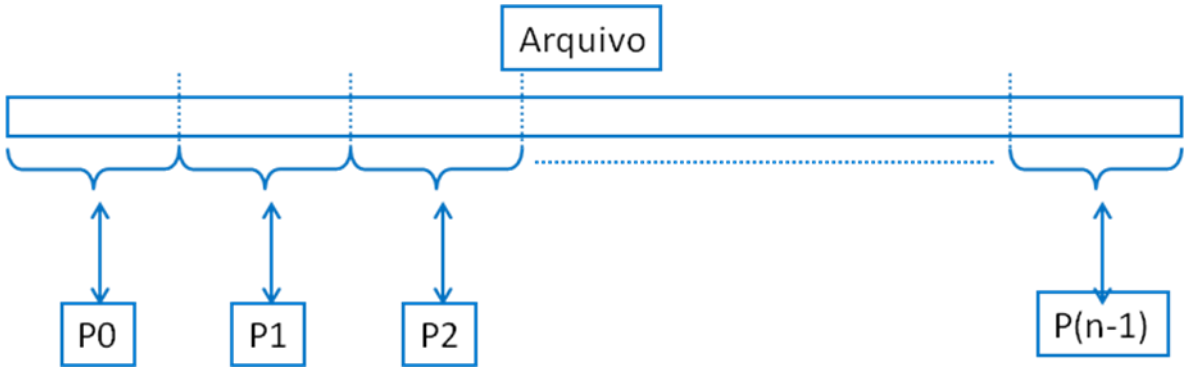

Figura 21. Arquivo sendo compartilhado pelos diversos nós de processamento. Obtida de (THAKUR, 2005)

O código abaixo em linguagem C exemplifica o uso das principais rotinas do MPI I/O:

```
MPI_File thefile;                                   !declaração do arquivo
for (i=0; i<BUFSIZE; i++)
   buf[i] = myrank * BUFSIZE + i;                   !preenchimento dos dados
MPI_File_open(MPI_COMM_WORLD, "testfile",           !abre arquivo
              MPI_MODE_CREATE | MPI_MODE_WRONLY,
              MPI_INFO_NULL, &thefile);
MPI_File_set_view(thefile, myrank * BUFSIZE * sizeof(int),   !configura a view
                  MPI_INT, MPI_INT, "native",
                  MPI_INFO_NULL);
MPI_File_write(thefile, buf, BUFSIZE, MPI_INT,      !escreve na parte p do arquivo
               MPI_STATUS_IGNORE);
MPI_File_close(&thefile);                           !fecha o arquivo
```

Voltando ao código ICANT, os arquivos *TS.dat, imagR.dat, realR.dat* que armazenam respectivamente as informações de simetria na matriz R, a parte imaginária de todos os elementos de R e a parte real de todos os elementos de R, podem ter algumas centenas de megabytes e foram os principais alvos da paralelização de E/S implementada no programa.

4. RESULTADOS E DISCUSSÃO

A implementação da paralelização do programa ICANT iniciou com a versão *Icantdo7.f90* oriunda de (SOUZA, 2007) e, após a paralelização de memória resultou na versão estável *Icantdo7Parallelv3.1.6.f90*. A paralelização da entrada/saída foi aplicada nesta, e gerou a versão *Icantdo7Parallelv3.1.7.f90*.

A forma escolhida para a análise do ganho de desempenho obtido com a paralelização e reestruturação de memória no código ICANT foi o cálculo do *speedup* do programa para várias configurações de número de nós nos *clusters* do NBCGIB e do CENAPAD, além de medidas de eficiência. Inicialmente foram feitas execuções no cluster do NBCGIB, pois neste não há concorrência pelos recursos assim com é no CENAPAD que, por fazer parte de um centro nacional de computação de alto desempenho, é utilizado por pessoas de todo o país, e, exatamente por isso, existe um sistema de submissão de trabalhos que implementa as políticas de uso dos vários nós de processamento que são compartilhados entre trabalhos seriais e paralelos. Neste último também não há possibilidade de acesso via *ssh* nos nós de processamento, somente é permitido no servidor e isso impossibilitou tarefas como a análise do consumo de memória no decorrer da execução do programa, que precisavam ser feitas nos nós de processamento.

Os resultados obtidos no cluster do NBCGIB para o caso de 3248 elementos foram bem satisfatórios e mostram um ganho de *speedup* próximo do ideal, sendo que ouve diferença mínima entre os *speedups* relativo e absoluto e quase não houve perda de eficiência (ver Figuras 22 e 23).

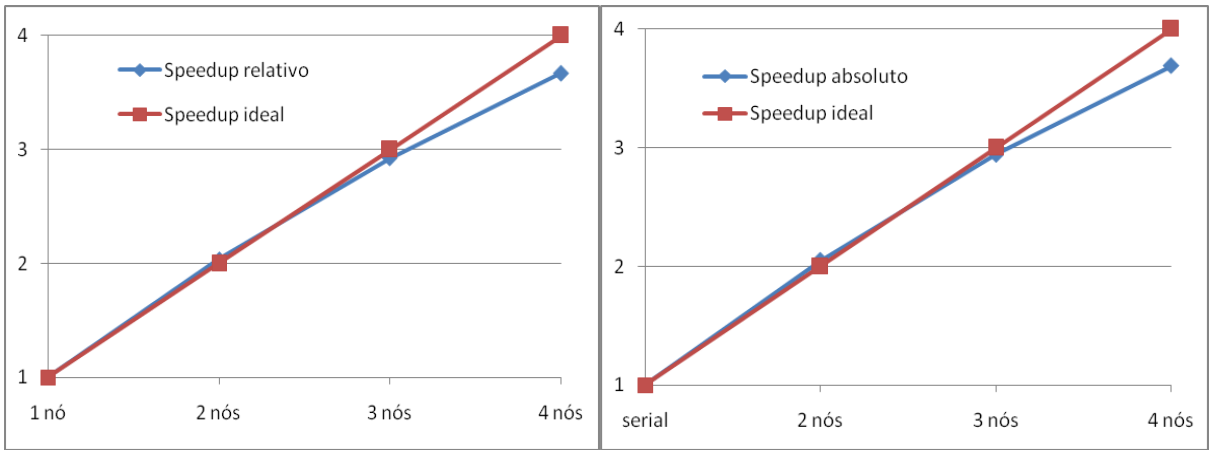

Figura 22. *Speedups* obtidos com a versão *Icantdo7Parallelv3.1.6.f90*. Como se pode observar ambos *speedups* estão bem próximos do ideal, porém já começa a existir uma pequena degradação com 4 nós. Execuções para um caso de 3248 elementos.

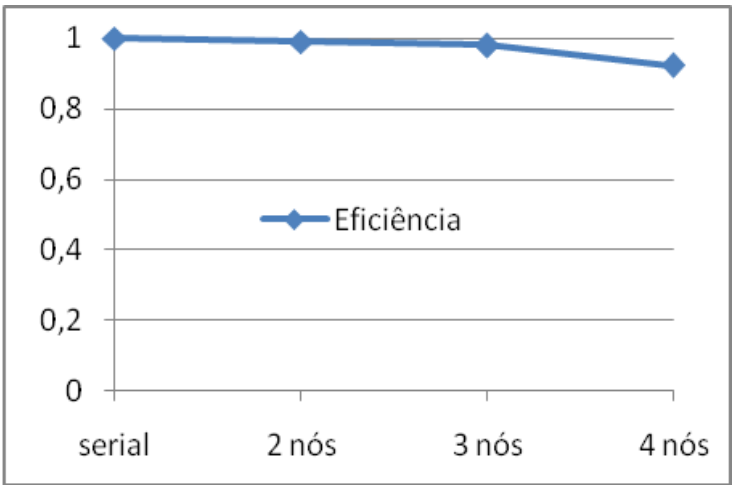

Figura 23. Eficiência do programa paralelo ICANT para até 4 nós. O cálculo da eficiência é feito dividindo-se o speedup obtido com um número de nós *p* pelo número de nós *p.*

Esta mesma análise foi realizada no cluster no CENAPAD e mostrou a tendência do *speedup* para uma configuração com até 8 nós (ver Figura 24). Vale ressaltar que a pequena degradação do speedup pra 8 nós pode ter sido causada também por causa do compartilhamento dos recursos com outros programas.

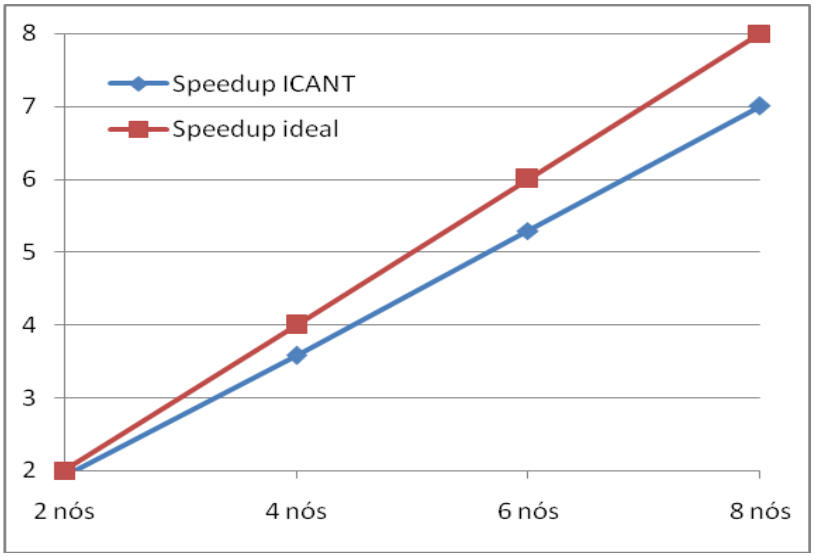

Figura 24. *Speedup* do programa ICANT obtido para um caso de 1412 elementos, demonstrando comportamento aceitável da curva de *speedup* do ICANT em relação ao *speedup* real.

A análise do ganho na utilização dos recursos de memória também foi avaliada e apresentou excelentes resultados como se pode ver na Figura 25.

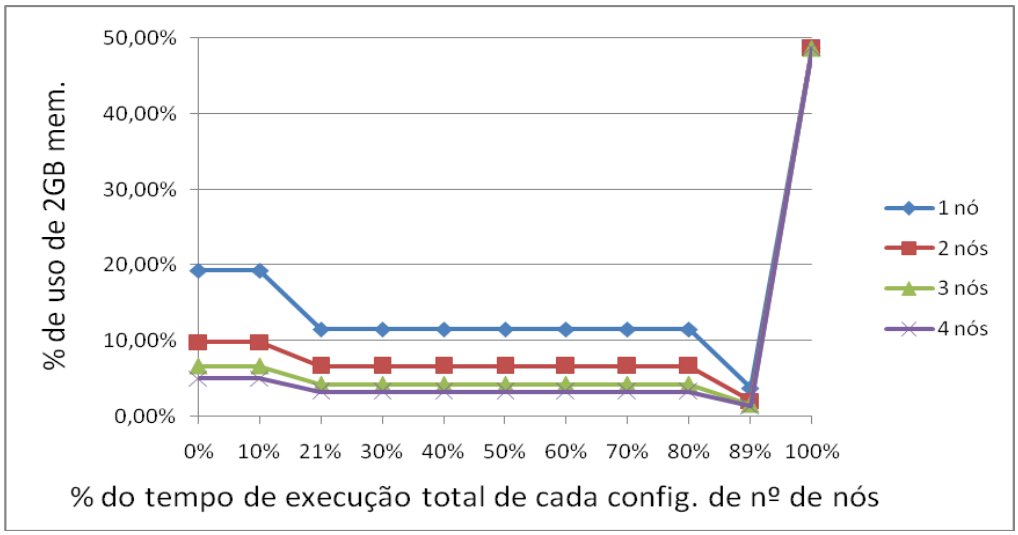

Figura 25. Uso de memória do ICANT no decorrer do tempo de execução (eixo x). O pico observado no uso de memória a partir dos 89% do tempo de execução para cada configuração de nós reflete o trecho final do código onde não foi aplicada ainda paralelização.

A reestruturação de memória no código melhorou muito o uso desse recurso chegando, nos trechos que sofreram reestruturação (até 89% no eixo *x*), a uma economia de aproximandamente 50% no uso de memória (ver Figura 26).

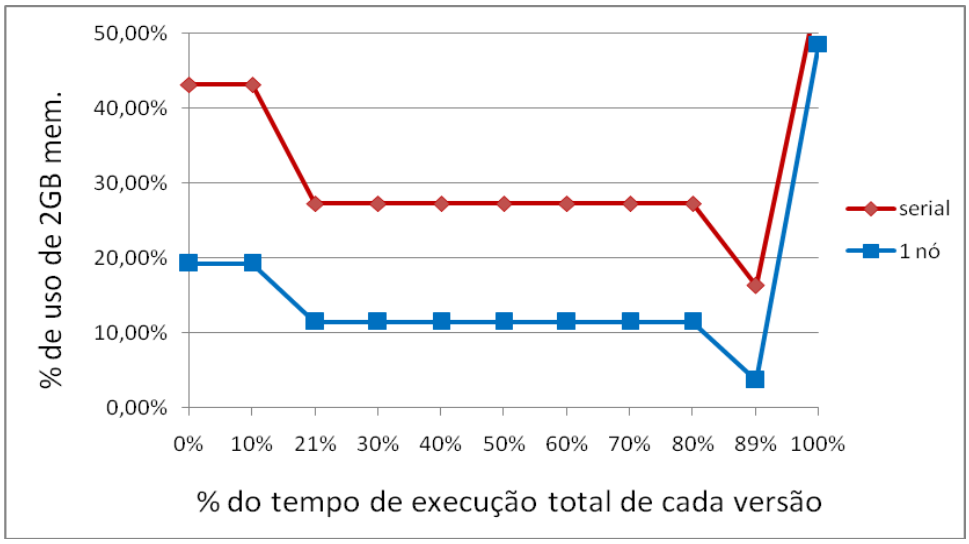

Figura 26. Comparação do uso de memória (porcentagens do eixo y relativas a 2 Gigabytes de memória) no decorrer do tempo de execução (porcentagens do eixo x) do programa ICANT nas versões serial e paralela com 1 nó.

Os resultados numéricos mantiveram uma precisão mínima de 7 casas decimais para diferentes configurações de número de nós. Na Tabela 2 pode-se observar essa variação.

Tabela 2. Variação dos resultados numéricos para cada configuração de nós para o caso de 1412 elementos executado no CENAPAD-SP.

Versão	Resultado
1 nó	Power 1A at feeders= (7.78981526769488,-109.471382733756)
2 nós	Power 1A at feeders= (7.78981526769488,-109.471382733756)
4 nós	Power 1A at feeders= (7.7898152**5623922**,-109.4713827**22397**)
8 nós	Power 1A at feeders= (7.7898152**7703488**,-109.4713827**04494**)

De forma geral os resultados foram bastante expressivos e provaram a viabilidade e o potencial de ganho com paralelização de códigos científicos usando MPI para clusters *beowulf.*

5. CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi alcançada a reestruturação e paralelização de memória e a paralelização de Entrada/Saída no programa ICANT. Foram paralelizadas as sub-rotinas: *TRJessai, IxElim, FindSym, TRJessai2 e IxBac*. Foram paralelizadas parcialmente as subrotinas *Continui e Separati*, faltando apenas as sub-rotinas finais do código *Inverse e Puissanc*. A leitura e escrita dos arquivos *TS.dat, realR.dat*, *imagR.dat* foi reescrita para MPI I/O. Resultados obtidos com a paralelização de memória são essencialmente os mesmos da versão serial com muito boa escalabilidade e ótimo ganho na economia de memória.

Devido a alguns problemas não esperados como a complexidade da paralelização de cada sub-rotina, o tempo requerido para cada paralelização foi maior que o esperado e, por conseguinte, o processo de paralelização de memória foi interrompido em 80% (faltando duas sub-rotinas para serem paralelizadas) para não exceder o prazo limite desse projeto. Devido a esse fato também algumas otimizações na comunicação do programas e outras melhorias deixaram de ser implementadas. A reestruturação de memória no código e a paralelização de memória foram requisitos completamente alcançados.

Foi criado um pequeno manual sobre como montar o ambiente de desenvolvimento utilizado neste projeto. Este se fez necessário, pois existiu a

necessidade por parte de outros grupos de computação paralela da utilização deste ambiente de reconhecida eficiência para o desenvolvimento paralelo.

Como trabalhos futuros poder-se-ia ter:

- Estudar a escalabilidade da versão paralela implementada neste projeto, executando-as com uma maior quantidade de nós;
- Otimizações na comunicação MPI do programa (minimização das mensagens);
- Compilação mais completa de um manual das estratégias adotadas e da montagem ambiente de desenvolvimento;
- Finalização do processo de paralelização de memória (sub-rotinas *Inverse* e *Puissanc*);
- Pode-se também ter a aplicação de dois níveis de paralelização juntos no ICANT (como já se estuda), sendo um em nível de clusters/máquinas (como o que foi feito neste trabalho) e o outro em nível de processadores *multicore*. Isto poderia ser feito através da integração MPI com OpenMP o que possibilitaria que em um cluster biprocessado como o do CENAPAD ou HyperThreading como o do NCGIB o programa, além ser executado em vários nós, cada nó iria utilizar todos os processadores ou *cores* disponíveis.

6. REFERÊNCIAS BIBLIOGRÁFICAS

FOLLATH, E. The New Cold War: The Global Battle for Natural Resources. In: **Spiegel Magazine**. 18 ago. 2006. Disponível em: <http://www.spiegel.de/international/spiegel/0,1518,grossbild-685811-429968,00.html>. Acesso em: 18 jun. 2008.

CRUZ, B. I. O futuro sem petróleo. In: **Diário Econômico**. 08 nov. 2006. Disponível em:<http://diarioeconomico.sapo.pt/edicion/diarioeconomico/opinion/columnistas/pt/desarrollo/706940.html>. Acesso em: 18 jun. 2008.

YOUNGQUIST, W. The Post-Petroleum Paradigm - and Population. In: **Journal Population & Environment**. ed. Springer Netherlands, ISSN 0199-0039, v. 20, n. 4. p. 297-315. 1999.

LAWSON, J. D. Power from Nuclear Fusion. In: **Nature**. v. 180, Issue 4590, p. 780-782 1957.

GALVÃO, R. M. O. Comentário sobre Proposta do Prof. Rogério Cerqueira Leite para o Programa Nuclear Brasileiro. In: **Boletim 008/2008 SBFP**. Disponível em: <http://www.sbf1.sbfisica.org.br/boletim/lemensagem1.asp?msg=39>. Acesso em: 18 jun. 2008.

AMARANTE-SEGUNDO, G. S.; *et al.* JET ITER-Like Antenna Calculations with ICANT Code. In: Encontro Brasileiro de Física da Plasmas, 2005, Niterói. **Resumos do Encontro Brasileiro de Física de Plasmas**, 2005.

CST MICROWAVE STUDIO ®, **User Manual**, Version 5.0,CST GmbH, Germany, 2004.

KOCH R.; BHATNAGAR V.P.; MESSIAEN A. M.; D. van Eester ***Com. Phys. Comm.*** **40**,1, 1986.

SWANSON D. G. **Plasma Waves**. ed Academic Press, 26, 1989.

AMARANTE-SEGUNDO, G. S.; *et al*. RF antenna analysis with the ICANT code. **Fusion Engineering and Design**, v. 81, p. 2205-2212, 2006.

AMARANTE-SEGUNDO, G. S. Modernização e Paralelização de Software Científico e Modelamento de Aquecimento de Plasmas em Tokamaks por Ondas de Radiofreqüência. **Projeto Aprovado no edital PRODOC/DCR 001/2005, CNPq-FAPESB**, 2005.

ITER**. Iter-Entreprises.** Disponível em: < http://www.iter-entreprises.com/sections/iter/projet >. Acesso em 18 jun. 2008.

SILVEIRA, P. S.; SAMPAIO G. A.; TORRES M.; VIEIRA, M. Plasma Physics Application Parallelization Ion Cyclotron Antennas. In: Escola Regional de Computação Bahia - Alagoas - Sergipe, 8., 2008, Salvador. **Anais...** Salvador: UFBA, 2008.

GROPP W.; LUSK E.; SKJELLUM A. **Using MPI: portable parallel programming with the message-passing interface**, ed. 2, Cambridge, Mass, MIT Pr. 1999.

MPI Standard 1.1. MPI: A message-passing interface standard. **Message Passing Interface Forum.** 12 jun. 1995. Disponível em: < http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html >. Acesso em: 02 dez. 2007.

MPI Standard 2.0. A message-passing interface standard. **Message Passing Interface Forum.** July 18, 1997. Disponível em: < http://www.mpi-rum.org/docs/mpi-20-html/mpi2-report.html >. Acesso em: 25 jan. 2008.

LANCELLOTTI, V. *et a.* **Nucl. Fusion 46** S476-S499 doi:10.1088/0029-5515/46/7/S10, 2006.

SILVEIRA, P. S.; Torres M.; Sampaio G. A.; VIEIRA, M. Paralelização de Aplicações Científicas Código ICANT (Ion Cyclotron Antennas). In: Workshop em Sistemas Computacionais de Alto Desempenho, 8., 2007, Gramado. **Anais…** Gramado: Hotel Serra Azul, 2007.

SOUZA M. V. V.; AMARANTE-SEGUNDO G. S. Otimização e Modernização de Software Científico para Modelagem do Aquecimento de Plasma em Tokamaks por Radiofrequência. In: Escola Regional de Computação Bahia - Alagoas - Sergipe, 7., 2007, Vitória da Conquista. **Anais...** Vitória da Conquista: UESB, 2007.

PECOUL S., **PhD thesis** Université Henri Poincaré, Nancy I, 1998.

THAKUR R. Introduction to Parallel I/O and MPI-IO. In: **11th Annual Summer Computing Institute**, 2005. Disponível em : < http://www.sdsc.edu/us/training/ workshops/institute2005/schedule.html >. Acessado em: 30 jun. 2008.