UNESP – UNIVERSIDADE ESTADUAL PAULISTA
JÚLIO MESQUITA FILHO
FACULDADE DE ENGENHARIA
CAMPUS DE BAURU

LEANDRO JOSÉ DA SILVA DE PAIVA

MODELOS DINÂMICOS SIMPLIFICADOS, PARA VERIFICAÇÃO DO COMPORTAMENTO DE REDE DE SATÉLITES AMARRADOS POR CABOS FLEXÍVEIS, USANDO-SE O MÉTODO EXPLÍCITO DAS DIFERENÇAS FINITAS

Bauru
2014

LEANDRO JOSÉ DA SILVA DE PAIVA

MODELOS DINÂMICOS SIMPLIFICADOS, PARA VERIFICAÇÃO DO COMPORTAMENTO DE REDE DE SATÉLITES AMARRADOS POR CABOS FLEXÍVEIS, USANDO-SE O MÉTODO EXPLÍCITO DAS DIFERENÇAS FINITAS

Dissertação apresentada como requisito para a obtenção do título de Mestre em Engenharia Elétrica da Universidade Estadual Paulista "Júlio de Mesquita Filho", Área de Concentração Automação.

Orientador: Prof. Tít. José Manoel Balthazar.

Bauru
2014

Paiva, Leandro José da Silva.
Modelos Dinâmicos Simplificados, Para Verificação do Comportamento de Rede de Satélites Amarrados por Cabos Flexíveis, Usando-se o Método Explícito das Diferenças Finitas / Leandro José da Silva de Paiva, 2014

114 f.

Orientador: José Manoel Balthazar

Dissertação (Mestrado)–Universidade Estadual Paulista. Faculdade de Engenharia, Bauru, 2014

1. Modelos dinâmicos. 2. Satélites amarrados. 3. Computação paralela. I. Universidade Estadual Paulista. Faculdade de Engenharia. II. Título.

ATA DA DEFESA PÚBLICA DA DISSERTAÇÃO DE MESTRADO DE LEANDRO JOSE DA SILVA DE PAIVA, DISCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, DO(A) FACULDADE DE ENGENHARIA DE BAURU.

Aos 17 dias do mês de dezembro do ano de 2014, às 15:00 horas, no(a) Sala Saturno/FUNDEB, reuniu-se a Comissão Examinadora da Defesa Pública, composta pelos seguintes membros: Prof. Dr. JOSE MANOEL BALTHAZAR do(a) Departamento de Engenharia Mecânica/UNESP/FEB, Prof. Dr. ANGELO MARCELO TUSSET do(a) Departamento de Engenharia Mecânica/Universidade Tecnológica Federal do Paraná, Prof. Dr. VINICIUS PICCIRILLO do(a) Departamento de Matemática/Universidade Tecnológica Federal do Paraná, sob a presidência do primeiro, a fim de proceder a arguição pública da DISSERTAÇÃO DE MESTRADO de LEANDRO JOSE DA SILVA DE PAIVA, intitulado "MODELOS DINÂMICOS SIMPLIFICADOS, PARA VERIFICAÇÃO DO COMPORTAMENTO DE REDE DE SATÉLITES AMARRADOS POR CABOS FLEXÍVEIS, USANDO-SE O MÉTODO EXPLÍCITO DAS DIFERENÇAS FINITAS". Após a exposição, o discente foi arguido oralmente pelos membros da Comissão Examinadora, tendo recebido o conceito final: Aprovado _____. Nada mais havendo, foi lavrada a presente ata, que, após lida e aprovada, foi assinada pelos membros da Comissão Examinadora.

Prof. Dr. JOSE MANOEL BALTHAZAR

Prof. Dr. ANGELO MARCELO TUSSET

Prof. Dr. VINICIUS PICCIRILLO

“A vida é como andar de bicicleta. Para manter o equilíbrio é preciso se manter em movimento”.

(Albert Einstein)

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Tít. José Manoel Balthazar e demais professores que trabalham com ele, pela amizade, apoio, incentivo e por importantes ensinamentos prestados, tanto científicos quanto pessoais, auxiliando assim em uma melhor formação.

Aos meus pais, Orlando Ribeiro de Paiva e Eroni Felipe da Silva, a minha noiva Mônica da Silva Carneiro, e aos demais familiares pelo incentivo e apoio na superação das dificuldades, além do carinho em família. Ao meu primo e amigo MS. Luís Manoel de Paiva Souza, por ter aberto as portas do conhecimento a mim, sem as quais não teria me desenvolvido como profissional e ser humano.

Aos meus amigos de estudos e discussões pelas longas reuniões em torno de uma mesa, que sempre trouxeram bons frutos, e pela amizade cultivada durante a realização do presente trabalho.

A todos os funcionários da Faculdade de Engenharia de Bauru que auxiliaram com documentações, dúvidas, sempre prestando serviços da melhor qualidade.

A CAPES e a Pró Reitoria da Faculdade de Engenharia, pelo apoio financeiro.

Paiva, L. J. S, **Modelos Dinâmicos Simplificados, para Verificação do Comportamento de Rede de Satélites Amarrados por Cabos Flexíveis, Usando-se o Método Explícito das Diferenças Finitas**. 2014. N f. 114 Dissertação (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia, Universidade Estadual Paulista, Bauru, 2014.

Resumo

Este trabalho apresenta um estudo sobre alguns modelos dinâmicos simplificados de satélites amarrados por cabos flexíveis, utilizando o método explícito das diferenças finitas e o método de diferença central, para a modelagem matemática dos problemas apresentados e utilizando algoritmos sequenciais e paralelos escritos em linguagem de programação C++ para a simulação destes problemas, inicialmente apresentam-se um estudo sobre satélites amarrados, computação paralela, e ambiente de troca de mensagens entre processos MPI, para em seguida avaliar os modelos simplificados programas sequenciais são criados e um estudo sobre a viabilidade da criação de programas equivalentes paralelizados é iniciado, aos problemas paralelizados utilizam-se um protocolo de troca de mensagens e o tempo necessário para alcançar a solução do mesmo problema com um programa sequencial e um programa paralelo é avaliado, aspectos como speedup e eficiência são obtidos para o programa paralelo com o objetivo de avaliar se a paralelização de determinado problema é viável.

Palavras-chave: Modelagem, Satélites, Amarrados, Computação, Paralela.

Paiva, L. J. S, **Simplified Dynamic Models for Network Behavior Verification Tethered Satellite for Flexible Cables, Using the Explicit Finite Difference Method**. 2014. 114p Dissertation (Master degree in electrical engineering) – Faculty of Engineering, São Paulo State University, Bauru, 2014.

Abstract

This paper presents a study of some simplified dynamic models of satellites tethered by flexible cables and using the explicit finite difference method, the method of central difference for the mathematical modeling of problems presented and using sequential algorithms and writings parallels in the programming language C++ for the simulation of these problems, first we present a study on tethered satellites, parallel computing, and messaging environment between processes, MPI, then to evaluate the simplified models sequential programs are created and a study on the feasibility of establishing programs equivalent parallel starts, parallel to problems using a protocol for the exchange of messages and the time required to reach a solution to the same problem with a sequential program and a parallel program is evaluated, aspects such as speedup and efficiency are obtained for the parallel program with the aim of evaluating the parallelization of a given problem is feasible.

Keywords: Modeling, Satellite, Tethered, Computing, Parallel.

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Supercomputadores, suas redes e novas técnicas de programação surgiram como aliados nas novas pesquisas e um campo que atrai a atenção de muitos pesquisadores nos tempos atuais é a engenharia aeroespacial.

Segundo Grama et al. (2003), os crescentes avanços em tecnologia possibilitaram um maior poder de processamento de informações de forma computacional, o que levou ao desenvolvimento de processadores mais robustos, tecnologias miniaturizadas, e sistemas avançados para interconectar todos esses novos equipamentos.

Pacheco (2011) demonstra que em consequência desses avanços surge um novo problema ligado à geração de energia térmica por esses equipamentos como um fator limitante ao seu crescimento. Além disso, supercomputadores possuem custo alto, o que reduz o acesso à tecnologia e limita a capacidade de processamento de um dado problema.

Como solução, o processamento paralelo é uma alternativa satisfatória, permitindo distribuir um dado problema em várias “pequenas” partes e distribui-los a vários computadores, para que no final obtenhamos um ganho no tempo utilizado para solucionar um problema.

O uso de problemas envolvendo satélites amarrados é motivado pela variedade de sistemas complexos que podem ser desenvolvidos para o estudo do comportamento desses sistemas e com isso analise de desempenho de sistemas de computação de alto desempenho. Porém a simulação desses sistemas não é algo trivial, e para isso estudos sobre técnicas de integração são necessárias, bem como o uso de computadores com alto desempenho de processamento.

1.1 Objetivos

Os objetivos deste trabalho é apresentar a utilização de um sistema de computação de alto desempenho para solucionar problemas computacionais que

exigem um número grande de cálculos do processador, assim algoritmos que antes utilizam um único processador para encontrar a solução de determinado problema deve ser dividido em vários processadores com o intuito de dividir a carga computacional e obter os resultados com algoritmos paralelos em um espaço de tempo inferior ao obtido com algoritmos sequenciais.

1.2 Metodologia

A metodologia utilizada neste trabalho para atingir os objetivos propostos consiste inicialmente da analise de trabalhos já desenvolvidos na área, com ênfase na modelagem de satélites amarrados em virtude da carga computacional exigida na simulação destes sistemas, avanços graduais nos estudos do problema foram efetuados, o trabalho apresentado evolui de um modelo de sistema simples constituído apenas de uma massa presa a um elástico e simulado de forma sequencial, para um modelo mais complexo, duas massas ligadas por um cabo, simulando dois satélites e finalmente são realizados estudos de problemas utilizando N satélites amarrados por cabos flexíveis um programa sequencial e paralelo é desenvolvido para obter dados de desempenho.

Para os problemas desta pesquisa foi necessária a construção de um algoritmo sequencial e em seguida um algoritmo paralelo para a simulação de uma grande rede de satélites, até o presente momento novas simulações estão sendo realizadas. Após a execução dos programas, obtêm-se resultados em arquivos de texto, estes contem as informações sobre a simulação do problema para que os gráficos sejam desenhados no programa Matlab.

1.3 Disposição do trabalho

Este trabalho de dissertação esta disposto da seguinte forma:

O capitulo 2 aborda os sistemas computacionais de alto desempenho, exemplificando os modelos computacionais conhecidos e formas de avaliar a velocidade de processamento e desempenho de um sistema.

Adiante no capitulo 3 o Método das Diferenças Finitas, um forma de discretização, é exposto com sua forma de dedução, seu algoritmo computacional.

O capitulo 4 destaca o padrão MPI e a biblioteca MPI, ferramentas criadas para padronizar e auxiliar programadores a desenvolver algoritmos paralelos.

No capitulo 5 é exposto o estado da arte de satélites amarrados, partindo da sua evolução histórica e pesquisas realizadas, até os trabalhos que foram desenvolvidos recentemente na área.

No capitulo 6 o trabalho segue para a modelagem matemática e simulação numérica dos problemas estudados, primeiramente é proposto um problema de uma única massa presa a um elástico, as equações são montadas com o método da diferença central e um algoritmo e um programa sequencial é criado para simular o modelo.

Em seguida, no capitulo 7, expande-se o modelo de uma única massa para duas massas (satélites) ligadas por um cabo girando em torno da orbita geossíncrona da Terra, o novo modelo é testado utilizando novamente o método da diferença central para sua modelagem e programação em linguagem C++ para desenvolver o programa sequencial e paralelo, ambos são testados e uma análise de desempenho é efetuada.

Após a analise de alguns problemas envolvendo satélites amarrados por cabos flexíveis, no capitulo 8, um novo modelo propõe um problema genérico para N satélites amarrados através de cabos flexíveis, expõem-se simulações utilizando 3 e 8 satélites amarrados e os dados de eficiência obtidos entre um programa sequencial do problema e um programa paralelo são demonstrados.

As conclusões e expectativas de trabalhos futuros em visão dos resultados obtidos com as simulações de satélites amarrados por cabos flexíveis são demonstrados no capitulo 9 e 10.

2 SISTEMA COMPUTACIONAL DE ALTO DESEMPENHO

Sistemas computacionais de alto desempenho são capazes de efetuar um número de cálculos muito maior que sistemas computacionais comuns, e por isso possuem grandes aplicações em pesquisas cientificas.

Para a execução de programas paralelos é necessário que exista comunicação e sincronização entre os processos através da troca de mensagens, assim sendo, foram desenvolvidos ambientes para troca de mensagem, bibliotecas, para facilitar o trabalho do programador.

2.1 Arquiteturas Computacionais

De forma simplificada, entende-se por computação paralela a execução simultânea de vários programas ou rotinas em diferentes processadores, com o objetivo de solucionar um único problema.

Nos computadores convencionais, um programa é um conjunto de instruções que são transmitidos ao processador de forma a serem executados sequencialmente, como visto na Figura 1.

Figura 1 - Problema sendo resolvido em série – (CENAPAD, 2012)

O conceito de computação paralela envolve a distribuição de partes desse programa para ser executado em vários processadores, estando em um ou vários computadores interligados em uma rede, visível na Figura 2.

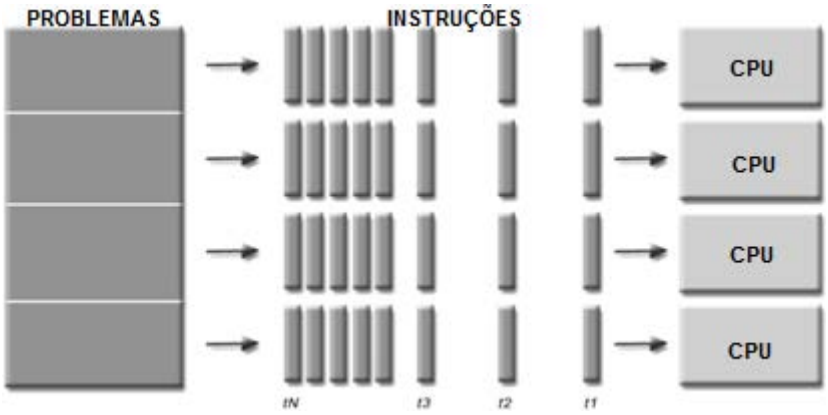

Figura 2 - Problema sendo resolvido de forma paralela – (CENAPAD, 2012)

Porém a modificar um algoritmo de um problema que será resolvido sequencialmente para um problema a ser resolvido paralelamente não é uma tarefa trivial.

Consiste da discretização do sistema utilizando o método das diferenças finitas, mais especificamente o método da diferença central para deduzir equações que substituem velocidades e acelerações existentes nas equações de movimento do sistema.

Segundo Wilkinson e Allen (2005), existe diversos tipos de arquiteturas computacionais, no caso de computadores paralelos existe uma classificação proposta por Flynn, adotado desde 1966, que permite compararmos computadores paralelos a partir de sua capacidade de lidar com instruções e dados. Essa classificação incluem quatro possibilidades:

- **SISD – *Single Instruction, Single Data*;**
- **SIMD – *Single Instruction, Multiple Data*;**
- **MISD – *Multiple Instruction, Single Data*;**
- **MIMD – *Multiple Instruction, Multiple Data*.**

SISD, Figura 3, são computadores seriais que executam uma instrução por vez, onde somente um fluxo de dados é utilizado por ciclo de CPU (*Central Processing Unit*, ou Unidade Central de Processamento), representando a maioria dos computadores com um único processador que possui apenas um núcleo.

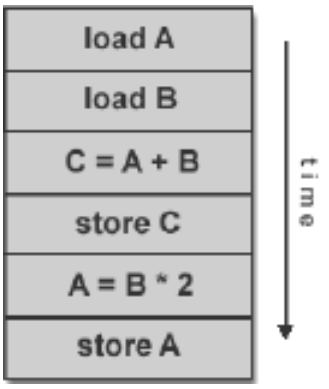

Figura 3 – SISD – Single Instruction, Single Data (CENAPAD, 2012)

SIMD, Figura 4, representa um tipo de computador paralelo, possui varias unidades centrais de processamento que executam a mesma instrução (programa), porém operando com um conjunto de dados diferente, essa arquitetura foi idealizada para aplicações que precisam processar vetores.

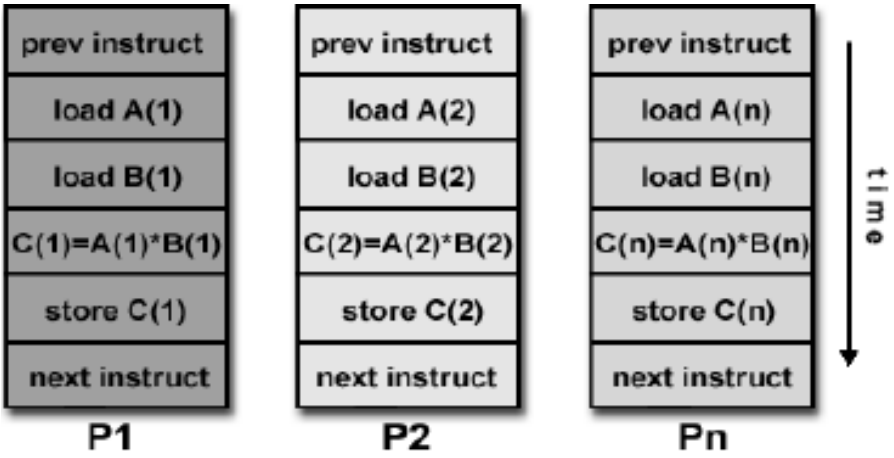

Figura 4 – SIMD – Single Instruction, Multiple Data (CENAPAD, 2012)

MISD, Figura 5, representa computadores utilizados em geral para pesquisas especificas, possuindo várias unidades centrais de processamento analisando um mesmo conjunto de dados.

Como possível aplicação pode-se citar um computador com múltiplos algoritmos de criptografia tentando decodificar uma mensagem, ou um computador com múltiplos filtros de frequência efetuando analise em um mesmo sinal (CENAPAD, 2012).

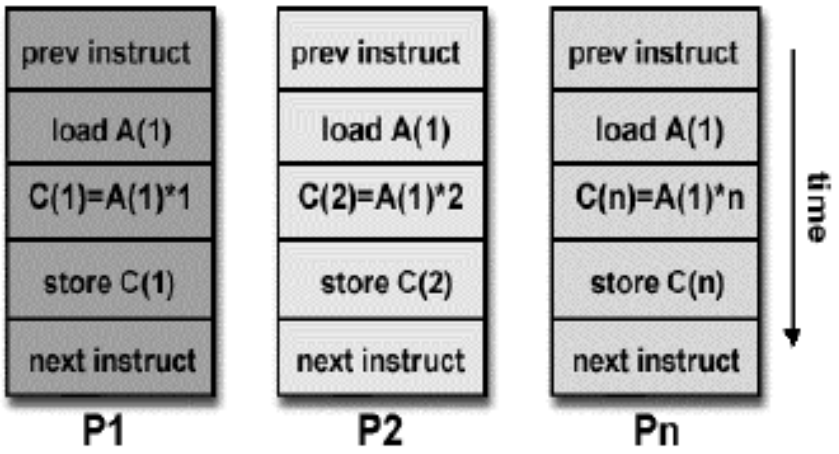

Figura 5 – MISD – Multiple Instruction, Single Data (CENAPAD, 2012)

MIMD, Figura 6, abrange o tipo de computador paralelo mais comum nos dias atuais, entre estes computadores de alto desempenho em Grid, são computadores onde cada CPU pode processar com conjuntos de instruções diferentes, nestes a execução dos programas pode ser síncrona ou assíncrona, também possuem essa arquitetura sistemas computacionais do tipo clusters.

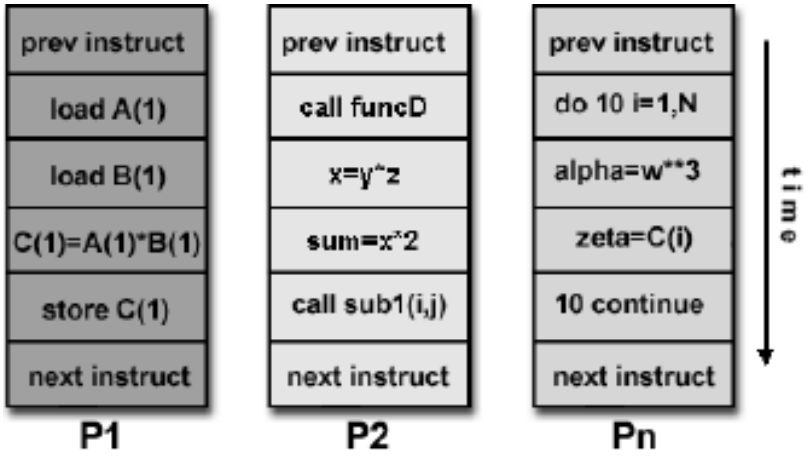

Figura 6 – MIMD – Multiple Instruction, Multiple Data (CENAPAD, 2012)

Segundo Wilkinson e Allen (2005) para compararmos o desempenho de um programa sequencial e seu equivalente paralelizado utiliza da Lei de Amdahl, que determina o potencial de aumento de velocidade com base na porcentagem de paralelismo considerando a decomposição funcional de um programa.

Existem várias formas de trabalhar com computação de alto desempenho, uma delas é utilizando Threads (um programa pode lançar processos concorrentes, onde o computador executa-os de forma simultânea, dividindo a

carga entre vários processadores existentes), e definir processamento paralelo como uma forma eficiente do processamento de informações com ênfase na exploração de eventos concorrentes no processo computacional.

Isso nada mais é do que a divisão de uma determinada aplicação para ser executada por vários elementos de processamento, ou seja, processamento paralelo é a cooperação, utilizando-se da comunicação entre os elementos e o sincronismo das mensagens, buscando eficiência através da quebra do fluxo sequencial de instruções (SAVITCH, 2004).

A plataforma Windows foi selecionada para desenvolver os programas do processamento paralelo pelo pouco desenvolvimento de pesquisa cientifica paralela nesse sistema operacional.

Foi também julgada a facilidade de trabalho com a plataforma, resultando assim, na velocidade da computação paralela aliada à simplicidade do Windows.

Sistemas do tipo Grid é o novo paradigma na área de computação distribuída, e seu nome vem da similaridade física com uma rede de energia elétrica, onde estações remotamente interligadas dividem a carga de energia exigida na rede e caso alguma falhe, as demais continuam funcionando e fornecendo energia.

Um Grid prevê mecanismos de alto desempenho para descobrir e negociar acesso a recursos geograficamente afastados, tudo isso foi possível com os recentes avanços da internet e grandes redes de computadores.

Segundo Karniadakis e Kirby (2005), os avanços na rede de computadores foram tão rápidos que chegam a dobrar a cada nove meses, isso representa o dobro da taxa prevista na Lei de Moore, isso implica que o desempenho de redes de longa distancia (WAN) aumentem em duas ordens de grandeza a cada cinco anos, um avanço considerável.

2.2 Sistemas do tipo Grid (GridUNESP)

Durante a pesquisa foi utilizada a biblioteca MS-MPI para paralelizar o problema, o equipamento utilizado para as simulações iniciais foi um notebook Dell, Pentium Core i5, com dois núcleos reais com velocidade de processamento de 2.5 GHz, simulando virtualmente quatro núcleos, 4 Gb de memória RAM rodando sistema operacional MS-Windows 7 e os programas foram feitos em Visual C++ 2010 Express, e seus gráficos foram desenhados utilizando o programa MatLab R2011a.

Com o aumento do problema, a necessidade de computadores com alto desempenho de processamento surgiu, a solução para trabalhos futuros é utilizar o cluster da UNESP, conhecido como GridUNESP (Figura 7), uma estrutura conhecida como *grid computing* (computação em grade), que interliga vários centros computacionais da universidade no estado de São Paulo, disponibilizando diversos recursos computacionais.

Figura 7 - GridUNESP (retirado de www.unesp.br/portal#!/gridunesp, acesso: 25/03/2014 às 12:51)

O sistema é composto por 8 clusters, distribuídos em 7 sede da universidade espalhadas pelo estado, vide Tabela 1, garantindo um alto desempenho de processamento, o que permite a simulação de modelos complexos em um curto espaço de tempo.

Tabela 1 - Infraestrutura do GridUNESP

CLUSTER CENTRAL	CLUSTERS SECUNDÁRIOS
Processamento • 256 servidores SUN X4150 • 2048 cores (Intel Xeon 2.83 GHz) • Interconexão Infinitiband 4X DDR (20 Gbps) **Armazenamento** • 36 TB via DAS de fibra ótica (StorageTek 6140) • 96 TB em 4 servidores SUN X4500 **4 Servidores** • SUN X4250, com 16 cores, 32 GB e HD SAS	**7 Clusters contendo** • 16 Servidores de processamento (128 cores) – SUN X4150 **2 Servidores** • SUN 4250 • 12TB em DAS FC - StorageTek

Sistemas do tipo Grid é o novo paradigma na área de computação distribuída, e seu nome vem da similaridade física com uma rede de energia elétrica, onde estações remotamente interligadas dividem a carga de energia exigida na rede e caso alguma falhe, as demais continuam funcionando e fornecendo energia.

Um Grid prevê mecanismos de alto desempenho para descobrir e negociar acesso a recursos geograficamente afastados, tudo isso foi possível com os recentes avanços da internet e grandes redes de computadores.

Segundo Karniadakis e Kirby (2005), os avanços na rede de computadores foram tão rápidos que chegam a dobrar a cada nove meses, isso representa o dobro da taxa prevista na Lei de Moore, isso implica que o desempenho de redes de longa distancia (WAN) aumentem em duas ordens de grandeza a cada cinco anos, um avanço considerável.

O GridUNESP é baseado no código fonte aberto Globus Toolkit, mantido por comunidades de software livre, o sistema fornece infraestrutura de hardware e software que implementa protocolos para identificação segura, alocação e liberação de recursos do sistema.

Permitindo acesso remoto ao servidor, um programa que é executado de forma continua e tem por objetivo lidar com solicitações de serviços, por clientes, estações de trabalho, computadores pessoais que através de identificação na rede de computadores tem acesso aos recursos do Grid.

2.3 Velocidade de Processamento e Desempenho

Segundo Wilkinson e Allen (2005), dentre os vários pontos interessantes existentes na busca por soluções utilizando programas paralelos, determinar o quanto mais rápido ficaria para encontrar a solução de um problema utilizando computação paralela é um ponto fundamental.

Isso pode ser obtido pela comparação do tempo que foi gasto para resolver um problema com o melhor algoritmo sequencial com o tempo gasto para resolver o mesmo problema utilizando um algoritmo paralelo.

2.3.1 Speedup

Este valor obtido é conhecido como Speedup, S(p), ou “fator de aumento de velocidade”, sua formulação pode ser vista na Equação 1.

$$S(p) = \frac{T_S}{T_P} \quad (1)$$

Com T_S representando o tempo necessário para solucionar o problema utilizando um algoritmo sequencial e T_P representando o tempo necessário para solucionar o problema utilizando um algoritmo paralelo.

Como basicamente algoritmos sequenciais utilizam-se apenas de um processador, enquanto que, algoritmos paralelos utilizam p processadores, em uma análise teórica, o Speedup pode ser denotado em termos de passos computacionais, como visto na Equação 2.

$$S(p) = \frac{Passos\_computacionais\_com\_um\_processador}{Passos\_computacionais\_com\_p\_processadores} \quad (2)$$

Para cálculos sequenciais, podemos usar complexidade de tempo, sendo que isso pode ser estendido para algoritmos paralelos e aplicados o Speedup é necessário considerar o tempo de comunicação, que geralmente é muito superior do que o tempo de processamento.

Teoricamente o aumento máximo de velocidade na solução de determinado problema é geralmente p com p processadores (Speedup linear).

O aumento de velocidade, Speedup, de p seria obtido quando o cálculo pode ser dividido em processos de duração igual, com um processo mapeado para cada processador e sem sobrecarga adicional na solução paralelizada.

2.3.2 Eficiência

A eficiência, E, é obtida pela diferença do tempo de execução de um programa utilizando-se de um processador pelo tempo de execução de um programa paralelo multiplicado pelo número de processadores como visto na Equação 3.

$$E = \frac{T_S}{T_P * p} \quad (3)$$

Com T_S sendo o tempo de execução utilizando um processador, T_P o tempo de execução utilizando vários processadores e p o número de

processadores. A transformação da Equação 3 permite obter o valor da eficiência, E, como uma porcentagem, como visto na Equação 4.

$$E = \frac{S(p)}{p} * 100\% \quad (4)$$

Como exemplo, a obtenção de um valor de eficiência, E = 50%, denotaria que os processadores envolvidos no algoritmo paralelo para solução de um problema estão utilizando metade do tempo que um algoritmo sequencial levaria.

3 Protocolo de Troca de Mensagens (MPI - Message Passing Interface)

Karniadakis e Kirby (2005) observa que com os avanços em pesquisa na área de computação paralela surgiu a necessidade da criação de uma padronização, visando portabilidade, pois até então cada grupo de programadores desenvolvia sua forma de comunicação entre os sistemas.

Surgiu então, em 1994, depois de dois anos de pesquisa, o padrão MPI (*Message Passing Interface*), um ambiente de troca de mensagem padronizado e portável para qualquer arquitetura.

Durante o processo de padronização do MPI, desenvolve-se junto o MPICH, uma implementação que proveu meios de testar as funcionalidades do padrão.

Mais tarde o padrão MPI avançou para sua segunda versão, seguindo os avanços o MPICH2 foi lançado, já compatível com sua primeira versão para manter a portabilidade.

Para sistemas Microsoft Windows existe o MS-MPI, criado pela empresa com base no MPICH2, portanto sendo, compatível com o ambiente. A Tabela 2 descreve as funcionalidades do padrão MPI.

Tabela 2 – Funcionalidade do padrão MPI.

Comunicação ponto-a-ponto e coletiva	O MPI permite todos os tipos de comunicação, bufferizada, não-bufferizada, bloqueante, não-bloqueante.
Suporte a grupos de processos	O MPI relaciona os processos em grupos e os identifica pela sua classificação dentro destes denominado *rank*.
Suporte para contextos de comunicação	Contextos são escopos que relacionam um determinado grupo de processos, assim sendo, determinam quais grupos de processos podem receber determinada mensagem. Dessa forma um grupo de processos só é capaz de comunicar-se dentro do seu próprio contexto.
Suporte a diversas topologias	O MPI fornece ferramentas que permitem o programador definir a estrutura topológica com a qual os processos de um determinado grupo se relacionam.

3.1 A Biblioteca MPI

Segundo Paiva et al. (2014), a execução do padrão MPI em um programa tem utilizado um único código fonte, que quando compilado gera um único programa executável que é executado por todos os processos da aplicação. Pode-se definir o que cada processo do programa faz utilizando estruturas condicionais do tipo *if* para cada *rank*. Uma simples demonstração dessa estrutura pode ser visto na Figura 8.

```
if ( rank == 0 )
{
    função_A;
}
if ( rank == 1 )
{
    função_B;
}
if ( rank == 2 )
{
    função_C;
}
```

Figura 8 - Exemplo de estrutura condicional para MPI

Karniadakis e Kirby (2005), diz que a biblioteca MS-MPI permite ao programador o desenvolvimento de programas que utilizam o padrão MPI em linguagem de programação C, C++ e FORTRAN, possui compatibilidade com MPICH2 e mais de 160 funções.

Integra-se com o Microsoft Visual C++ 2010 e sua biblioteca pode ser chamada facilmente indicando a pasta onde esta localizada, para executar o código fonte, basta compilarmos e executarmos a seguinte linha de comando pelo console do Microsoft Windows dentro da pasta que esta o programa executável:

mpiexec –n 4 <nome do programa>.exe.

Esse procedimento é utilizado para simulações em um único computador que possui 4 núcleos.

Assumindo a utilização do compilador de código aberto GNU g++ para compilar os programas escritos em C++, ou seja, com extensão .cpp.

Estando conectado a uma máquina que possui os compiladores necessários executar as seguintes instruções para cada caso:

Para o uso apenas de bibliotecas nativas da linguagem C++ com MPI:

g++ -o <meuprograma> <meuprograma.cpp> -lmpi

Para o caso de uso da biblioteca MPI em conjunto com a biblioteca Math:

g++ -o <meuprograma> <meuprograma.cpp> -lmath –lmpi

No caso de se utilizar outras bibliotecas não nativas a linguagem C++ que devem ser definidas pelo programador:

g++ -o <meuprograma> <meuprograma.cpp> -I/users/kirby/includes –L/users/kirby/libs –lSCmathlib –lmath –lmpi

As letras I e L são respectivamente o local de origem onde estão os arquivos de cabeçalho e bibliotecas (header files e libraries) que serão incluídos e o local de destino onde esses serão incluídos.

As funções básicas utilizadas na programação paralela são descritas na Tabela 3.

Tabela 3 - Funções básicas utilizadas pela biblioteca MPI

MPI_Init	Estabelece o ambiente necessário para troca de mensagens e a sincronização de todos os processos, é o primeiro argumento utilizado para inicializar rotinas dentro do ambiente MPI. Sendo que todas as linhas do programa escritas antes desse comando não fazem parte do ambiente MPI.
MPI_Comm_rank	Utilizada para atribuir a uma variável o atual identificador de um processo dentro de um comunicador.
MPI_Comm_size	Retorna o número de processos MPI existentes dentro de um grupo de rotinas.
MPI_Send	Utilizado para enviar uma mensagem no MPI.
MPI_Recv	Usado para o recebimento de uma mensagem no MPI.
MPI_Bcast	Esta rotina permite a um processo o envio de mensagem para todos os processos contidos em um mesmo grupo.
MPI_Reduce	Esta função permite que os resultados parciais efetuados por cada processo de um grupo sejam combinados e enviados a um único processo.
MPI_Finalize	Encerra o ambiente de troca de mensagem MPI, toda linha de código escrita após esse comando não faz parte de um ambiente MPI.

4 MÉTODO EXPLÍCITO DAS DIFERENÇAS FINITAS

Segundo Burder (2001), a ideia principal no método de diferenças finitas é usar aproximações de derivadas. Dessa forma, a equação diferencial de movimento governante e as condições de contorno associadas, se aplicáveis, são substituídas pelas equações de diferenças finitas correspondentes.

O método consiste de uma abordagem numérica para integração de equações diferencias que tem como característica principal satisfazer as equações diferenciais governantes em intervalos de tempo discretos.

O sistema resultante em termos de deslocamentos é algébrico, esparso e não acoplado o que favorece enormemente sua implementação em computação paralela (SIMIONI; BALTHAZAR; BRASIL, 2007).

Substitui-se então o domínio da solução por um numero finitos de pontos em uma malha e procuramos determinar os valores da solução desejada nesses pontos.

Segundo Rao (2008), substituímos o domínio da solução no tempo por um numero finito (discreto) de pontos em uma malha, procuramos assim determinar os valores da solução para cada ponto, estabelecemos que cada ponto da grade possua a mesma distancia um do outro, assim por meio de expansões em série de Taylor expressamos x_{i+1} e x_{i-1}, como na Figura 9.

$$x_{i+1} = x_i + h\dot{x}_i + \frac{h^2}{2}\ddot{x}_i + \frac{h^3}{6}\dddot{x}_i + \cdots$$

$$x_{i-1} = x_i - h\dot{x}_i + \frac{h^2}{2}\ddot{x}_i - \frac{h^3}{6}\dddot{x}_i + \cdots$$

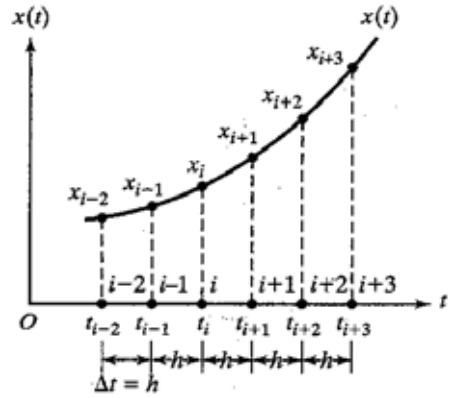

Figura 9 - Pontos de grade (RAO, 2008)

As formulas são deduzidas utilizando séries de Taylor, sendo três tipos, ou formulas resultantes: diferença para trás, diferença para frente e diferença central. A terceira é conhecida por ser mais precisa e por isso é mais utilizada.

Na Figura 9, h representa a variação de tempo, para obter a diferença central em um sistema com um grau de liberdade, obtêm-se das expansões em série de Taylor os dois primeiros termos e subtraindo-os temos a aproximação de diferença central pelo método das diferenças finitas até a derivada de primeira ordem de x em $t = t_i$ pela equação (5).

$$\dot{X}_i = \frac{X_{i+1} - X_{i-1}}{2h} \tag{5}$$

Então, pela soma dos dois primeiros obtemos a derivada de segunda ordem pela equação (6).

$$\ddot{X}_i = \frac{X_{i+1} - 2X_i + X_{i-1}}{h^2} \tag{6}$$

Utilizando como exemplo uma equação (7) governante do movimento de um sistema com um grau de liberdade que possui amortecimento viscoso.

$$m\,(\ddot{X}_i) + \; c(\dot{X}_i) \; + \; kX_i \; = \; F\,(t) \tag{7}$$

Determina-se que o espaço de tempo da solução seja dividido em partes iguais, também determina-se que as condições iniciais do sistema são dadas por $x(t = 0) = x_0$ e $x'(t = 0) = x'_0$.

Substitui-se as derivadas obtidas da soma e diferença das expansões em série de Taylor, escrevendo-as no ponto na grade i, da Figura 9 e a equação (8) fica:

$$m\,(\frac{X_{i+1} - 2X_i + X_{i-1}}{h^2}) + \; c(\frac{X_{i+1} - X_{i-1}}{2h}) \; + \; kX_i \; = \; F\,(t_i) \tag{8}$$

Organizando a equação (8), o primeiro passo no tempo x_{i+1} é isolado para que possa ser obtida a equação (9) resultante, que é conhecida como formula de recorrência (pois necessita de um passo anterior em conjunto com o atual para que o próximo passo no tempo seja calculado).

$$X_{i+1} = \frac{\left(2m/h^2\right)X_i + \left(c/2h - m/h^2\right)X_{i-1} - F\,(t_i)}{\left(m/h^2 + c/2h\right)} \tag{9}$$

A fórmula de recorrência não é auto iniciável e por isso é necessário calcular o passo anterior inicial pelas equações (10) e (11) utilizando as condições iniciais dadas.

$$\ddot{x}_0 = \left(\frac{F\ (t=0) - \left(c\dot{x}_0\right) - kx_0}{m} \right) \tag{10}$$

$$x_{-1} = x_0 + \left(\dot{x}_0 h\right) + \left(\frac{1}{2} \ddot{x}_0 h^2 \right) \tag{11}$$

O algoritmo contendo os passos para solucionar um problema utilizando o método da diferença central é descrito na Tabela 4.

Tabela 4 – Algoritmo dos passos para uso do Método de Diferença Central

ALGORITMO SEQUENCIAL
Passo 1: Obtém-se as expansões em série de Taylor um passo a frente e outra atrás na malha. **Passo 2:** . Efetua-se a soma e subtração dessas expansões para determinar os termos que vamos discretizar, velocidades e acelerações. **Passo 3:** Substitui-se as velocidades e acelerações da equação de movimento do sistemas por aproximações que foram obtidas com a série de Taylor. **Passo 4:** Manipula-se as equações para isolar o próximo passo e assim obter a fórmula de recorrência do sistema. **Passo 5:** Para que o sistema possa calcular seu primeiro passo é necessário saber o passo anterior, o histórico do momento x(t) e x(t-1), então devem ser deduzidos o passo anterior determinando as condições iniciais como x(0) = 0 x’(0) =0. **Passo 6:** Calcula-se a aceleração inicial. **Passo 7:** Escreva o segundo passo. **Passo 8:** Repita até que alcance o número de passos desejados na simulação do sistema. **Passo 9:** Atualiza-se o passo. **Passo 10:** Escreva a nova posição (essa informação será reutilizada a cada passo). **Passo 11:** Fim.

5 SATÉLITES AMARRADOS POR CABOS FLEXÍVEIS

Segundo Ellis (2010), o conceito de satélites amarrados foi inicialmente proposto por Tsiolkovsky em 1895, em seu trabalho ele propôs uma forma de gerar gravidade artificial, envolvendo a ligação de um satélite a um contra peso e girando o sistema, porém a primeira aplicação pratica de um satélite amarrado só foi possível durante o programa Gemini da NASA na década de 1960, quando uma espaçonave, de nome Gemini, foi conectada a um veículo espacial alvo por um cabo de 30 metros.

O objetivo de ligar os dois era testar no espaço manobras de atracagem, bem como a possibilidade de gerar gravidade, como proposto por Tsiolkovsky.

Em 1975, novos experimentos foram realizados com satélites amarrados, Colombo propôs implantar uma sonda a partir de um ônibus espacial, este ainda esta sendo desenvolvido, e o cabo possuiria 100 km de extensão, este poderia ser utilizado para realizar medições eletromagnéticas na magnetosfera terrestre, tal ideia abriu caminho para novas missões reais com satélites amarrados nos anos 80 (ELLIS, 2010).

Uma parceria entre o Japão e Estados Unidos lançou uma série de experimentos com foguetes-sonda, na época conhecidos como *Tethered Payload Experiment* (TPE), basicamente o experimento consistia na separação do foguete da sonda em determinado ponto do voo, porém mantinham-se conectados por um cabo.

Várias experiências foram realizadas nessas condições, todas relacionadas as propriedades eletrodinâmicas do sistema, três diferentes voos foram realizados com sucesso, onde dados foram obtidos, o ultimo, em 1983, o cabo que interligava o sistema alcançou 418 metros de comprimento.

Em 1989 a agencia espacial canadense lançou o OEDIPUS-A, o objetivo da missão era usar um sistema de satélites amarrados para fazer medições do

campo magnético da Terra na ionosfera, o sistema consistia de duas massas ligadas por um cabo, com comprimento de 958 metros durante a missão (posteriormente em 1995 a agencia espacial canadense lançou o OEDIPUS-C, com um cabo com 1174 km de comprimento, sua missão tinha objetivos similares da OEDIPUS-A, mas também envolvia experimentos com a dinâmica envolvida em satélites amarrados, a teoria e ferramentas computacionais envolvidas com esse sistema).

Segundo Ellis (2010), em 1992, com a TSS-1, e novamente em 1996, com a TSS-1R, a Agência Espacial Americana (NASA) em conjunto com a agência espacial italiana (ASI) testou o conceito de satélites amarrados (*Tethered Satellite*), também conhecidos como TSS, que apresentava um satélite menor ligado por um cabo a um satélite maior, como visto na Figura 10.

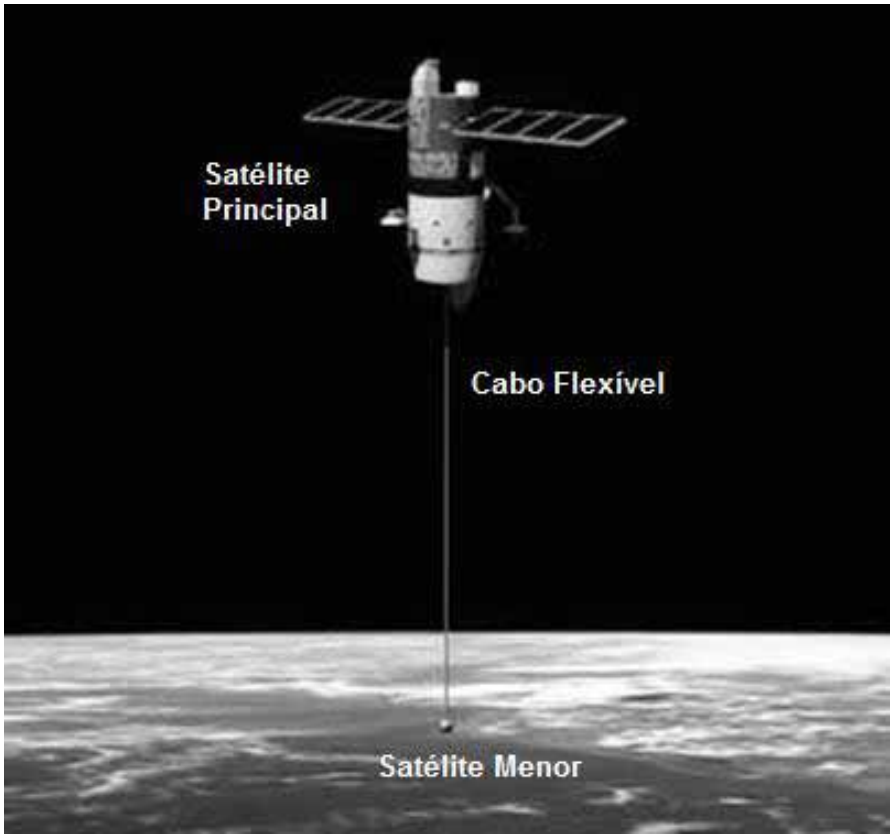

Figura 10 – Satélites amarrados por cabo flexível (NASA, retirado de http://www.nasa.gov/vision/universe/roboticexplorers/tethered_spacecraft.html, acesso: 22/01/2015 às 14:45)

Ellis (2010) descreve que na época a missão tinha como premissa o estudo da estabilização, da recuperação de satélites, da implantação de satélites no espaço e operação de sistemas elétricos de condução dentro do campo magnético da Terra. (COSMO; LORENZINI, 1997).

Beletsky e Levin (1993) demonstram que a manipulação de dois satélites interligados por um cabo flexível acabou por ser um desafio único de engenharia, pois ali surgiram diversos problemas relacionados à gravidade, força centrífuga, resistência atmosférica de acordo com a altura do satélite menor e a influencia que isso causava ao sistema como um todo.

Um dos problemas que pode ser solucionado com o uso de sistemas contendo dois satélites amarrados surgiu com o passar dos anos, depois de mais de 50 anos de missões espaciais, satélites desativados, danificados, estágios de foguetes de missões anteriores em baixa órbita tornaram-se detritos espaciais, alguns movendo-se em alta velocidade em torno da Terra.

Tais detritos representam risco real a missões já existentes no espaço, bem como futuras missões, representam risco a vida de astronautas em missões tripuladas em vista do grande potencial de danos que podem causar, isso devido a velocidade que se movem na orbita terrestre.

Ishige, Kawamoto e Kibe (2004) demonstra em seu trabalho o uso de um sistema contendo dois satélites amarrados por cabo flexível para a remoção de detritos espaciais, Figura 11, utilizando a eletrodinâmica do sistema, devido à força de Lorentz existente no cabo flexível com a sua interação com o campo geomagnético terrestre, para transferência de orbita e assim baixar a orbita de detritos espaciais.

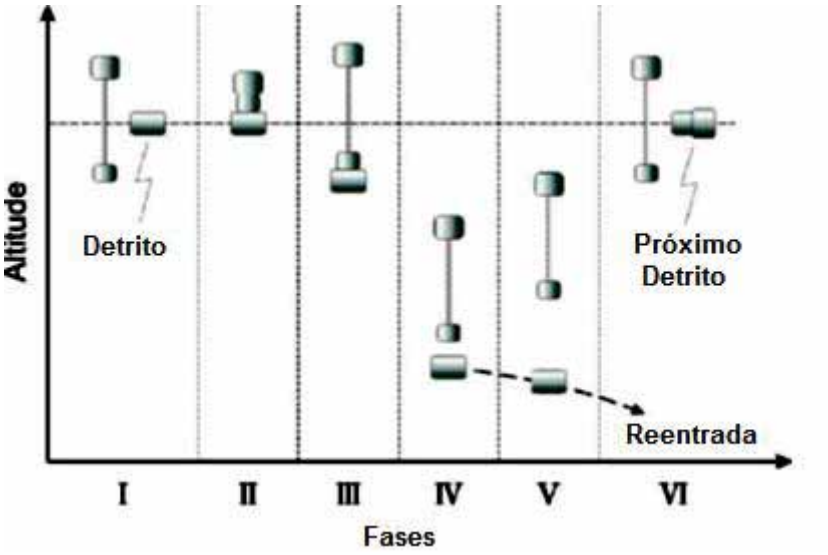

Figura 11 - Missão de detritos espaciais (ISHIGE; KAWAMOTO; KIBE, 2004)

O processo proposto envolve fases, Figura 11, o sistema é composto por dois satélites, uma maior e principal, e um menor para captura de detritos, e a extensão do cabo flexível é variável.

Basicamente os detritos estão contidos em uma orbita e o centro de massa do sistema alinha-se na mesma orbita ao aproximar-se dos detritos o cabo flexível é recolhido para que ocorra a captura do lixo espacial pelo satélite menor (I), o cabo é novamente estendido (III), o sistema de satélites perde altitude devido ao aumento da massa na região do satélite menor devido aos detritos capturados, com o sistema de satélites em uma orbita mais baixa, os detritos são liberados e destruídos na reentrada da atmosfera (IV, V, VI), enquanto que os dois satélites voltam a ganhar altitude e se equilibram novamente em uma orbita (V), o sistema esta pronto para capturar então novos detritos (VI).

A ideia prova uma forma eficiente de aproveitamento da capacidade de uma rede de satélites amarrados, o que a primeira vista, pode gerar economia e reduzir a produção de lixo espacial (satélites desativados, estágios de foguetes, Figura 12), em vista de uma forma mais eficiente de dispersão de informações, um problema que vem sendo muito discutido por pesquisadores nos dias de hoje.

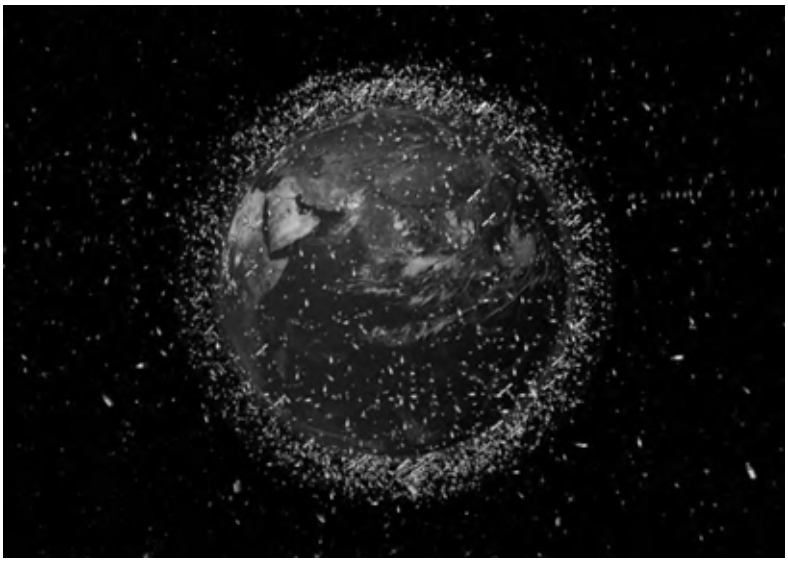

Figura 12 - Esboço de satélites artificiais ativos, inativos e estágios de foguetes ao redor da Terra. (PELT, 2009)

Yamaigiwa, Hiragi e Kishimoto, (2005) realizou estudo semelhante, com os mesmos objetivos de estudar o comportamento eletrodinâmico de satélites amarrados para baixar a orbita de detritos espaciais, em seu experimento o cabo (*Tether*) é considerado como massas segmentadas, como partículas conectadas por uma mola e um amortecimento viscoso, para assim simular um cabo flexível. Isso é visível na Figura 13, onde o lastro (*Ballast*, considerado o satélite menor) conecta-se a várias secções que se estendem até a nave mãe, que se conectam pela mola e o amortecedor viscoso até alcançar o satélite maior (*Mother Ship*, a nave mãe).

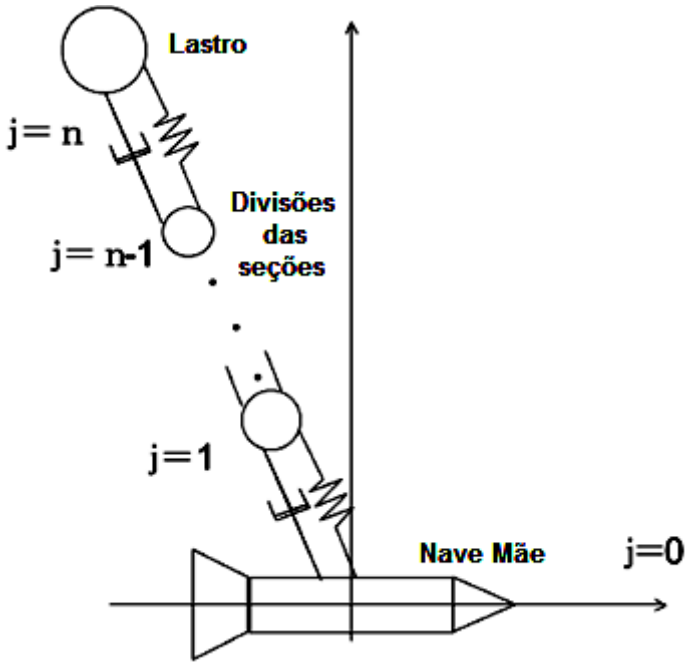

Figura 13 - Modelo onde o Tether é considerado uma sequencia de partículas interligadas por molas e amortecedores viscosos (YAMAIGIWA; HIRAGI; KISHIMOTO, 2005)

Os modelos propostos por Brasil, Fallara (2013) expõem o uso do método de diferenças finitas para estudar primeiramente dois cabos colineares engastados com uma massa disposta em uma das pontas deles, e o problema de três satélites amarrados dispostos de tal forma que configuram um triangulo equilátero.

Estes modelos apresentam significativa contribuição ao trabalho, sendo que o desenvolvimento das equações diferenciais de movimento do sistema pelo método da diferença central é semelhante por tratar-se de modelos equivalentes.

O problema com estruturas de cabos é muito estudado na área de engenharia civil, posteriormente tais estudos evoluíram para a área da engenharia aeroespacial com satélites amarrados.

O grande desafio encontrado é o material que o cabo é desenvolvido, pois o mesmo tem que ter a capacidade de resistir a grandes cargas tensionais, o uso de cabos compostos por nano tubos de carbono que poderiam solucionar este problema, pela sua grande resistência.

O primeiro modelo, Figura 14, apresenta dois cabos flexíveis de comprimento L engastados em uma de suas pontas a um ponto fixo e na outra a uma massa, essa esta sujeita a uma força peso P, nesse sistema a massa deve apresentar um deslocamento vertical, e a resposta para o deslocamento da massa é similar ao deslocamento de uma massa presa a um cabo flexível.

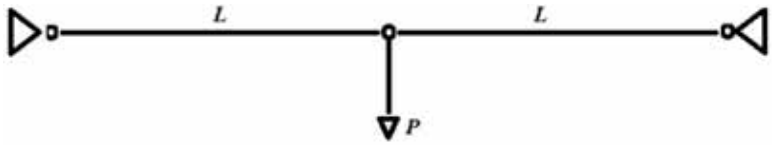

Figura 14 – Modelo com carga vertical (BRASIL; FALLARA, 2013)

A resposta obtida do primeiro problema, Figura 14, foi o deslocamento da única massa existente verticalmente durante um período de tempo, devido ao amortecimento esta cessa movimento.

O deslocamento em função do tempo da massa presa aos dois cabos flexíveis pode ser visto na Figura 15.

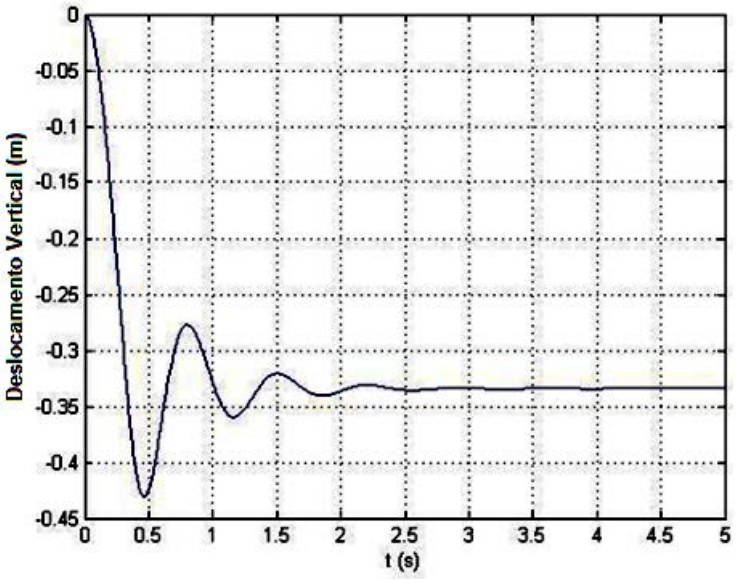

Figura 15 - Deslocamento vertical de uma massa em função do tempo (BRASIL; FALLARA, 2013)

O problema é interessante por apresentar não linearidades presentes nas forças restauradoras presentes nos cabos flexíveis, e sistemas equivalentes podem ser desenvolvidos para estudo.

O segundo modelo, Figura 16, apresenta três massas, M1, M2 e M3 conectadas entre si por cabos flexíveis com a mesma extensão configurando um triangulo equilátero, além disso, cada massa possui um cabo flexível engastado com comprimento L.

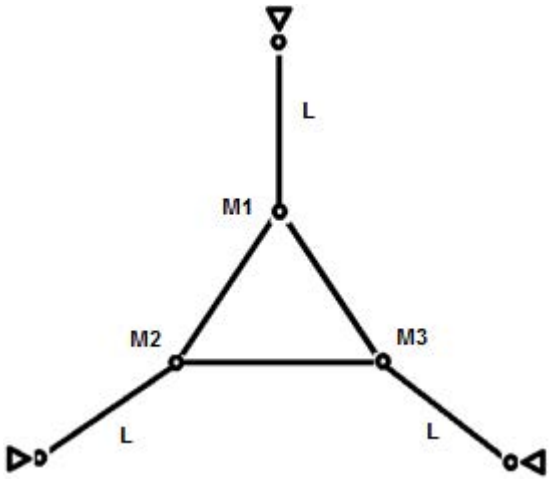

Figura 16 - Modelo com três massas (BRASIL; FALLARA, 2013)

No segundo problema (Figura 16), foi obtido o deslocamento vertical e horizontal de uma das massas presentes no sistema, estes deslocamentos novamente apresentam respostas não lineares e seu comportamento é visível na Figura 17.

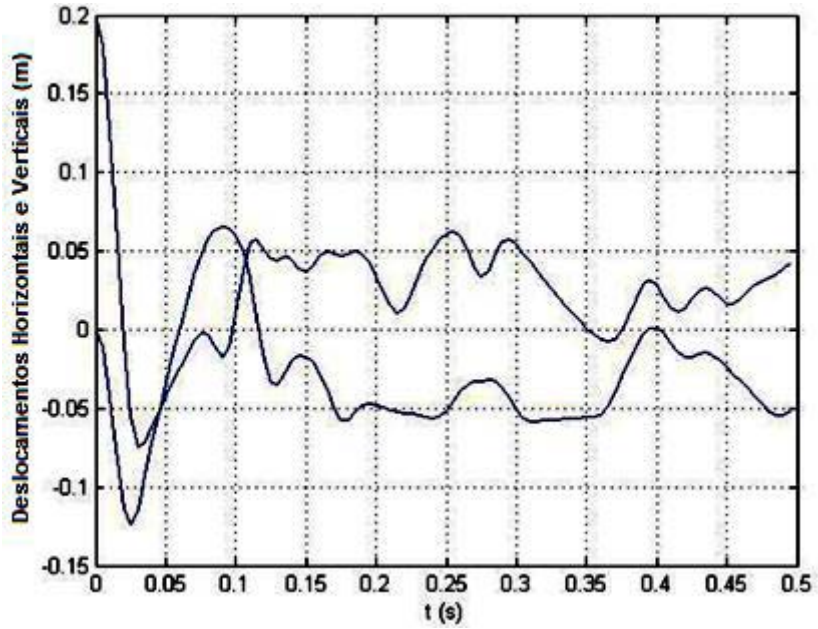

Figura 17 - Resposta do deslocamento no tempo de uma das massas para o problema envolvendo três massas (BRASIL; FALLARA, 2013)

Esses dois modelos representam grande contribuição ao trabalho como um todo, demonstrando que o uso do método das diferenças finitas é eficiente para solucionar problemas contínuos que necessitam de discretização para obter o comportamento do sistema no tempo.

Segundo Pelt (2009), o estudo do problema de satélites amarrados torna-se bem popular na órbita geossíncrona da Terra, orbita onde o satélite permanece parado em relação à Terra, por ser um ambiente estável para esses estudos, o uso de três satélites nessa orbita, seguindo na linha do equador, poderia ser utilizado para observar permanentemente o planeta, meteorologia, ou agir com um sistema de comunicação, rádio e televisão, o que abrangeria todo o globo terrestre, como na Figura 18.

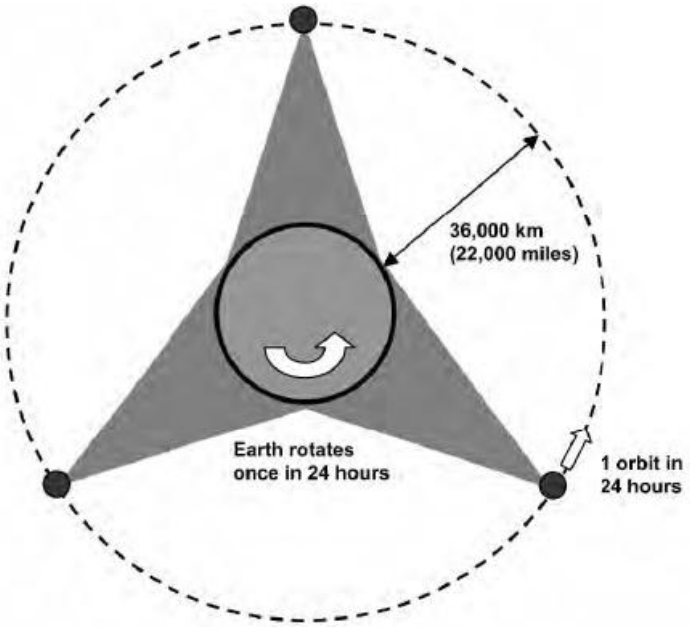

Figura 18 - Três satélites amarrados movendo-se na órbita geossíncrona da Terra. Space Tethers and Space Elevator (PELT, 2009)

A ideia prove uma forma eficiente de aproveitamento da capacidade de uma rede de satélites amarrados, o que a primeira vista, pode gerar economia e reduzir a produção de lixo espacial (satélites desativados, estágios de foguetes), em vista de uma forma mais eficiente de dispersão de informações, um problema que vem sendo muito discutido por pesquisadores nos dias de hoje.

6 MODELO MATEMÁTICO DE UMA MASSA PRESA A UM CABO FLEXÍVEL

Inicialmente trabalhou-se com um modelo simples de um grau de liberdade, uma massa presa a um cabo flexível, um algoritmo sequencial em C++ foi escrito para estudar o sistema, a simplicidade do modelo permitiu iniciar os estudos com diferenças finitas, pois o cabo flexível apresenta alterações em seu comprimento enquanto o sistema progride no tempo.

Como restrição ao sistema, determinou-se que o mesmo só teria deslocamentos na direção vertical, ou seja, um grau de liberdade, o sistema estaria inicialmente em repouso, e após um determinado tempo a massa seria solta e iniciaria movimento, a previsão é que o deslocamento deve cessar após um tempo devido à ação do amortecimento. Um esboço do sistema pode ser visto na Figura 19.

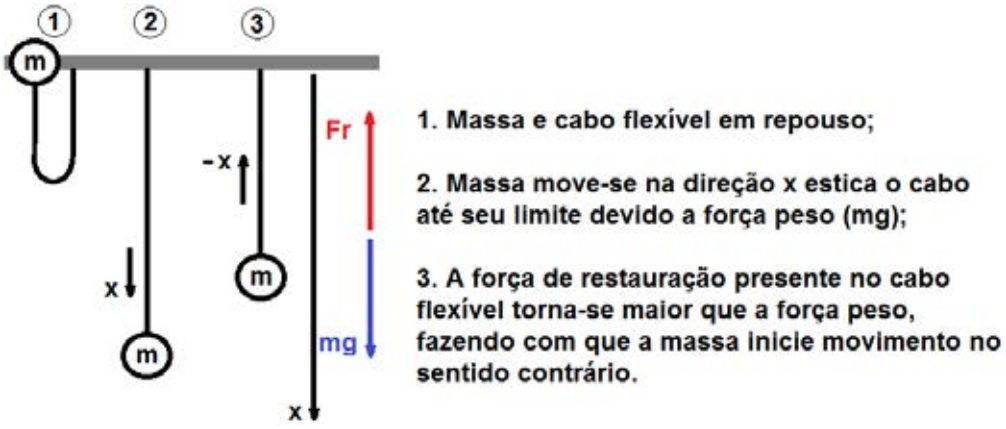

Figura 19 - Esboço do modelo matemático do sistema de uma massa presa por um cabo flexível

O objetivo com o primeiro modelo é testar a modelagem matemática do sistema utilizando o método da diferença central para deduzir as equações de movimento, o sistema por si só é simples e por isso não existe possibilidade de implementar um algoritmo paralelo para o mesmo.

Para modelagem matemática do problema utilizou-se da segunda lei de Newton, e as forças presentes no sistema foram à força de inércia (F_i), a força de

dissipação (F_d), a força restauradora (F_r) do cabo e a força de gravidade aplicada ao sistema (F), como visto na equação (12) e (13).

$$F_i = F - F_r - F_d \tag{12}$$

$$m\ddot{x} = mg - F_r - F_d \tag{13}$$

Pelas aproximações em série de Taylor deduzimos a velocidade (14) e aceleração (15):

$$\dot{X}_t = \frac{X_{t+\delta} - X_{t-\delta}}{2h} \tag{14}$$

$$\ddot{X}_t = \frac{X_{t+\delta} - 2X_t + X_{t-\delta}}{h^2} \tag{15}$$

Substituímos então as aproximações na equação diferencial de movimento do sistema (16).

$$m(\ddot{X}_t) + c(\dot{X}_t) + F_r = mg \tag{16}$$

Após a substituição das aproximações para velocidade (14) e aceleração (15) na equação de movimento (16) e reorganizando a equação, obtemos a formula de recorrência que descreve o comportamento do sistema permitindo avançar em cada passo, visto na equação (17).

$$X_{t+\delta} = \frac{m*g + \left(2m/h^2\right)X_t + \left(c/2h - m/h^2\right)X_{t-\delta} - F_r}{\left(m/h^2 + c/2h\right)} \tag{17}$$

Porém a formula de recorrência não é auto iniciada e necessita da dedução de seus dois passos anteriores para que funcione, facilmente obtidos pelas equações (18) e (19).

$$\ddot{x}_0 = \left(\frac{mg - (c\dot{x}_0) - F_r}{m} \right) \tag{18}$$

$$x = x_0 + (\dot{x}_0 \delta) + \left(\frac{1}{2} \ddot{x}_0 \delta^2 \right) \tag{19}$$

Para uma analise inicial da eficiência do algoritmo foram utilizados valores simbólicos apenas para validar o funcionamento do algoritmo, tais valores são adimensionais e estão descritos abaixo:

E = 1
A = 1
L = 1
m = 1
x0 = x'0 = 0
Num=800.

As variáveis E, A, L são constantes referentes ao cabo (respectivamente, constante do material, sessão transversal e comprimento); x0 e x'0 são constantes referentes a velocidade e aceleração iniciais da massa; Num é o número de passos do sistema.

A resposta para o deslocamento da massa existente no sistema pode ser visualizada na Figura 20.

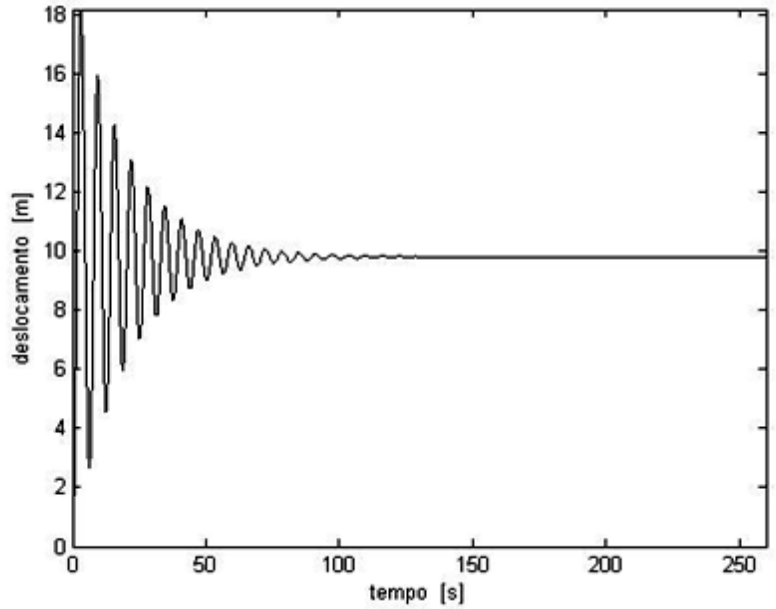

Figura 20 - Deslocamento do modelo constituído de uma massa presa a um cabo flexível (PAIVA et al., 2014 e SIMIONI; BALTHAZAR; BRASIL, 2007)

A partir dai observa-se que a massa inicialmente em repouso inicia seu deslocamento na vertical, porém sua amplitude diminui com o decorrer do tempo, devido à força da gravidade, cessando seu movimento após alguns instantes.

A solução do problema utilizando o método de diferenças finitas é semelhante à solução analítica do mesmo problema, o que valida a técnica utilizada para uso com problemas mais complexos.

O algoritmo desenvolvido para o problema, Tabela 5, permite visualizar de forma simples os cálculos realizados.

Tabela 5 - Algoritmo sequencial do modelo matemático de uma massa presa a um cabo flexível

ALGORITMO SEQUENCIAL
ENTRADA: Gravidade, massa da bola, altura do estado de repouso, comprimento do cabo flexível, propriedades do cabo flexível, velocidade e aceleração inicial da massa, amortecimento, tamanho do passo, número de passos. **SAÍDA:** Um arquivo matriz com o deslocamento da massa em relação ao tempo. **Passo 1:** Adquire e ajusta as variáveis do sistema. **Passo 2:** Escreve o primeiro passo (0x0). **x0=0.0;** **xl0=0.0;** **Passo 3:** Calcula amortecimento. **txAmort=0.05;** **k = (e*A)/(l);** **w = sqrt(k/m);** **c = txAmort * 2.0 * m * w;** **Passo 4:** Calcula tamanho do passo. **passo = (1.0/20.0)*(2.0 * pi /w);** **Passo 5:** Calcula Fr inicial. **Passo 6:** Calcule aceleração inicial. **xll0 = ((forca) - (c * xl0) - fr) / m;** **Passo 7:** Calcule e escreva o segundo passo. **x = x0 + (xl0 * passo) + ((1.0/2.0)* xll0 * (passo * passo));** **Passo 8:** Calcule os coeficientes fixos da equação principal. **xold=x0;** **i=2;** **a0=(2*m)/(passo*passo);** **a1=c/(2*passo)- m/(passo*passo);** **a2=m/(passo*passo) + c/(2*passo);**

Passo 9: Faça até que alcance o número de passos:

Passo 10: Atualize o passo.

Passo 11: Atualize Fr.

Passo 12: Calcule e escreva a nova posição.

Passo 13: Feche o arquivo.

Passo 14: Fim.

O código fonte do programa completo em linguagem de programação C++ pode ser visto no Anexo I.

7 MODELO MATEMÁTICO DE DOIS SATÉLITES AMARRADOS (EQUILIBRADOS)

Com o objetivo de aumentar a complexidade do sistema, um novo modelo foi criado, dessa vez com dois graus de liberdade.

O segundo modelo matemático abordou o problema das duas massas equilibradas (satélites) girando em torno da orbita geossíncrona da Terra, neste sistema o cabo que liga as duas massas também apresenta alterações em seu comprimento, as massas se distanciam uma da outra até que o cabo seja tensionado, o mesmo apresenta variações de comprimento até se estabilizar devido a força centrifuga e a força peso estarem atuando no sistema.

Primeiramente é criado um programa sequencial para validar o modelo do problema, em seguida um novo programa que utiliza paralelismo é criado e o desempenho entre os dois programas é avaliado.

As forças envolvidas nesse problema são: força de restauração (única comunicação existente entre os dois satélites, representada pelo cabo), força centrifuga, que leva os corpos para fora da órbita terrestre, força peso, devida à gravidade, que tende a fazer os corpos cair em direção a Terra.

A Figura 21 apresenta o modelo.

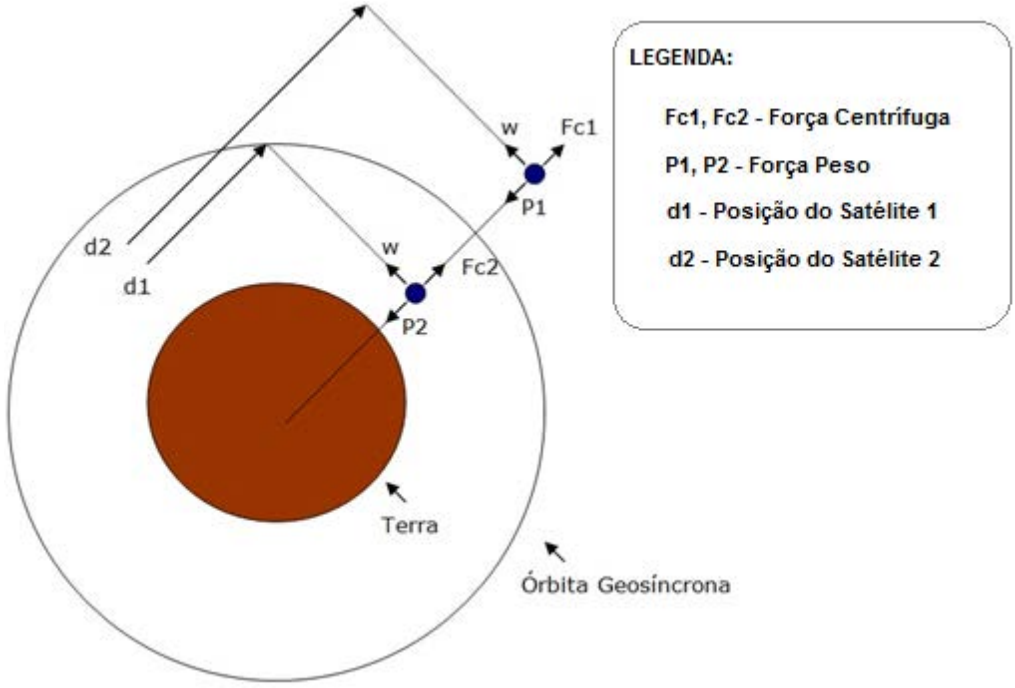

Figura 21 - Modelo de dois satélites equilibrados (PAIVA et al., 2014 e SIMIONI; BALTHAZAR; BRASIL, 2007)

Em um instante inicial, uma das massas está acima da órbita desejada e tenderá para fora devido a força centrifuga ser maior que a força peso, enquanto que a outra massa moverá na direção da Terra, pois a força peso será maior que a força centrifuga, em um dado momento o cabo que interliga os dois estará esticado e então ocorrerá um equilíbrio.

A força peso é definida pelas equações (20) e (21) obtida pela Lei de gravitação universal de Newton.

$$P1 = m1 * g * \left[R/\left(d1 + x1\right)\right]^2 \quad (20)$$

$$P2 = m2 * g * \left[R/\left(d2 + x2\right)\right]^2 \quad (21)$$

Onde:

P1 representa o peso na massa m1;
P2 representa o peso na massa m2;

d1 representa a distância inicial da massa m1 até o centro da Terra;
d2 representa a distância inicial da massa m2 até o centro da Terra;
x1 representa o deslocamento da massa m1;
x2 representa o deslocamento da massa m2;
R representa o raio da Terra.

A força centrífuga é definida pelas equações (22) e (23).

$$Fc1 = m1*(d1+x1)*w^2 \quad (22)$$

$$Fc2 = m2*(d2+x2)*w^2 \quad (23)$$

Onde:

Fc1 representa a força centrífuga na massa m1;
Fc2 representa a força centrífuga na massa m2;
d1 representa a distância inicial da massa m1 até o centro da Terra;
d2 representa a distância inicial da massa m2 até o centro da Terra;
x1 representa o deslocamento da massa m1;
x2 representa o deslocamento da massa m2;
w é a frequência de rotação das massas.

Assumindo que a distancia entre as duas massa é calculada pela equação (24), e também considerando que se essa distancia for menor que o comprimento do cabo L será nula, a força de restauração presente no cabo pode ser calculada pelas equações (25) e (26).

$$dist = (d1+x1) - (d2+x2) \quad (24)$$

$$Fr = k*(dist - L) \quad (25)$$

$$k = \left(\frac{E * A}{L} \right) \tag{26}$$

A rigidez do cabo é apresentada pela equação (26), enquanto que E representa o modulo de elasticidade do material, e A representa a área da seção do cabo.

Assim obtém-se as equações diferenciais do movimento em (27) e (28).

$$m1 * \ddot{x}1 + c1 * \dot{x}1 = Fc1 - P1 - Fr \tag{27}$$

$$m2 * \ddot{x}2 + c2 * \dot{x}2 = Fc2 - P2 + Fr \tag{28}$$

Onde:

c1 é coeficiente de amortecimento da massa m1 por ($c1 = \alpha * m1$);

c2 é coeficiente de amortecimento da massa m2 por ($c2 = \alpha * m2$);

α representa uma constante de amortecimento proporcional às massas.

Após a substituição das aproximações para velocidade e aceleração e reorganizando as equações obtemos as formulas de recorrência (29) e (30) para cada massa que descreve o comportamento do sistema permitindo avançar em cada passo.

$$X1_{t+\delta} = \frac{\left(2m1 / h^2 \right) X1_t + \left(c1 / 2h - m1 / h^2 \right) X1_{t-\delta} \;\; Fc1 - P1 - F_r}{\left(m1 / h^2 + c1 / 2h \right)} \tag{29}$$

$$X2_{t+\delta} = \frac{\left(2m2 / h^2 \right) X2_t + \left(c2 / 2h - m2 / h^2 \right) X2_{t-\delta} \;\; Fc2 - P2 - F_r}{\left(m2 / h^2 + c2 / 2h \right)} \tag{30}$$

Com as equações (29) e (30) pode-se então simular o deslocamento dos satélites observando que o único acoplamento existente entre as duas equações é a força de restauração presente no cabo Fr.

O comportamento dos dois satélites amarrados no sistema saiu como esperado, sendo que uma massa moveu-se em direção a Terra, enquanto a outra era lançada em direção ao espaço, em um determinado ponto no tempo o cabo estica, sofre certa vibração e estabiliza na orbita geossíncrona, a Figura 22 ilustra o ocorrido.

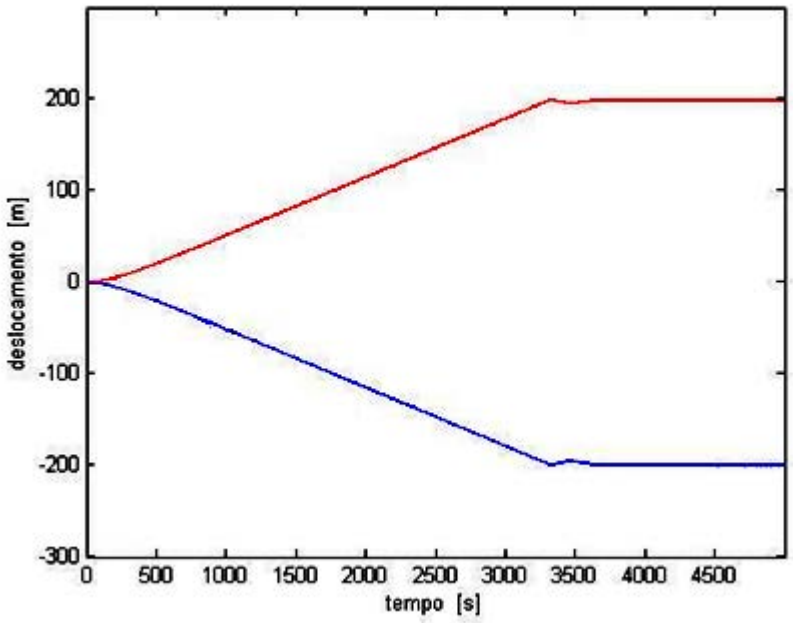

Figura 22 – Representação gráfica do deslocamento no tempo dos dois satélites até o momento de estabilização do sistema (PAIVA et al., 2014 e SIMIONI; BALTHAZAR; BRASIL, 2007)

As linhas obtidas no gráfico são representações dos dois satélites envolvidos no problema, elas se distanciam uma da outra e depois estabilizam em uma certa distancia, ou seja, os satélites movem-se em sentido opostos devido as forças que atuam sobre os mesmos, após o cabo que os interliga ser tensionado, os satélites estabilizam-se em uma distancia fixa.

A partir dai avalia-se o desempenho obtido comparando o tempo de processamento do programa sequencial que levou 15 milissegundos para resolver o problema, com o programa paralelo que levou 38 milissegundos para resolver o mesmo problema dividindo a tarefa entre dois processadores.

Para calcular o speedup procedemos pela equação (31).

$$S(p) = \frac{T_S}{T_P} = \frac{0{,}015}{0{,}038} = 0{,}39 \tag{31}$$

Com o valor de speedup obtidos aplicamos o resultado na equação (32) para obtermos o fator de eficiência.

$$E = \frac{S(p)}{p} * 100\% = \frac{0{,}39}{2}(100\%) = 19{,}5\% \tag{32}$$

Os dados de speedup e frequência foram respectivamente, S(p)=0,39 e E=19,5%, o que representam fatores de eficiência muito baixos para o problema.

Visivelmente é possível observar que a execução do mesmo problema paralelizado obteve praticamente o dobro de tempo para resolver o mesmo problema que o programa serial, isso ocorre devido à necessidade de comunicação entre os processadores envolvidos em resolver a tarefa, apesar de dividirem a trabalho entre eles, um depende do outro para efetuarem os cálculos, momento onde a comunicação torna-se crucial.

O algoritmo sequencial utilizado para o problema dos dois satélites equilibrados pode ser visto na Tabela 6, enquanto que o algoritmo paralelo

utilizado para o problema dos dois satélites equilibrados pode ser visto na Tabela 7, os dois algoritmos são uma representam de forma simplificada os passos que são utilizados para simulação do sistema.

A diferença entre o algoritmo sequencial para o algoritmo paralelo é basicamente que o segundo ao utilizar MPI, permite dividir a carga computacional de cada grau de liberdade do sistema para processadores diferentes, sendo que para que os cálculos de cada satélite seja efetuado, é necessária a comunicação desses dois processos devido à acoplagem dos dois pelo cabo flexível.

A divisão de carga entre processadores não é automática, o programador deve observar o algoritmo e código fonte do programa sequencial e determinar se ele é paralelizável e qual a melhor estratégia a ser tomada.

Após compilar o programa paralelo determinou-se o número de processadores a ser utilizado para simular o problema, o programa executável é enviado para dois processadores, sendo que, cada processador possui sua identificação distinta obtida através da função **MPI_Comm_rank**:

mpiexec –n 2 <nome do programa>.exe

Assim, após obter os dados de entrada do problema, estabeleceu-se o ambiente MPI, com base nas variáveis do sistema os primeiros cálculos que são comuns aos dois satélites são efetuados, através de estruturas condicionais determinou-se o que cada processador deve calcular, sendo que um dos processadores ficou encarregado de calcular também a força de restauração presente no cabo flexível.

A cada passo do sistema um processador envia ao outro sua posição x atualizada, o outro processador fica encarregado de enviar de volta o valor da força de restauração presente no cabo flexível, esse processo se repete até que o número de passos determinados seja alcançado.

Tabela 6 - Algoritmo sequencial do problema de dois satélites equilibrados

ALGORITMO SEQUENCIAL

ENTRADA: Número de passos, tamanho do passo, gravidade na Terra, distância dos corpos em relação à Terra, massa dos corpos, amortecimento relativo às massas, velocidades iniciais, propriedades do cabo flexível.
 	
SAÍDA: Um arquivo matriz com o deslocamento das duas massas em relação ao tempo.
 	
Passo 1: Adquire e ajusta as variáveis do sistema.
 	
Passo 2: Estabelece propriedades dos componentes do sistema.
 	
Passo 3: Escreve o primeiro passo (0x0).
 	
`x1 = x10 + ((x1l0 * passo) + ((1.0/2.0)* x1ll0 * ((passo * passo))));`
`x2 = x20 + ((x2l0 * passo) + ((1.0/2.0)* x2ll0 * ((passo * passo))));`
 	
Passo 4: Calcule G1 e G2.
 	
`g1=g*pow((r/(d1+x1)),2);`
`g2=g*pow((r/(d2+x2)),2);`
 	
Passo 5: Calcule P1 e P2.
 	
`p1=m1*g1;`
`p2=m2*g2;`
 	
Passo 6: Calcule Fc1 e Fc2.
 	
`fc1 = m1 * (d1+x1) * pow(w,2);`
`fc2 = m2 * (d2+x2) * pow(w,2);`
 	
Passo 7: Calcule Fr para o segundo passo.
 	
`Se (fr=((d1+x1)-(d2+x2)<l) faça:`
`  Fr=0;`
`Senão`
`  Fr=(k*(((d1+x1)-(d2+x2))-l));`
 	
Passo 8: Calcule aceleração inicial de ambas as massas.

```
x1ll0 = (fc1 - p1 - fr - c1 * x1l0)/m1;
x2ll0 = (fc2 - p2 + fr - c2 * x2l0)/m2;
```

Passo 9: Calcule e escreva o segundo passo para ambas as massas.

```
x1 = x10 + ((x1l0 * passo) + ((1.0/2.0)* x1ll0 * ((passo
* passo))));
x2 = x20 + ((x2l0 * passo) + ((1.0/2.0)* x2ll0 * ((passo
* passo))));
```

Passo 10: Calcule os coeficientes fixos das duas equações principais.

Passo 11: Faça até que alcance o número de passos:

Passo 12: Atualize o passo.

Passo 13: Atualize Fr, P, Fc e G para ambas as massas.

Passo 14: Calcule e escreva a nova posição para ambas as massas.

Passo 15: Calcule a porcentagem de conclusão.

Passo 16: Feche os arquivos.

Passo17: Fim.

Tabela 7 - Algoritmo paralelo para o problema de dois satélites equilibrados

ALGORITMO PARALELO

ENTRADA: Número de passos, tamanho do passo, gravidade na Terra, distância dos corpos em relação à Terra, massa dos corpos, amortecimento relativo às massas, velocidades iniciais, propriedades do cabo flexível.
 	
SAÍDA: Um arquivo matriz com o deslocamento das duas massas em relação ao tempo.
 	
Passo 1: Estabelece ambiente MPI.
 	
`MPI_Init(&argc,&argv);`
`MPI_Comm_size(MPI_COMM_WORLD,&numprocs);`
`MPI_Comm_rank(MPI_COMM_WORLD,&myid);`

Passo 2: Adquire e ajusta as variáveis do sistema.

```
num=50000;            Passos do sistema. (50000)
passo = 0.001;   Tamanho do passo. (0.001)
passo2=pow(passo,2);  Passo ao quadrado.
i=2;      O laço principal começará a partir do segundo
passo.
g=9.78;     Gravidade.

Do cabo flexível
l = 40400;      Comprimento do cabo flexível.
e = 2*pow(10.0,11);  Constante do material.
A = 1*pow(10.0,-4);  Área da sessão transversal.
k = (e*A)/(l);  Rigidez do cabo flexível.

Dos Satélites
w = 2 * pi/86400;    Frequência da rotação da terra.
r = 40000000/(2*pi); Raio da terra
```

Passo 3: Estabelece propriedades dos componentes do sistema.

```
Calculo da Órbita Geossíncrona
ha=(g*(r*r)/(w*w));
ha2=1.0/3.0;
d0=pow(ha,ha2);
Atribuição da força de restauração inicial.
d1=d0+20000;    Distancia de x1 a terra.
x10=0;          Posição relativa.
d2=d0-20000;    Distancia de x2 a terra. (inclui o cabo
flexível)
x20=0;          Posição relativa.
```

Se processo=0

Passo 4a: Escreve primeiro passo

Passo 5a: Prepara variáveis para calcular segundo passo.

```
m1=1000;   Massa 1.
c1=0.005*m1;    Amortecimento da massa 1.
x1l0=0;    Velocidade inicial de x1.
```

Passo 6a: Calcula e escreve o segundo passo.

```
g1=g*pow((r/(d1+x1)),2);   Constantes
gravitacionais.
p1=m1*g1;             Peso de cada Massa.
fc1 = m1 * (d1+x1) * pow(w,2);
x1ll0 = (fc1 - p1 - fr - c1 * x1l0)/m1;
x1 = x10 + ((x1l0*passo) + ((1.0/2.0)* x1ll0 *
(passo2)));
```

Passo 7a: Calcule os coeficientes fixos da equação principal.

Passo 8a: Faça até que alcance o número de passos:

Passo 9a: Recebe X2 de processo=1.

Passo 10a: Calcule Fr.

Passo 11a: Envie Fr para processo=1.

Passo 12a: Atualize passo, G1, P1 e Fc1.

Passo 13a: Calcule e escreva novo X1.

```
dt=i*passo;
g1=g*pow((r/(d1+x1)),2);
p1=m1*g1;
fc1=m1 * (d1+x1)* pow(w,2);
x1mid=x1;
x1= (fc1 - p1 - fr+c1x1*x1+c2x1*x1old)/c3x1;
```

Passo 14a: Calcule a porcentagem de conclusão.

Passo 15a: Feche o arquivo.

Se processo=1

Passo 4b: Escreve primeiro passo (0x0).

Passo 5b: Prepara variáveis para calcular segundo passo.

```
m2=1000;   Massa 2.
c2=0.005*m2;    Amortecimento da massa 2.
x2l0=0;    Velocidade inicial de x2.
```

Passo 6b: Calcula e escreve o segundo passo.

```
g2=g*pow((r/(d2+x2)),2); Constantes gravitacionais.
p2=m2*g2; Peso de cada Massa.
fc2 = m2 * (d2+x2) * pow(w,2);
x2ll0 = (fc2 - p2 + fr - c2 * x2l0)/m2;
x2 = x20 + ((x2l0 * passo) + ((1.0/2.0)* x2ll0 *
(passo2)));
```

Passo 7b: Calcule os coeficientes fixos da equação principal.

Passo 8b: Faça até que alcance o número de passos:

Passo 9b: Envia X2 de processo=0.

Passo 10b: Atualize passo, G2, P2 e Fc2 (espera para processo=0).

Passo 11b: Recebe Fr de processo=0.

Passo 12b: Calcule e escreva novo X2.

```
dt=i*passo;
g2=g*pow((r/(d2+x2)),2);
p2=m2*g2;
fc2=m2 * (d2+x2)* pow(w,2);
x2mid=x2;
Recebe de Processo 0 fr
x2=(fc2 - p2 + fr + c1x2*x2 +
```

```
      c2x2*x2old)/c3x2;
                x2old=x2mid;
```

Passo 13b: Feche o arquivo.

Passo (16a e 14b): Termine o comunicador.

Passo (17a e 15b): Fim.

8 MODELAGEM MATEMÁTICA DO SISTEMA DE N SATÉLITES CONECTADOS POR CABOS FLEXÍVEIS

O modelo de Simioni, Balthazar e Brasil (2007), esboçado na Figura 23 é composto por várias massas, sendo que uma delas é a própria Terra, todas as massas estão interligadas por cabos fazendo com que uma dependa da outra em seus cálculos.

Para encontrarmos uma solução inicialmente utilizamos coordenadas polares, sendo que estabelecido um patamar, todos os outros nós do sistema dependeram daquele para definir seus ângulos de inclinação.

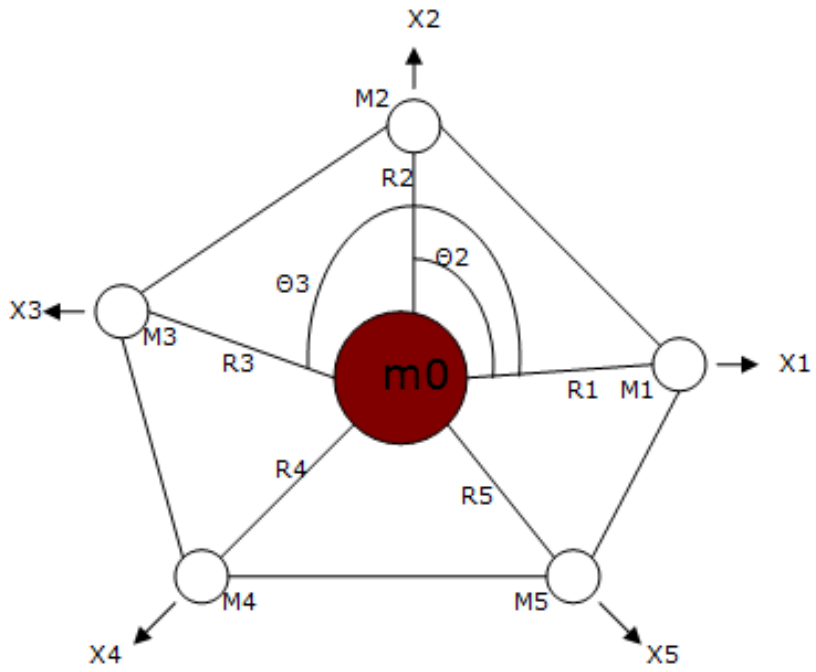

Figura 23 - Modelo matemático de um sistema de satélites amarrados para N graus de liberdade (SIMIONI; BALTHAZAR; BRASIL, 2007)

Segundo Simioni, Balthazar e Brasil (2007), as coordenadas polares foram escolhidas especialmente pela facilidade que proveem, frente às coordenadas geométricas.

Definido o uso de coordenadas polares, retomamos o desenvolvimento de um algoritmo sequencial, para efetivar o desenvolvimento da solução, e então, transforma-la em uma solução com paralelismos.

Onde:

x1 representa a posição da massa m1;

R1 representa o raio da massa m1 para m0 (m0 é fixa, representa a Terra);

θ representa o ângulo da coordenada polar da massa em relação à x1 (patamar).

Para estudar o modelo computacionalmente, a forma que os dados são inseridos no algoritmo vetorialmente, onde os dados são lidos de um arquivo principal de entrada.

Dados os valores iniciais é calculada a rigidez de cada cabo e a distância entre as massas, assumindo que o deslocamento seja somente radial.

Com o raio atualizado de cada massa, encontra-se a nova distancia entre massas utilizando trigonometria simples. Casos particulares são tratados, como o de o nó inicial ser a Terra, ou mesmo se as massas forem alinhadas radialmente.

A cada passo do sistema, as forças de restauração nos cabos são novamente calculadas através da consulta do comprimento atualizado de cada cabo. Caso este tenha passado de seu comprimento inicial, uma nova força de restauração é calculada.

Após o calculo dessa força para cada cabo que chega a uma dada massa, essa é decomposta na direção radial, admitindo-se que a força restauradora resultante esteja nessa direção.

Para o caso particular da massa inicial ser a Terra, a força já está na direção radial. Nos outros casos é calculada, segundo a Figura 24 utilizando trigonometria básica.

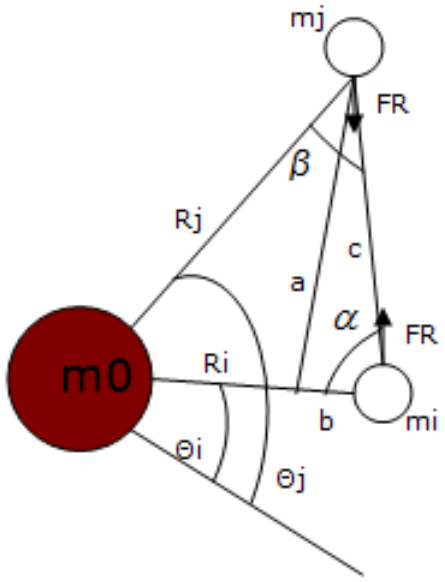

Figura 24 – Determinação das forças de restauração para o sistema de N satélites amarrados (SIMIONI; BALTHAZAR; BRASIL, 2007)

$$a = Rj * sen(\Theta j - \Theta i)$$
$$b = Ri * Rj \cos(\Theta j - \Theta i) \quad (33)$$

$$\cos\alpha = {}^{b}\!/_{Cr}$$
$$\beta = \pi - \alpha - \Theta j + \Theta i \quad (34)$$

E então é realizada a decomposição de Fr, sendo que em *Fmi= FR*cos* α e *Fmj=FR*cosβ*.

Feita a decomposição das forças de restauração, resta apenas recalcular a força centrifuga e o peso para cada massa em cada passo, utilizando os raios atualizados. Para tal, usamos as seguintes expressões:

$$Ri = Ri + Xi * (t + \delta) - Xi * (t)$$
$$FCi = \omega^2 * Mi * Ri$$
$$Pi = Mi * G\left({}^{Ri}\!/_{R0}\right)^2 \quad (35)$$

Onde Ri representa o novo raio da massa; FCi representa a força centrifuga de uma determinada massa; w representa a frequência do sistema (a mesma da Terra); Pi representa o peso de uma determinada massa.

Com base nas equações deduzidas por Simioni, Balthazar e Brasil (2007), pode-se desenvolver o algoritmo do sistema para simula-lo computacionalmente, assim, o algoritmo sequencial foi desenvolvido e implementado, Tabela 8.

Tal algoritmo trouxe o resultado que esperávamos: maior carga computacional, maior tempo de processamento e tempo útil de processamento.

Tabela 8 - Algoritmo sequencial do problema com N satélites

ALGORITMO SEQUENCIAL
ENTRADA: Número de Nós do sistema, Número de Cabos do Sistema, Número de passos a ser executado, Tamanho do passo do sistema, Mapeamento dos Nós, Amortecimento do sistema, Mapeamento dos Cabos. **SAÍDA:** Um arquivo matriz com o deslocamento das N-1 (excluindo a Terra que é fixa) massas em relação ao tempo. /* Adquire e ajusta as variáveis do sistema. */ **Passo 1:** Leia (num. nós, num. cabos, num. passos, tamanho do passo). /* Estabelece propriedades dos componentes do sistema. */ **Passo 2:** Ajuste os vetores para começarem em 0. **Passo 3:** Zera os vetores do sistema. /* Adquire e mapeia as massas do sistema e suas propriedades. */ **Passo 4:** Faça até numero de nós **Passo 5:** leia(massa, raio, angulo, velocidade inicial, aceleração inicial). **Passo 6:** Adquire o amortecimento total do sistema. /* Adquire e mapeia os cabos do sistema e suas propriedades. */ **Passo 7:** Faça até numero de cabos: **Passo 8:** leia (nó inicial, nó final, E*A, comprimento total). **Passo 9:** Escreve as posições iniciais das massas do sistema. /* Calcula a rigidez e as forças restauradoras iniciais.*/

ea[n]=ea[n]/dl[n];

Passo 10: Faça até número de cabos
/* Recalcula componente dos cabos para cálculo da Fr */

Passo 11: E*A[n]=E*A[n]/comprimento[n].

Passo 12: Se nó está ligado a Terra

comprimento = raio do nó - raio da Terra.
ajuste alfa e beta de C.

Passo 12: Se não

Calcule as componentes de Pitágoras.
Calcule Pitágoras.
Ajuste alfa e beta de C.

Passo 13: Se houve esticamento do cabo flexível através de C

Frn=0.

Se não

Frn=ak*(c-dli).

Passo 14: Atualize Fr(nó inicial).

Passo 15: Atualize Fr(nó final).
/* Prepara a inicialização do laço principal */

Passo 16: Faça até número de nós

Passo 17: Atualize Fc, P, Fam, Aceler.

Passo 18: Calcule e escreva o segundo passo.

Passo 19: Calcule as constantes para as eqs. principais.
/* Laço principal para integração */

Passo 20: Faça até numero de passos

Passo 21: Limpe vetor de forças restauradoras.

Passo 22: Faça até numero de cabos

Passo 23: Se há alguma das pontas sendo a Terra.

Atualize C, alfa e beta.

Passo 23: Se não

Calcule componentes de Pitágoras.

Calcule Pitágoras.

Atualize alfa e beta.

Passo 24: Se houve esticamento

Frn=0

Passo 24: Se não

Frn=ak(comprimento atual - comprimento inicial do cabo flexível)

Passo 25: Atualize Fr (nó inicial).

Passo 26: Atualize Fr (nó final).

Passo 27: Faça até número de nós

Passo 28: Atualize Fcen, P.

Passo 29: Calcule e escreva novas posições das massas.

Passo 30: Calcule a porcentagem de conclusão.

Passo 40: Feche arquivo.

Em seguida temos o algoritmo paralelizado para o problema com N satélites na Tabela 9.

Tabela 9 - Algoritmo paralelo para o problema com N satélites

ALGORITMO PARALELO
ENTRADA: Número de Nós do sistema, Número de Cabos do Sistema, Número de passos a ser executado, Tamanho do passo do sistema, Mapeamento dos Nós, Amortecimento do sistema, Mapeamento dos Cabos. **SAÍDA:** Um arquivo matriz com o deslocamento das N-1 (excluindo a Terra que é fixa) massas em relação ao tempo. **Passo 1:** Estabelece ambiente MPI. **MPI_Init(&argc,&argv);**

MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);

Passo 2: Adquire e ajusta as variáveis do sistema.

pi=acos(-1.0);
r0=20000000/pi;
g0=9.78;
omega=2.0*pi/86400;

Passo 3: Estabelece propriedades dos componentes do sistema.

Se processo==0

Passo 4a: Define o nome do arquivo N para o nó N.

Passo 5a: Coleta dados de entrada a partir de entrada.txt e ajusta índices para iniciarem em 0.

Passo 6a: Envia dados do arquivo de entrada para todo o comunicador.

Passo 7a: Escreve primeiro passo (0x0).

Passo 8a: Calcula força restauradora inicial e envia para o comunicador.

Passo 9a: Calcula e escreve o segundo passo.

Passo 10a: Inicializa variáveis do laço principal.
/* Laço principal de integração. */

Passo 11a: Faça até que alcance o número de passos:

Passo 12a: Sincroniza a cada 20 passos.

Passo 13a: Obtém os raios de todas as massas do comunicador.

a=r[nfi]*sin(teta[nfi]-teta[nin]);
b=r[nin]-r[nfi]*cos(teta[nfi]-teta[nin]);
c=sqrt(a*a+b*b);
calfa=b/c;
alfa=acos(calfa);
cbeta=cos(pi-alfa-teta[nfi]+teta[nin]);

Passo 14a: Calcula vetor Fr para todas as massas.

Se (c < dli) faça

frn=0;

senão

frn=ak*(c-dli);

Passo 15a: Envia o vetor Fr para todos os nós do comunicador.

Passo 16a: Atualize passo, força centrífuga e peso.

Passo 17a: Calcule e escreva a nova posição.

Passo 18a: Feche o arquivo.

Passo 19a: Termina comunicador.

Se processo!=1

Passo 4b: Define o nome do arquivo N para o nó N.

Passo 5b: Coleta pacote de dados de entrada a partir do nó 0.

Passo 6b: Escreve primeiro passo (0x0).

Passo 7b: Recebe Fr atualizado do nó 0.

Passo 8b: Calcula e escreve segundo passo.

Passo 9b: Inicializa variáveis do laço principal.

/* Laço principal de integração. */

Passo 10b: Faça até que alcance o número de passos:

Passo 11b: Sincroniza a cada 20 passos.

Passo 12b: Envia o raios para o nó 0.

Passo 13b: Atualize passo, força centrífuga e peso. (Enquanto o nó 0 está calculando Fr.)

Passo 14b: Recebe Fr atualizado do nó 0.

Passo 15b: Calcule e escreva a nova posição.

Passo 16b: Feche o arquivo.

Passo 17b: Termina comunicador.

No algoritmo paralelo para o problema com N satélites amarrados por cabos flexíveis houve uma abordagem diferenciada quanto a obtenção dos parâmetros necessários para simular o sistema.

Por se tratar de um programa genérico optou-se por utilizar um arquivo do tipo texto para definir as configurações do sistema, essa estratégia de obtenção de dados é vantajosa, pois permite a alteração das características do problema sem que seja necessário compilar novamente os programas. A estrutura do arquivo de entrada de dados segue o disposto na Tabela 10.

Tabela 10 - Configuração do arquivo de entrada de dados

Linha					
01	Número de nós existentes				
02	Número de cabos flexíveis				
03	Número de passo a ser executado				
04	Tamanho do passo				
05	M(N) Massa	R(N) Raio	Teta (N) Ângulo	Xa(N) V. Inicial	Xp(N) A. Inicial
...	Para N nós (Terra + N Satélites)				
07	Coeficiente de amortecimento				
08	Ni(N) Nó Inicial	Nf(N) Nó Final	E*A(N)	L(N) Comprimento do Cabo	
...	Para N cabos flexíveis				

Já para o arquivo de saída com o deslocamento de N satélites determinou-se que no programa sequencial os dados de todos os satélites seriam impressos em um único arquivo, porém no programa paralelo isso não seria possível em virtude de estarmos trabalhando com vários processadores que estariam escrevendo em um único arquivo ao mesmo tempo corrompendo assim as informações contidas ali.

A estratégia utilizada foi a criação de um arquivo de saída com os deslocamentos para cada satélite do sistema, assim, dependendo do número de satélites amarrados temos N arquivos de deslocamento.

O algoritmo sequencial e paralelo inicia sua execução de forma semelhante, efetuando o mapeamento dos nós e cabos do sistema da mesma

forma, lendo o arquivo de configuração descrito na Tabela 10 e posicionando as informações em variáveis e vetores, os dois primeiros parâmetros que determinam o número de nós e de cabos presentes no sistema determinam a forma que uma estrutura de repetição ira percorrer o arquivo em busca das informações.

O terceiro e quarto parâmetro do arquivo de entrada determina o número de passos que o programa ira executar e o tamanho de cada passo do sistema, sendo que esse tamanho é fixo.

As diferenças entre a execução do programa sequencial com o paralelo iniciam-se quando são definidos os valores das constantes do sistema.

No algoritmo sequencial são definidas as constantes do sistema, o arquivo de entrada é aberto para leitura e seus dados estão prontos para serem mapeados, enquanto que o arquivo de saída é criado para escrita estando pronto para receber as informações de deslocamento de cada satélite.

Com os dados mapeados e as constantes calculadas o programa sequencial passa e efetuar o calculo da força de restauração inicial e o calculo da posição inicial de cada nó do sistema, com exceção para o nó que representa a Terra.

Em seguida o programa passa para o laço de repetição principal do sistema para calcular cada nós existente um a um, são obtidos dados do passo anterior e do passo atual para o calculo do próximo passo.

Ao final da execução de cada laço principal o programa escreve no arquivo de saída as posições atualizadas de cada um dos satélites presentes no problema, pulando em seguida para próxima linha do arquivo aguardando nova entrada de informações.

Quando chegar ao final do número de passos determinado o programa sai do laço principal, fecha o arquivo de saída e informa o tempo que foi gasto durante todo processo de calculo.

No algoritmo paralelo após o mapeamento dos vetores e alocação de memórias para as variáveis inicia-se as instruções de paralelização MPI com as funções MPI_Init, MPI_Comm_size e MPI_Comm_rank.

Nesse ponto o programa paralelo é distribuído em vários processos para vários processadores, através de estrutura condicional é determinado que o um dos nós, com id 0, fica responsável por abrir o arquivo de entrada com as configurações do sistema.

Em virtude de a comunicação ser um ponto crucial em programas paralelos, determinou-se que alguns dados seriam empacotados em vetores para que a transferência fosse única para os processos.

A função MPI_Barrier é utilizada possibilita sincronizar os processos que são executados, garantindo assim que nenhum deles tente executar alguma tarefa sem ter recebido informações necessárias para seus cálculos.

Após a leitura do arquivo de configuração e alocação das informações pertinentes dos nós e cabos em vetores os mesmos são distribuídos através da função MPI_Scatter para que os demais processos, essas informações são pertinentes para calcular o deslocamento atual e o anterior de cada nó e os cabos que o interligam.

Utilizando a função MPI_Barrier os processos que terminam seus cálculos primeiro permanecem aguardando todos os outros processos do contexto terminarem suas tarefas, isso garante a troca sincronizada de informações entre os vários processos, utilizando MPI_Bcast as informações atualizadas são distribuídas novamente para que o programa passe para o laço principal de integração do sistema.

No laço principal de integração inicialmente utiliza-se uma estrutura condicional para verificar o número de passos que o programa já efetuou e caso seja superior a 20 executa a função MPI_Barrier para garantir a sincronização do sistema.

Para cada passo de integração os nós dos sistemas e a força de restauração são calculados nos processos, MPI_Bcast garante que todos os nós recebam o valor da força de restauração Fr para recalcularem suas posições x(n).

Cada processo escreve em um arquivo de saída os deslocamentos para o nó que esta a efetuar os cálculos, escrever em arquivos independentes foi a estratégia escolhida para evitar atrasos nos cálculos devido a uma fila para gravação dos dados de saída, bem como perda de dados caso os processos gravassem as informações de deslocamento ao mesmo tempo.

Ao encerrar a integração de todos os passos o programa efetua o calculo do tempo que foi gasto durante a simulação do problema para N satélites amarrados com o programa paralelo, bem como o ambiente MPI é encerrado.

8.1 Simulação utilizando 3 satélites amarrados

Paiva et al. (2014) apresenta uma simulação utilizando quatro nós, sendo um deles fixo e utilizado como patamar, a Terra, seis cabos flexíveis ligando satélites adjacentes (massas m1, m2, m3) com a Terra foi realizado, como visto na Figura 25.

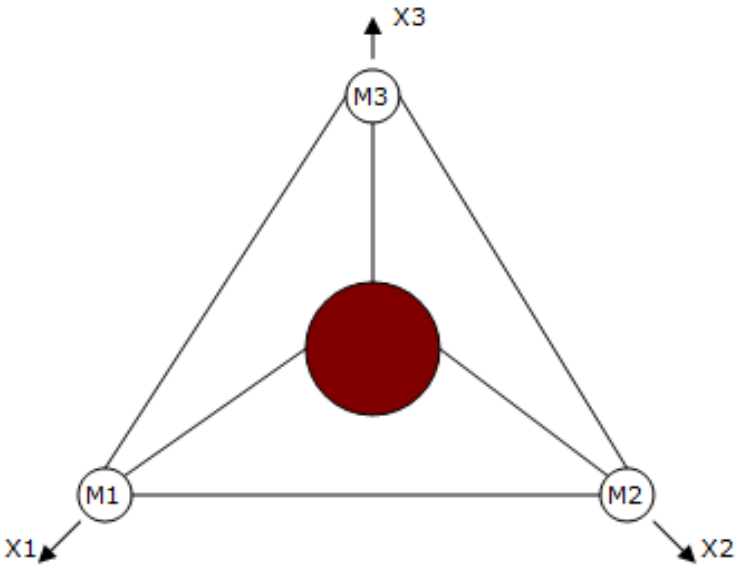

Figura 25 - Modelo matemático do sistema de satélites amarrados com 4 nós (PAIVA et al., 2014 e SIMIONI; BALTHAZAR; BRASIL, 2007)

Para o programa deve-se as seguintes considerações:

Os satélites, massas m1, m2 e m3 possuem 1000kg, e todas as massas estão a uma distância de 42366200m da terra, porém, diferindo nos ângulos de projeção, em 0, 2.09439510239, e 4.18879020478.

Já para os cabos, os que ligam os nós adjacente são de 72000000m de comprimento, enquanto os que ligam os nós à terra, 36000000m de comprimento.

O hardware utilizado nas simulações foi um notebook Core i5-2450M com dois núcleos que simulam quatro núcleos, com velocidade de 2.50GHz cada, com memória instalada de 4 GB, com Microsoft MPI instalado.

O gráfico gerado pelos dados obtidos na simulação de Paiva et al. (2014) e Simioni, Balthazar e Brasil (2007) pode ser visto na Figura 26.

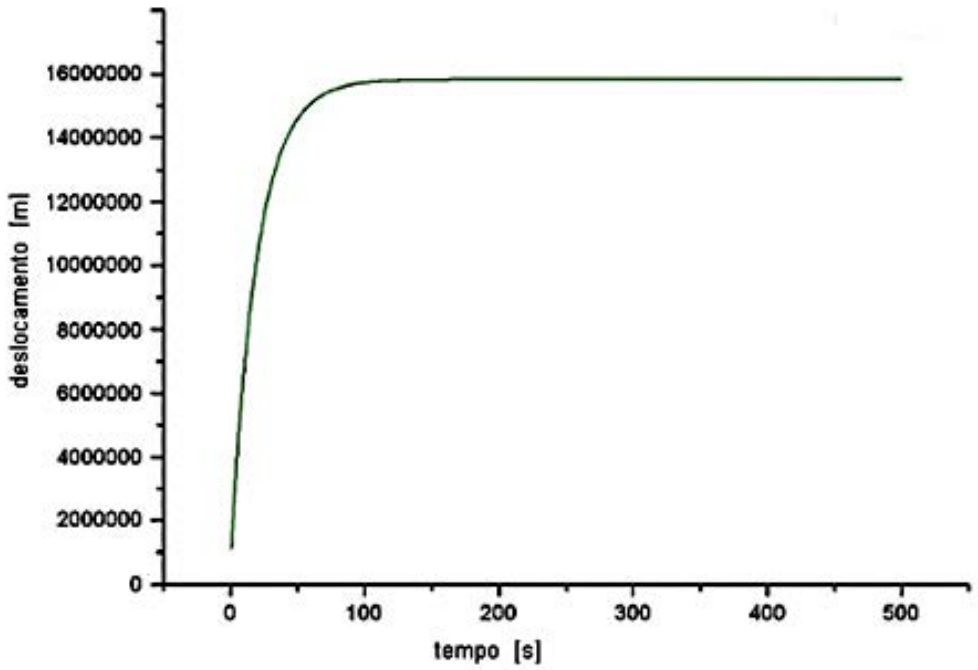

Figura 26 – Representação do deslocamento dos três satélites até equilíbrio em órbita (PAIVA et al., 2014 e SIMIONI; BALTHAZAR; BRASIL, 2007)

O gráfico apresenta uma única curva que representa o deslocamento dos três satélites do sistema, deve-se a isso o deslocamento igual das massas m1, m2 e m3.

Uma nova avaliação de desempenho é realizada comparando o tempo de processamento do programa serial que levou 8 segundos, para resolver o problema, com o programa paralelo que levou 7 segundos para resolver o mesmo problema dividindo a tarefa entre 4 processadores.

Para calcular o speedup procedemos pela equação (36).

$$S(p) = \frac{T_S}{T_P} = \frac{8}{7} = 1{,}142857 \tag{36}$$

Com o valor de speedup obtidos aplicamos o resultado na equação (37) para obtermos o fator de eficiência.

$$E = \frac{S(p)}{p} * 100\% = \frac{1{,}142857}{4}(100\%) = 28{,}57\% \quad (37)$$

Os dados de speedup e frequência foram respectivamente, S(p)=1,14 e E=28,57%, o que representa um aumento na eficiência do uso dos processadores nas simulações, o aumento na carga computacional de certa forma compensa o tempo de sincronização necessário (comunicação para atualização das forças de restauração) e a eficiência aumenta.

8.2 Simulação utilizando 8 satélites amarrados

Como próximo passo uma nova simulação utilizando oito satélites é realizada, sendo que o número de nós calculados pelos programas serão nove, em virtude de um deles representar a própria Terra, como visto na Figura 27.

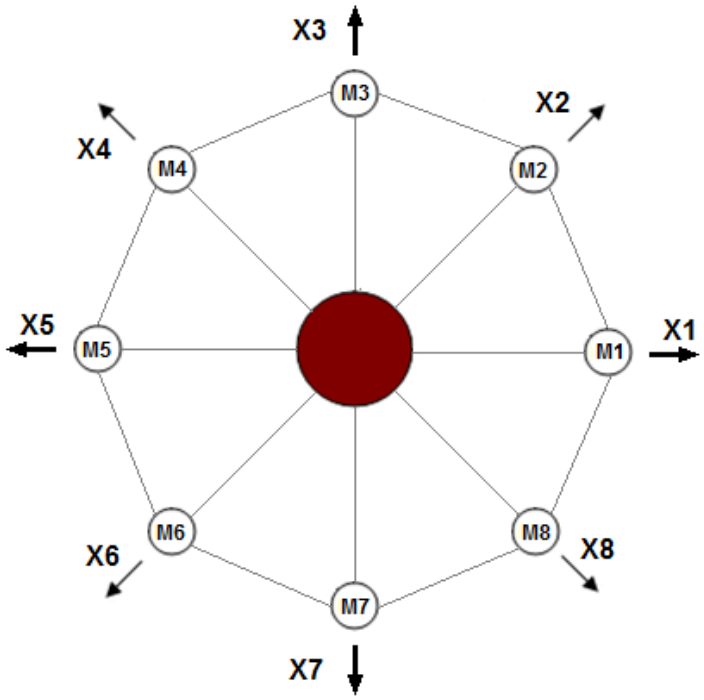

Figura 27 - Modelo do problema com 8 satélites

O arquivo de configuração utilizado nos programas sequenciais e paralelos para este problema possuem as seguintes configurações:

Os satélites, massas m1, m2, m3, m4, m5, m6, m7, m8 possuem 1000kg, e todas as massas estão a uma distância de 42366200 m da terra, porém, diferindo nos ângulos de projeção que são respectivamente: 0; 0,78539816; 1,57079633; 2,35619449; 3,14159265; 3,92699082; 4,71238898 e 5,49778714.

Já para os cabos, os que ligam os nós adjacente são de 72000000m de comprimento, enquanto os que ligam os nós à terra, 96000000m de comprimento. A resistência do cabo também é proporcional.

O hardware utilizado nas simulações foi um notebook Core i5-2450M com dois núcleos que simulam quatro núcleos, com velocidade de 2.50GHz cada, com memória instalada de 4 GB, com Microsoft MPI instalado.

Com a redução de passo a execução do programa sequencial levou 5 segundos, enquanto que a execução do programa paralelo levou 4 segundos assemelhando-se aos resultados obtidos com a simulação com 3 satélites.

Para calcular o speedup procedemos pela equação (38).

$$S(p) = \frac{T_S}{T_P} = \frac{5}{4} = 1{,}25 \tag{38}$$

Com o valor de speedup obtidos aplicamos o resultado na equação (39) para obtermos o fator de eficiência.

$$E = \frac{S(p)}{p} * 100\% = \frac{1{,}25}{4}(100\%) = 31{,}25\% \tag{39}$$

Os dados de speedup e frequência foram respectivamente, S(p)=1,25 e E=31,25%, o que representa um aumento na eficiência do uso dos processadores nas simulações, espera-se que com supercomputadores as simulações sejam capazes de chegar a um índice de eficiência maior, se considerar que o problema com 8 satélites amarrados por cabos flexíveis foi executado em um hardware limitado a 4 processadores, e que por isso a carga de processamento que deveria ser executada em 8 processadores teve que ser distribuída aumentando assim o tempo de execução do programa paralelo.

9 CONCLUSÕES

Este trabalho apresenta um estudo de três modelos matemáticos simplificados utilizando o método explicito das diferenças finitas e computação sequencial e paralela.

Observou-se que no primeiro problema, o modelo matemático de uma massa presa a um cabo flexível, que apresenta um grau de liberdade, neste utilizou-se do método das diferenças finitas para obter a solução do sistema em um determinado espaço discreto de tempo, as técnicas de modelagem do problema foram tradicionais utilizando a Segunda Lei de Newton, porém com os termos da primeira e segunda derivada, a velocidade a aceleração do sistema, substituído por aproximações obtidas de expansões em séries de Taylor, isso permitiu chegar à fórmula de recorrência, necessária para simular o sistema computacionalmente.

Após equacionar o problema e efetuar a simulação com um programa sequencial em linguagem C++ foi obtido sucesso nos resultados esperados e o sistema se comportou semelhante a um sistema real de uma massa presa a um elástico flexível.

No segundo problema, a complexidade do primeiro sistema de uma massa presa a um cabo flexível foi expandida, sendo utilizada a representação de dois satélites conectados por um cabo flexível, movendo na órbita geossíncrona da Terra enquanto giram em torno dela. Os dois satélites partem do mesmo ponto inicial, distanciam-se devido à ação da força peso e a força centrífuga movendo-se opostas uma a outra até o cabo ser tensionado, então os satélites se estabilizam a 180º um do outro.

Para esse problema foram utilizados dois programas, um sequencial e um paralelo, após obter as soluções com o programa sequencial foi estudado a melhor abordagem para paralelizar o sistema e ficou decidido que a simulação procederia com dois nós, sendo que um ficaria responsável por recalcular a força

elástica contida no cabo flexível a cada passo, utilizando MPI a troca de mensagens entre os nós seria única e exclusivamente as novas condições do cabo flexível, e os demais cálculos dependeriam dessa informação.

A simulação do modelo matemático de dois satélites amarrados por um cabo flexível respondeu como o esperado, em virtude do Protocolo de Troca de Mensagens, MPI, utilizar a comunicação como fator primordial para decomposição de um problema, foi previsto um atraso que seria gerado pela comunicação entre os nós nos programas paralelos.

O modelo permitiu observar um fator importante quanto à paralelização de um problema, ficou visível que o problema do modelo matemático de duas massas equilibradas é mais eficiente quando resolvido de forma sequencial, pois o processamento ainda é pequeno diante da troca de informações necessária.

Ao avaliar os resultados das simulações observou-se que o programa sequencial obteve a solução do problema em um tempo menor que o programa paralelo, os cálculos de speedup e eficiência ficaram respectivamente em S(p)=0,39 e E=19,5% o que reforçou a ideia do modelo matemático de dois satélites equilibrados ser melhor resolvido com um programa sequencial.

A pesquisa expande-se para um terceiro modelo matemático com N satélites amarrados por cabos flexíveis, neste modelo os satélites se interligam por cabos flexíveis além de cada um possuir uma ligação com a Terra, em virtude de coordenadas cartesianas serem complexas para um sistema desse tipo, optou-se pela utilização de coordenadas polares para definir o ângulo de projeção de cada satélite perante a Terra.

O sistema assemelha-se ao modelo de dois satélites equilibrados conectados por um cabo flexível, onde existe a necessidade de recalcular as forças de restauração de cada cabo flexível para calcular a nova posição de cada satélite.

Porém, nesse modelo matemático genérico de N satélites amarrados existe a necessidade de recalcular a nova posição de cada satélite utilizando trigonometria básica em virtude dos vários satélites existentes e a disposição dos mesmos.

Neste problema foram efetuadas duas simulações para avaliar o funcionamento dos programas e também a eficiência que pode ser obtida utilizando computação paralela.

Na primeira simulação decidiu-se utilizar quatro nós, sendo um deles a própria Terra e os outros três satélites amarrados entre si e a ela totalizando a configuração um total de seis cabos flexíveis, o tempo de execução gasto pelo programa sequencial foi de 8 segundos para chegar a solução, sendo que o programa paralelo levou 7 segundos para chegar a mesma solução dividindo a carga dos cálculos do problema entre 4 processadores.

Deste modo, alcançou-se um resultado positivo em medidas de tempo para encontrar a solução do problema, restando desta forma avaliar a eficiência da paralelização. Os dados de speedup e eficiência são respectivamente, S(p)=1,142857 e E=28,57%, como esperado ouve um aumento na eficiência da paralelização do programa ao aumentar a complexidade dos cálculos efetuados por cada processador para alcançar a solução.

Em virtude dos resultados obtidos confirmando que o aumento do número de nós calculados no sistema aumentaria a eficiência do programa paralelo uma segunda simulação foi realizada, nesta decidiu-se utilizar nove nós, sendo um deles a própria Terra e os outros oito nós satélites amarrados por cabos flexíveis entre si e com a própria Terra.

Inicialmente as simulações transcorriam bem até determinado ponto e depois paravam de executar ocasionando um travamento no sistema, o mesmo algoritmo e programa da primeira simulação foi utilizado, assim sendo o problema não poderia ser no programa e sim em suas configurações no arquivo de entrada de dados ou em suas limitações de hardware, após descartar problemas no

arquivo de entrada de dados constatou-se que as limitações impostas pelo hardware estavam causando o travamento do sistema.

Assim, para esse problema, devido às limitações de hardware existentes, optou-se por reduzir o número de passos do mesmo, utilizando um computador limitado a apenas quatro núcleos, estes teriam que compartilhar os cálculos dos nós restantes, aumentando o tempo para encontrar a solução do problema.

Reduzidos os passos nos programa sequencial e paralelo os dados obtidos em ambos foram os mesmos, confirmando assim que os programas eram equivalentes, o programa sequencial levou 5 segundos para alcançar a solução do problema e o paralelo levou 4 segundos, assemelhando-se bastante aos resultados obtidos com a simulação envolvendo 4 nós, os dados de speedup e eficiência foram respectivamente, S(p)=1,25 e E=31,25%, o que representou um aumento na eficiência do uso dos processadores no programa paralelo, como era previsto.

10 TRABALHOS FUTUROS

Ficou visível que a cada aumento no número de satélites nas simulações obtinha-se um valor maior de eficiência, assim sendo, para novas analises é necessário utilizar um número maior de satélites com o objetivo de aumentar a carga computacional, tornando assim o atraso criado pela comunicação aceitável em relação ao alto processamento de informações necessário.

Como trabalhos futuros pretende-se aumentar o número de nós do modelo matemático de N satélites amarrados por cabos flexíveis e simular essas novas condições em supercomputadores do tipo grid, como o GridUNESP, sistemas capazes de lidar com um grande número de cálculos computacionais.

GridUNESP possibilitará a simulação de um grande número de nós, porém sua estrutura computacional é diferenciada em relação a um computador normal, os códigos fontes do programas provavelmente terão que ser modificados para trabalhar em sua estrutura, porém é previsto que este tempo de readaptação seja recompensador em virtude da alta capacidade computacional que o sistema GridUNESP pode fornecer.

11 REFERÊNCIAS BIBLIOGRÁFICAS

ALPATOV, A. P., BELETSKY, V. V., DRANOVSKII, V. I., KHOROSHILOV, V. S., PIROZHENKO, A. V., TROGER, H., ZAKRZHEVSKII, A. E. **Dynamics of Tethered Space Systems**. CRC Press - Taylor & Francis Group, Boca Raton, 2010.

BALTHAZAR, J. M., BRASIL, R. M. R. L. F., FENILI, A., MEZA, M. E. M. **Modelagem e Simulação da Dinâmica e Controle de Clusters de Satélites Atirantados com Possível Inclusão de um "Robot" em Ambiente Espacial**. 2º Workshop Dinâmica de Satélites Artificiais. ICT-UNIFESP, São José dos Campos, 2014.

BELETSKY, V. V., LEVIN, E. M. **Dynamics of Space Tether Systems**. American Astronautical Society. San Diego, California, 1993.

BRASIL, R. M. L. R. F., FALLARA, V. **Numerical Assessment of the Nonlinear Dynamics of Clusters of Tethered Satellites**. CILAMCE, Proceedings of the XXXIV Iberian Latin-American Congress on Computational Methods in Engineering Z.J.G.N Del Prado, ABMEC, Pirenópolis, GO, Brasil, November 10-13, 2013.

BRASIL, R.M.L.R.F. **Programas de Microcomputadores para Análise Dinâmica de Estruturas: um grau de liberdade**. Boletim Técnico do Departamento de Engenharia de Estruturas e Fundações, Escola Politécnica da USP, São Paulo, 1992.

BURDER, R. FAIRES, J. D. **Numerical Analysis**. Cengage Learning, 7ª ed., 2001.

CARROLL, J. A. **Guidebook for Analysis of Tether Applications**. Martin Marietta Corporation and NASA Marshall Space Flight Center. 1985.

CENAPAD – SP. **Introdução ao MPI, Centro Nacional de Processamento de Alto Desempenho**. Campinas, São Paulo. http://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_MPI.pdf. Acesso em: 24/07/2013.

COSMO, M. L., LORENZINI, E. C. **Tethers In Space Handbook**. 3ª ed. NASA Marshall Space Flight Center. 1997.

ELLIS, J. R. **Modeling, Dynamics, and Control of Tethered Satellite Systems**. Dissertation. Virginia Polytechnic Institute and State University, Blacksburg. 2010.

GRAMA, A., GUPTA, A., KARYPIS, G., KUMAR, V. **Introduction to Parallel Computing**. Pearson Education Limited, 2ª ed., 2003.

ISHIGE, Y., KAWAMOTO, S., KIBE, S. **Study on Electrodynamic Tether System For Space Debris Removal**. Acta Astronautica 55, pag. 917-929, Science Direct, 2004.

KARNIADAKIS, G., KIRBY, R. M. **Parallel Scientific Computing In C++ And MPI: A Seamless Approach to Parallel Algorithms and Their Implementation**. Cambridge University Press, 2ª ed., 2005.

PACHECO P. S. **An Introduction to Parallel Programming**. Elsevier, 1ª ed., 2011.

PAIVA, L. J. S., BALTHAZAR, J. M., SILVEIRA, M., PONTES JUNIOR, B. R., BRASIL, R. M. L. R. F., SIMIONI, B. . **Comportamento Dinâmico Não Linear de Satélites Amarrados por Cabos Flexíveis**. CONEM – Congresso Nacional de Engenharia Mecânica, Uberlândia, 2014.

PELT, M. V. **Space Tethers and Space Elevators**. Copernicus Books, Springer Science+Business Media. New York, 2009.

RAO, S. S. **Vibrações Mecânicas**. 4.ª ed. Pearson Prentice Hall, São Paulo. 2008.

SAVITCH, J. W. **C++ Absoluto**. Addison Wesley, 1ª ed., 2004.

SIMIONI, B., BALTHAZAR, J. M., BRASIL, R. M. L. R. F. **Rede de Satélites Geossíncronos: Uma Abordagem à Computação Paralela**. Dincon, 6º Brazilian Conference on Dynamics, Control and Their Applications. São José do Rio Preto, SP. May 21-25, 2007.

TROVO, D. G. E., BALTHAZAR, J. M. **Métodos Sequencial e Paralelo das Diferenças Finitas Aplicados à Ciência da Engenharia**. 4º Congresso Temático de Dinâmica, Controle e Aplicações. DINCON, Bauru, 2005.

WILKINSON, B., ALLEN, M. **Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers**. Pearson Education, New Jersey, 2ª ed., 2005.

YAMAIGIWA, Y., HIRAGI, E., KISHIMOTO, T. **Dynamic Behavior of Electrodynamic Tether Deorbit System On Elliptical Orbit and Its Control By**

Lorentz Force. Aerospace Science and Technology 9, pag. 366-373, Science Direct, 2005.

12 ANEXO I

Tabela 11 - Código Fonte: Modelo de uma massa presa a um cabo flexível

Código

```
// Massa Presa ao Elastico Serial.cpp : Defines the entry point for the
console application.
//

#include "stdafx.h"
#include <stdio.h>
#include <math.h>

/* Valores de entrada e condições iniciais.*/
double x0,xl0,xll0,g,m,a,l,e,A,forca,txAmort;

/* Valores para o cálculo do amortecimento. */
double k,w,c;

/* Usadas para o cálculo do Xt+At. */
double x, xold, xmid, fr, a0,a1,a2;

/* Usada como variável do tempo. */
double i,passo,dt;

/* Usado para o cálculo da frequência.*/
double pi = 3.14159265358979323846264 3;

/* Arquivo com resultado para plotar o gráfico. */
FILE * res;

int _tmain(int argc, _TCHAR* argv[])
{
	/* Abrindo o arquivo de resultados. */
	res = fopen("dados.txt","w");

	/* --> Definindo Constantes. */
	g = 9.8;	// Constante gravitacional.
	m = 1;		// Massa da bola.
	a = 1;		// Altura do estado de repouso.
	l = 1;		// Comprimento máximo sem esticar.
	e = 1;		// Constante do material.
	A = 1;		// Área da sessão transversal.
	forca = m*g;

	/* --> Entrada dos valores iniciais. */
	x0=0.0;			// Valor de x0 inserido para teste.
	xl0=0.0;	// Valor de xl0 inserido para teste.

	// Escreve o Xt-At. (x0). Neste caso, dt*0=0.
	fprintf(res,"0.000000,%lf\n",x0);

	/* --> Cálculo do amortecimento. */
	txAmort=0.05;			// Taxa de amortecimento
	k = (e*A)/(l);			// Rigidez.
	w = sqrt(k/m);			// Frequência.
	c = txAmort * 2.0 * m * w; // Amortecimento.

	/* --> Cálculo da constante do passo da integração */
	passo = 0.001; //(1.0/20.0)*(2.0 * pi /w);
	// Será multiplicado por i.
```

```
/* Cálculo da força restauradora inicial, usada no cálculo da aceleração
inicial. */
/* Para este caso, sempre será 0. */
              if (x + a < l)
                     fr=0;
              else
                     fr=(k*(x+a-l));

       /* --> Cálculo da aceleração inicial */
       xll0 = ((forca) - (c * xl0) - fr) / m;
       // Usando a força restauradora inicial.

       /* --> Cálculo do Xt para o Xt+At (x1)*/
       x = x0 + (xl0 * passo) + ((1.0/2.0)* xll0 * (passo * passo));

       // passo*1=passo. (Passo de integração).
       fprintf(res,"%lf,%lf\n",passo,x);

       /* --> Cálculo do Xt+AT. */
       xold=x0;      // xold é usado no cálculo.
       i=2;          // Passo 0 e 1 já calculados.

       a0=(2*m)/(passo*passo);           // Argumento do X atual.
       a1=c/(2*passo)- m/(passo*passo);  // Argumento do X anterior.
       a2=m/(passo*passo) + c/(2*passo); // Argumento da divisão.

       do
       {
              dt=i*passo;   // Cálculo do tempo do passo de integração.

              /* Cálculo do novo Fr. */
              if (x + a < l)
                     fr=0;
              else
                     fr=(k*(x+a-l));

              /* Cálculo do próximo X */
              xmid=x;// Salva o X atual para se tornar o X anterior.

              x = (forca + (a0)*x + (a1)*xold - fr)/(a2);
              // Calcula o X próximo em cima do X atual.

              xold=xmid;
              // Salva o X anterior para se tornar anterior do anterior.

              fprintf(res,"%lf,%lf\n",dt,x);
       // Escreve as variações de vibração com o dominio do tau em i.

              i++;   // Incrementa i.
       }
       while (i < 260000);
       // Fim da integração.

       // Fechamento do arquivo utilizado para resultado.
       fclose(res);


       return 0;
}
```

13 ANEXO II

Tabela 12 - Código Fonte: Modelo de duas massas equilibradas sequencial

Código

```
// Duas_Massas_Equilibradas_Serial2.cpp : Defines the entry point for the
console application.
//

#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <time.h>

/* Variaveis para o dominio de X1. */
long double x1,p1,fc1,d1,x10,x1l0,x1ll0,m1,c1x1,c2x1,c3x1;
long double g1,x1mid,x1old;
/* Variaveis para o dominio de X2. */
long double x2,p2,fc2,d2,x20,x2l0,x2ll0,m2,c1x2,c2x2,c3x2;
long double g2,x2mid,x2old;
/* Variaveis para troca de informações. */
long double fr,k;
/* Valores de entrada e condições iniciais.*/
long double d0,g,num,l,e,A,r;
/* Valores para o cálculo do amortecimento. */
long double w,c1,c2;

/* Usadas para o cálculo do Xt+At. */
long double xold,xmid;

/* Usada como variável do tempo. */
long double i,porc,passo, dt;

/* Usado para o cálculo da frequencia.*/
double pi = 3.14159265358979323846264643;

/* Arquivo com resultado para plotar o gráfico. */
FILE * resx1;
FILE * resx2;

/* Variáveis para cálculo de tempo de execução */
clock_t end, start;


int _tmain(int argc, _TCHAR* argv[])
{
	start = clock(); // Obtém primeiro tempo

	/* Do programa */
	num=50000;		// Passos do sistema. 5000000
	passo = 0.001;		// Tamanho do passo. 0.001

	i=2;   // O laço principal começará a partir do segundo passo.
	g=9.78;			// Gravidade no nível da terra.

	/* Do cabo flexível */
	l=40400;		// Comprimento do cabo flexível.
	e = 2*pow(10.0,11);	// Constante do material.
	A = 1*pow(10.0,-4);	// Área da sessão transversal.
	k = (e*A)/(l);		// Rigidez do cabo flexível.
```

```
        /* Das massas */
        w=2 * pi/86400;      // Frequência da rotação da terra.
        r=40000000/(2*pi);   // Raio da terra

        // Calculo do raio da órbita geossíncrona
        long double ha=(g*(r*r)/(w*w));
        long double ha2=1.0/3.0;
        d0=pow(ha,ha2);
        // Fim do cálculo

        d1=d0+20000;         // Distancia de x1 a terra.
        d2=d0-20000;  // Distancia de x2 a terra. (inclui o cabo flexível)
        m1=1000;                    // Massa 1.
        m2=1000;                    // Massa 2.
        c1=0.005*m1;         // Amortecimento da massa 1.
        c2=0.005*m2;         // Amortecimento da massa 2.
        x1l0=0;                             // Velocidade inicial de x1.
        x2l0=0;                             // Velocidade inicial de x2.
        x10=0;
        x20=0;                      // Posição relativa.

        /* Primeiro Passo */
        fprintf(resx1 = fopen("x1.dat","w"),"0.000000,%Lf\n",x1);
        fprintf(resx2 = fopen("x2.dat","w"),"0.000000,%Lf\n",x2);

        /* Zona do segundo passo. */
        g1=g*pow((r/(d1+x1)),2);    // Constantes gravitacionais.
        g2=g*pow((r/(d2+x2)),2);

        p1=m1*g1;                   // Peso de cada Massa.
        p2=m2*g2;

        fc1 = m1 * (d1+x1) * pow(w,2);    // Forças centrífugas
        fc2 = m2 * (d2+x2) * pow(w,2);
        /* Força de restauração (se menor que comprimento do cabo,
        // Fr nula, caso contrário Fr calculado
        */
        fr=((d1+x1)-(d2+x2)<l)?0:(k*(((d1+x1)-(d2+x2))-l));
        x1ll0 = (fc1 - p1 - fr - c1 * x1l0)/m1; // Aceleração inicial
        x2ll0 = (fc2 - p2 + fr - c2 * x2l0)/m2; // MOD +fr

        /* Passo inicial */
        x1 = x10 + ((x1l0 * passo) + ((1.0/2.0)* x1ll0 * ((passo *
passo)))); //(passo * passo)
        x2 = x20 + ((x2l0 * passo) + ((1.0/2.0)* x2ll0 * ((passo *
passo)))); //(passo * passo)

        fprintf(resx1,"%lf,%Lf\n",passo,x1);
        fprintf(resx2,"%lf,%Lf\n",passo,x2);

        /* Zona do terceiro até N-Ésimo passo. */
        x1old=x1;
        x2old=x2;

        c1x1=(2*m1)/(pow(passo,2)); //(pow(passo,2)
        c2x1=c1/(2*passo)- m1/(pow(passo,2)); //(pow(passo,2)
        c3x1=m1/(pow(passo,2)) + c1/(2*passo); //(pow(passo,2)

        c1x2=(2*m2)/(pow(passo,2));//(pow(passo,2)
        c2x2=c2/(2*passo)- m2/(pow(passo,2));//(pow(passo,2)
        c3x2=m2/(pow(passo,2)) + c2/(2*passo);//(pow(passo,2)
```

```
	do
	{
		/* Zona de atualizações de variáveis. */
		dt=i*passo;

		fr=((d1+x1)-(d2+x2)<l)?0:(k*(((d1+x1)-(d2+x2))-l));

		g1=g*pow((r/(d1+x1)),2);
		g2=g*pow((r/(d2+x2)),2);

		p1=m1*g1;
		p2=m2*g2;

		fc1=m1 * (d1+x1)* pow(w,2);
		fc2=m2 * (d2+x2)* pow(w,2);

		/* Zona de cálculo para atualizações. */
		x1mid=x1;
		x1= (fc1 - p1 - fr + c1x1*x1 + c2x1*x1old)/c3x1;
		x1old=x1mid;

		x2mid=x2;
		x2= (fc2 - p2 + fr + c1x2*x2 + c2x2*x2old)/c3x2; //MOD +fr
		x2old=x2mid;

		i++;

		/* Escrever no arquivo. */
		if (((long int)i % 1000) == 0)
			{
			fprintf(resx1,"%Lf,%Lf\n",dt,x1);
			fprintf(resx2,"%Lf,%Lf\n",dt,x2);
			}

		/* Porcentagem de conclusão. */
		if (((long int)i % 1000000) == 0)
			{
			porc=i/num*100;
			printf("%2.2lf%%\r",porc);
			}
	}
	while (i < num);

	end = clock(); // Obtém tempo final

	/* Fechamento do arquivo do resultado */
	fclose(resx2);
	fclose(resx1);

	//Imprime tempo de execução em ms, dividindo por 1000 obtemos o
valor em segundos
	printf("Tempo gasto: %4.0f ms\n\n",1000*(double)(end-
start)/(double)(CLOCKS_PER_SEC));

	// FIM DO PROGRAMA //
	return 0;
}
```

14 ANEXO III

Tabela 13 - Código fonte: Modelo de duas massas equilibradas paralela

Código

```
// Duas Massas Equilibradas Paralela.cpp : Defines the entry point for
the console application.
#include "stdafx.h"
#include <mpi.h>
#include <stdio.h>
#include <math.h>

#define MAX_BUF 4096

/* Variáveis para o domínio de X1. */
long double x1,p1,fc1,d1,x10,x1l0,x1ll0,m1,c1x1,c2x1,c3x1;
long double g1,x1mid,x1old;

/* Variáveis para o domínio de X2. */
long double x2,p2,fc2,d2,x20,x2l0,x2ll0,m2,c1x2,c2x2,c3x2;
long double g2,x2mid,x2old;

/* Variáveis para troca de informações. */
long double fr,k;

/* Valores de entrada e condições iniciais.*/
long double d0,g,num,l,e,A,r;

/* Valores para o cálculo do amortecimento. */
long double w,c1,c2;

/* Usadas para o cálculo do Xt+At. */
long double xold,xmid;

/* Usada como variável do tempo. */
long double i,porc,passo,passo2,dt;

/* Usado para o cálculo da frequência.*/
double pi = 3.14159265358979323846264338327950288419716939937510;

/* Arquivo com resultado para plotar o gráfico. */
FILE * resx1;
FILE * resx2;

/* Variaveis do MPI para contagem do tempo. */
double startwtime = 0.0, endwtime;

/* Variavies do ambiente MPI. */
int numprocs, myid;
MPI_Status status;
MPI_Request request;


int _tmain(int argc, char* argv[])
{
	MPI_Init(&argc,&argv);
	MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
	MPI_Comm_rank(MPI_COMM_WORLD,&myid);

	startwtime = MPI_Wtime(); // Contador de tempo.
```

```
        /* Do programa */
        num=50000;// Passos do sistema. (50000)
        passo = 0.001;      // Tamanho do passo. (0.001)
        passo2=pow(passo,2);// Passo ao quadrado.
        i=2;   // O laço principal começará a partir do segundo passo.
        g=9.78;             // Gravidade no nível da terra.

        /* Do cabo flexível */
        l = 40400;          // Comprimento do cabo flexível.
        e = 2*pow(10.0,11); // Constante do material.
        A = 1*pow(10.0,-4); // Área da sessão transversal.
        k = (e*A)/(l);      // Rigidez do cabo flexível.

        /* Das massas */
        w = 2 * pi/86400;           // Frequência da rotação da terra.
        r = 40000000/(2*pi);        // Raio da terra

        // Calculo do raio da órbita geossíncrona
        long double ha=(g*(r*r)/(w*w));
        long double ha2=1.0/3.0;
        d0=pow(ha,ha2);
        // Fim do cálculo do raio da órbita geossíncrona

        /* Atribuição da força de restauração inicial.  */
        d1=d0+20000;  // Distancia de x1 a terra.
        x10=0;        // Posição relativa.
        d2=d0-20000;  // Distancia de x2 a terra. (inclui o cabo flexível)
        x20=0;        // Posição relativa.

        /* Divisão do programa em três nós: X1 (calcula FR), X2 */
        if (myid == 0)
        {
               printf("Iniciando processo de cálculo.\n");
               fflush(stdout);
               /* Variáveis das massas  */
               m1=1000;      // Massa 1.
               c1=0.005*m1;  // Amortecimento da massa 1.
               x1l0=0;       // Velocidade inicial de x1.

               /* Primeiro Passo */
               fprintf(resx1 = fopen("x1.dat","w"),"0.000000,%Lf\n",x1);

               /* Zona do segundo passo. */
               g1=g*pow((r/(d1+x1)),2);   // Constantes gravitacionais.
               p1=m1*g1;                  // Peso de cada Massa.
               fc1 = m1 * (d1+x1) * pow(w,2);

               x1ll0 = (fc1 - p1 - fr - c1 * x1l0)/m1;

               x1 = x10 + ((x1l0*passo) + ((1.0/2.0)* x1ll0 * (passo2)));

               fprintf(resx1,"%lf,%Lf\n",passo,x1);

        /* Zona do terceiro até N passo. */
               x1old=x1;

               c1x1=(2*m1)/passo2;
               c2x1=(c1/(2*passo))- (m1/passo2);
               c3x1=(m1/passo2) + (c1/(2*passo));

               /* Início do laço de passos. */
               do
```

```
                        {

        MPI_Recv(&x2,1,MPI_DOUBLE,1,1,MPI_COMM_WORLD,&status);
                        fr=((d1+x1)-(d2+x2)<l)?0:(k*(((d1+x1)-(d2+x2))-l));

                        MPI_Isend(&fr, 1, MPI_DOUBLE, 1, 2,
MPI_COMM_WORLD,&request); // era status

                        dt=i*passo;
                        g1=g*pow((r/(d1+x1)),2);
                        p1=m1*g1;
                        fc1=m1 * (d1+x1)* pow(w,2);

                        x1mid=x1;

                        x1= (fc1 - p1 - fr+c1x1*x1+c2x1*x1old)/c3x1;

        x1old=x1mid;

                        /* Escrever no arquivo. */
                        if (((long int)i % 10000) == 0)
                                {
                                // Escreve no arquivo 1 o resultado de x1.
                                fprintf(resx1,"%Lf,%Lf\n",dt,x1);
                        }

                        /* Escrever na tela. */
                        if (((long int)i % 10000) == 0)
                                {
                                porc=i/num*100;
                                printf("Concluido: %2.2lf%%\r",porc);
                                fflush(stdout);
                                }
                        i++;
                        }
                        while (i < num);

                        /* Fechamento do arquivo de resultados. */
                        fclose(resx1);

                        /* Imprime o tempo do processo. */
                        endwtime = MPI_Wtime(); // Contador de tempo.
                        printf("Tempo de execuçao no processador 0: %f
segundos.\n", endwtime-startwtime);

        } // Fecha nó 0.

        if (myid == 1)
        {
                /* Variáveis das massas  */
                m2=1000;        // Massa 2.
                c2=0.005*m2;  // Amortecimento da massa 2.
                x2l0=0;         // Velocidade inicial de x2.

                /* Primeiro Passo */
                fprintf(resx2 = fopen("x2.dat","w"),"0.000000,%Lf\n",x2);

                /* Zona do segundo passo. */
                g2=g*pow((r/(d2+x2)),2);
                p2=m2*g2;
                fc2 = m2 * (d2+x2) * pow(w,2);
```

```
                x2ll0 = (fc2 - p2 + fr - c2 * x2l0)/m2; // MOD +fr
                x2 = x20 + ((x2l0 * passo) + ((1.0/2.0)* x2ll0 *
(passo2)));

                fprintf(resx2,"%lf,%Lf\n",passo,x2);

                /* Zona do terceiro até N passo. */
                x2old=x2;

                c1x2=(2*m2)/passo2;
                c2x2=(c2/(2*passo))- (m2/passo2);
                c3x2=(m2/passo2) + (c2/(2*passo));

                printf("\n");
                fflush(stdout);

                /* Início do laço de passos. */
                do
                        {
                        MPI_Isend(&x2, 1, MPI_DOUBLE, 0, 1,
MPI_COMM_WORLD,&request); // Envio de X2 (era status)

                        dt=i*passo;

                        g2=g*pow((r/(d2+x2)),2);
                        p2=m2*g2;
                        fc2=m2 * (d2+x2)* pow(w,2);

                        x2mid=x2;


        MPI_Recv(&fr,1,MPI_DOUBLE,0,2,MPI_COMM_WORLD,&status);

                        x2=(fc2 - p2 + fr + c1x2*x2 + c2x2*x2old)/c3x2;
                        // MOD +fr

                        x2old=x2mid;

                        /* Escrever no arquivo. */
                        if (((long int)i % 10000) == 0)
                                {
                                fprintf(resx2,"%Lf,%Lf\n",dt,x2);
                                // Escreve no arquivo 2 o resultado de x2.
                                }
                        i++;
                        }
                        while (i < num);
                        endwtime = MPI_Wtime(); // Contador de tempo.
                        printf("Tempo de execuçao no processador 1: %f
segundos.\n", endwtime-startwtime);

        /* Fechamento do arquivo de resultados. */
                        fclose(resx2);

        }       // Fecha nó 1.

        /* Termina o comunicador. */
        MPI_Finalize();


        return 0;
}
```

15 ANEXO IV

Tabela 14 - Código Fonte: Problema para N satélites sequencial

Código

```
// N_Massas_Sequencial.cpp : Defines the entry point for the console
application.
//

#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <time.h>
#define MAX_NUM_NOS 100
#define MAX_NUM_CABOS 100


int _tmain(int argc, _TCHAR* argv[])
{
	/* Definição de constantes. */
	long double pi,r0,g0,omega;

	/* Definição de entrada e saída. */
	FILE* ENT;
	FILE* SAI;

	/* Variáveis do domínio do programa. */
	int nn, nc, npas, n, t_comeco;
	long double dt;

	/* Mapeamento dos nós, somado à terra. */
	long double m[MAX_NUM_NOS], r[MAX_NUM_NOS], teta[MAX_NUM_NOS],
xa[MAX_NUM_NOS], xp[MAX_NUM_NOS];

	/* Amortecimento. */
	long double cam;

	/* Mapeamento dos cabos. */
	long double ni[MAX_NUM_CABOS], nf[MAX_NUM_CABOS],
ea[MAX_NUM_CABOS], dl[MAX_NUM_CABOS];

	/* Variáveis referentes a força restauradora e a rigidez. */
	long double dli, ak, a, b, c, calfa, alfa, cbeta, frn,
fr[MAX_NUM_NOS];
	int nin, nfi;

	/* Variáveis referentes a inicialização. */
	long double fcen, peso, fam, acel, ck[MAX_NUM_NOS],
cat[MAX_NUM_NOS], can[MAX_NUM_NOS],x[MAX_NUM_NOS];

	/* Variáveis do laço principal. */
	long double pcha, t, np;

	/* Definição de constantes. */
	pi=acos(-1.0);
	r0=20000000/pi;
	g0=9.78;
	omega=2.0*pi/86400;

    /* Definição da entrada. */
    if ((ENT = fopen("entrada.txt","r"))== NULL)
```

```
    printf("Nao pude abrir arquivo de entrada. Verificou a existencia de
entrada.txt?");
    else
       {
       if ((SAI = fopen("saida.dat","w"))== NULL)
       printf("Nao pude abrir arquivo de saida. Disco protegido contra
gravacao?");
    else
    {
       /* Começa a contagem do tempo. */
       t_comeco = time(NULL);

       /* Dados de Entrada. Domínio do programa. */
       fscanf(ENT,"%d%d%d%lf",&nn, &nc, &npas, &dt);          /* Num. Nós,
Num. Cabos, Num. Passos, Tam. do passo. */

       /* Enunciado inicial do programa. */
       printf("Programa para analise dinamica de sistemas de satelites ao
redor da terra.\n");
       printf("-------------------------------------------------------------
---------------\n\n");
       printf("Numero de nos: %d\n",nn);
       printf("Numero de cabos: %d\n",nc);
       printf("Numero de passos do sistema: %d\n",npas);
       printf("Tamanho de cada passo: %2.4lf\n\n",dt);
       printf("-------------------------------------------------------------
---------------\n\n");
       printf("Inicializacao de vetores...");
       /* Inicialização dos vetores. */

       /* Ajuste para vetores que começam em 0. */
       nn--;
       nc--;

       /* Primeiramente uma contagem até o NN, e depois, até o NC. */
       for (n=0 ; n <= nn ; n++)

       m[n]=r[n]=teta[n]=xa[n]=xp[n]=fr[n]=ck[n]=cat[n]=can[n]=x[n]=0;

       for (n=0 ; n <= nc ; n++)
              ni[n]=nf[n]=ea[n]=dl[n]=0;
       printf(" ok.\n");

       /* Mapeamento das massas. */

       /* XA[n] estabelece a inicialização do sistema. Para a solução de
EDO's é necessária a presença de um X[0]. */
       /* Do mesmo modo XP = X' (velocidade inicial). */
       /* Ao mesmo tempo que ele adquiri os valores das posições das
massas, ele as escreve no arquivo de saída. */

       printf("Mapeamento das massas...");
       fprintf(SAI,"0.00000");

       for (n = 0; n <= nn ; n++)
              {
                     fscanf(ENT,"%lf %lf %lf %lf %lf\n", &m[n], &r[n],
&teta[n], &xa[n], &xp[n]);
              }
       printf(" ok.\n");

       /* Amortecimento proporcional às massas. */
```

```
        printf("Leitura do amortecimento...");
        fscanf(ENT,"%lf", &cam);
        printf(" ok.\n");

        /* Mapeamento dos cabos. */
        printf("Mapeamento dos cabos...");
                for (n = 0; n <= nc ; n++)
                {
                fscanf(ENT,"%lf %lf %lf %lf", &ni[n], &nf[n], &ea[n],
&dl[n]);
                }
        printf(" ok.\n");

        /* Imprime as posições iniciais no arquivo. */
        /* A terra não é impressa. */
        for (n = 0; n < nn ; n++)
                {
                        fprintf(SAI," %lf",xa[n]);
                }
        fprintf(SAI,"\n");

        /* Cálculo da rigidez e forças restaurados iniciais. */
        printf("Rigidez e forca restauradora inicial...");

        for (n = 0; n <= nc ; n++)
                {
                /* Rigidez K = EA(i)/L(i). */
                        ea[n]=ea[n]/dl[n];

                /* FR = K(i)*(C(i)-L(i)) */
        /* Adquire informações para o cálculo da força de restauração
        de um determinado cabo. */
                nin=ni[n];
                nfi=nf[n];
                ak=ea[n];
                dli=dl[n];

                /* Verifica se o nó em questão está ligado à terra. */

                if ((nin == nn) || (nfi == nn))
                        {
                                /* O nó em questão é a terra, logo, c = r(nfi)
- r0. */
                                c=r[nfi]-r0;
                                calfa=1;
                                cbeta=-1;
                        }

                /* Não está ligado, é um nó normal. */
                else
                        {
                                a=r[nfi]*sin(teta[nfi]-teta[nin]);
                                b=r[nin]-r[nfi]*cos(teta[nfi]-teta[nin]);
                                c=sqrt(a*a+b*b);
                                calfa=b/c;
                                alfa=acos(calfa);
                                cbeta=cos(pi-alfa-teta[nfi]+teta[nin]);
                        }

                /* Verifica se houve esticamento. */
                if (c < dli)
                        frn=0;
```

```
                else
                        {
                /* Se houve esticamento, é calculado uma nova força de
                restauração normal. */
                        frn=ak*(c-dli);
                                }
                /* Decomposição para encontrar a força de restauração na
                direção radial. */
                fr[nin]+=frn*calfa;
                fr[nfi]+=frn*cbeta;  /* Mesmo para nó final. */
        }
        printf(" ok.\n");

        /* Inicialização para o laço principal */
        printf("Inicializacao para o laco principal...");

        for (n=0; n <= nn ; n++ )
                {
                        fcen=omega*omega*m[n]*r[n];

/* Altera a força gravitacional, dependendo de sua localização. */
                        peso=m[n]*g0*(r0/r[n])*(r0/r[n]);

                        /* Força de amortecimento dependente do nó. */
                        fam=cam*m[n]*xp[n];
                        acel=(fcen-peso-fr[n]-fam)/m[n];
        /* Da inicialização do sistema, oriundo do projeto anterior. */
                        x[n]=xa[n]+xp[n]*dt*0.5*dt*dt*acel;
                        /* Constantes multiplicativas dos membros da EDO. */
                        ck[n]=m[n]*(1/dt/dt+cam/2/dt);
                        cat[n]=2*m[n]/dt/dt;
                        /* Const. do X atual. */
                        can[n]=m[n]*(1/dt/dt-cam/2/dt);
                        /* Const. do X anterior. */
                }

        /* Escreve as posições iniciais das massas, e pula de linha no
arquivo de saída. */
        fprintf(SAI,"%lf",dt);
        for (n = 0; n < nn ; n++)
                {
                        fprintf(SAI," %lf",x[n]);
                }
        fprintf(SAI,"\n");

        printf(" ok.\n");

        /* Integração das diferenças centrais */
        /* Sessão do loop principal do programa */
        printf("Integracao por diferencas finitas...\r");

        t=dt;

        for (np = 1; np <= npas; np++)
                {
                /* Cálculo das forças restauradoras para cada passo. */
                /* Zerar vetor de forças restauradoras a cada passo. */
                        for (n=0; n <= nn; n++)
                                fr[n]=0;

                        for (n=0; n <= nc; n++)
                                {
```

```
                                    nin=ni[n];
                                    nfi=nf[n];
                                    ak=ea[n];
                                    dli=dl[n];
                /* Checa se o cabo em questão está ligado à terra. */
                        if ((nin == nn) || (nfi == nn))
                                {
                /* O nó em questão é a terra, logo, c = r(nfi) - r0. */

                                    c=r[nfi]-r0;
                                    calfa=-1;
                                    cbeta=1;
                                }
                                /* Não está ligado, é um nó normal */
                        else
                                {
                                    /* Não está ligado, é um nó normal */
                                    a=r[nfi]*sin(teta[nfi]-teta[nin]);
                                    b=r[nin]-r[nfi]*cos(teta[nfi]-
teta[nin]);
                                    c=sqrt(a*a+b*b);
                                    calfa=b/c;
                                    alfa=acos(calfa);
                                    cbeta=cos(pi-alfa-teta[nfi]+teta[nin]);
                                }
                        if (c < dli)
                                frn=0;
                        else
                                {
                                frn=ak*(c-dli);
                                }
                        fr[nin]+=frn*calfa;
                        fr[nfi]+=frn*cbeta;
                }

        /* Cálculo da força centrifuga, peso e cálculo de X */
        /* Atualização da dinâmica do sistema para a atualização das
posições das massas */

        for (n=0 ; n <= nn ; n++)
                {
                        fcen=omega*omega*m[n]*r[n];
        /* Altera a força gravitacional, dependendo de sua localização. */
                        peso=m[n]*g0*(r0/r[n])*(r0/r[n]);
        /* Numerador da equação principal. */
                        pcha=fcen-peso-fr[n]+cat[n]*x[n]-can[n]*xa[n];
                        xa[n]=x[n];
                        x[n]=pcha/ck[n];
                        r[n]=r[n]+x[n]-xa[n];
                                }
                        t=t+dt;

        /* Escreve na saida as posições atualizadas das massas, e pula
linha no arquivo de saída. */

                        fprintf(SAI,"%lf",t);
                        for (n = 0; n <= nn - 1 ; n++)
                                {
fprintf(SAI," %lf",x[n]);
                                }
                        fprintf(SAI,"\n");
```

```
                        /* Informa o progresso do processamento ao usuário
*/
                        if (((long int)np % 10000) == 0)
                                {
                                printf("Integracao por diferencas finitas...
%2.1lf%%\r",np/npas*100);
                                }
                }
        printf("%Integracao concluida com sucesso!
\n\n");
        printf("-------------------------------------------------------------
----------------\n\n");
        printf("O tempo total de processamento: %d
segundo(s).\n",time(NULL)-t_comeco);
        printf("A matriz de resultados foi gerada em saida.dat.");

        }

}

        return 0;
}
```

16 ANEXO V

Tabela 15 - Código Fonte: Problema para N satélites paralelo

Código

```
// N_Massas_Paralelo.cpp : Defines the entry point for the console
application.
//

#include "stdafx.h"
#include <mpi.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#define MAX_NUM_NOS 100
#define MAX_NUM_CABOS 100

int i=0,numprocs=0, myid=0;
double startwtime = 0, endwtime=0;
char cont;

/* Análise de condições da comunicação. */
MPI_Status status;

int _tmain(int argc, char* argv[])
{
	/* Definição de constantes. */
	register long double pi=0,r0=0,g0=0,omega=0;

	/* Variáveis do domínio do programa. */
	int nn=0, nc=0, npas=0, dominio[3], n=0, t_comeco=0;
	long double dt=0;

	/* Ponteiros de arquivos de entrada e saída */
	FILE* ENT;
	FILE* SAI;
	char nome_arquivo_entrada[15];
	char nome_arquivo_saida[15],indice_arquivo_saida[10];

	/* Mapeamento dos nós, somado à terra. */
	/* Em frente a cada vetor, uma variavel do mesmo tipo será criada
para que os dados possam ser distribuidos para cada nó. */
	long double m[MAX_NUM_NOS], r[MAX_NUM_NOS], teta[MAX_NUM_NOS],
xa[MAX_NUM_NOS], xp[MAX_NUM_NOS];
	long double mx=0, rx=0, tetax=0, xax=0, xpx=0;

	/* Amortecimento. */
	long double cam;

	/* Mapeamento dos cabos. */
	/* Em frente a cada vetor, uma variavel do mesmo tipo será criada
para que os dados possam ser distribuidos para cada nó. */
	long double ni[MAX_NUM_CABOS], nf[MAX_NUM_CABOS],
ea[MAX_NUM_CABOS], dl[MAX_NUM_CABOS];
	long double nix=0, nfx=0, eax=0, dlx=0;

	/* Variáveis referentes a força restauradora e a rigidez. */
	long double fr[MAX_NUM_NOS];
```

```
        register long double dli, ak, a, b, c, calfa, alfa, cbeta, frn;
        register int nin, nfi;

        /* Variáveis referentes a inicialização. */
        register long double fcen=0, peso=0, fam=0, acel=0, ck=0, cat=0,
can=0,x=0;

        /* Variáveis do laço principal. */
        register long double pcha, t;
        register int np;

        MPI_Init(&argc,&argv);
        MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
        MPI_Comm_rank(MPI_COMM_WORLD,&myid);

        /* Definição de constantes. */
        pi=acos(-1.0);
        r0=20000000/pi;
        g0=9.78;
        omega=2.0*pi/86400;

        if (myid == 0)
                {
                /* Definição de entrada e saída. */
                strcpy(nome_arquivo_entrada,"entrada.txt");

                strcpy(nome_arquivo_saida,"massa_");
                itoa(myid,indice_arquivo_saida,10);
                strcat(nome_arquivo_saida,indice_arquivo_saida);
                strcat(nome_arquivo_saida,".dat");

            /* Abrindo arquivos. */
            if ((ENT = fopen(nome_arquivo_entrada,"r"))== NULL)
               {printf("Nao pude abrir arquivo de entrada. Verificou a
existencia de entrada.txt?"); fflush(stdout);}
            else
               {
               if ((SAI = fopen(nome_arquivo_saida,"w"))== NULL)
               {printf("Nao pude abrir arquivo de saida. Disco protegido
contra gravacao?"); fflush(stdout);}
            else
            {
                        /* Dados de Entrada. Domínio do programa. */
                        /* Num. Nós, Num. Cabos, Num. Passos, Tam. do passo.
*/
                        fscanf(ENT,"%d%d%d%lf",&nn, &nc, &npas, &dt);

                        /* Enunciado inicial do programa. */
                        printf("\n\n\n\n\n\n\n\n\n\nPrograma para analise
dinamica de sistemas de satelites ao redor da terra.\n");
                        printf("-----------------------------------------
----------------------------\n\n");
                        printf("Numero de nos: %d\n",nn);
                        printf("Numero de cabos: %d\n",nc);
                        printf("Numero de passos do sistema: %d\n",npas);
                        printf("Tamanho de cada passo: %lf\n\n",dt);
                        printf("-----------------------------------------
----------------------------\n\n");
                        fflush(stdout);

                        /* Ajuste das variáveis para começarem os vetores no
0. */
```

```
                    nn--;
                    nc--;

/* Os dados são empacotados em domínio, para que haja transferência
única. */
                    dominio[0]=nn;
                    dominio[1]=nc;
                    dominio[2]=npas;

/* Transferência dos dados do domínio do programa empacotados em um
vetor. */
                    printf("Transferencia de dados de entrada... ");
fflush(stdout);
                    MPI_Barrier(MPI_COMM_WORLD);
                    MPI_Bcast (&dominio, 3, MPI_INT, 0, MPI_COMM_WORLD);
                    /* Por Dt ser float, é mandado separadamente. */
                    MPI_Barrier(MPI_COMM_WORLD);
                    MPI_Bcast (&dt, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
                    printf("ok.\n"); fflush(stdout);

                    printf("Inicializacao de vetores...");
                    /* Inicialização dos vetores. */

                    /* Primeiramente uma contagem até o NN, e depois, até
o NC. */
                    for (n=0 ; n <= nn ; n++)
                           m[n]=r[n]=teta[n]=xa[n]=xp[n]=fr[n]=0;

                    for (n=0 ; n <= nc ; n++)
                           ni[n]=nf[n]=ea[n]=dl[n]=0;
                    printf(" ok.\n");fflush(stdout);

/* Mapeamento das massas. */
/* XA[n] estabelece a inicialização do sistema. Para a solução de EDO's é
necessária a presença de um X[0]. */
/* Do mesmo modo XP = X' (velocidade inicial). */
/* Ao mesmo tempo que ele adquiri os valores das posições das massas, ele
as escreve no arquivo de saída. */

                    printf("Mapeamento das massas...");fflush(stdout);
                    fprintf(SAI,"0.00000");

                    for (n = 0; n <= nn ; n++)
                           {
                                  fscanf(ENT,"%lf %lf %lf %lf %lf\n",
&m[n], &r[n], &teta[n], &xa[n], &xp[n]);
                           }
                    printf(" ok.\n");fflush(stdout);

                    /* Distribuição dos elementos dos vetores captados
para cada massa correspondente. */
                    printf("Transferencia do mapeamento das massas... ");
fflush(stdout);
                    MPI_Barrier(MPI_COMM_WORLD);

       MPI_Scatter(&m,1,MPI_DOUBLE,&mx,1,MPI_DOUBLE,0,MPI_COMM_WORLD);

       MPI_Scatter(&r,1,MPI_DOUBLE,&rx,1,MPI_DOUBLE,0,MPI_COMM_WORLD);

       MPI_Scatter(&teta,1,MPI_DOUBLE,&tetax,1,MPI_DOUBLE,0,MPI_COMM_WORL
D);
```

```
	MPI_Scatter(&xa,1,MPI_DOUBLE,&xax,1,MPI_DOUBLE,0,MPI_COMM_WORLD);

	MPI_Scatter(&xp,1,MPI_DOUBLE,&xpx,1,MPI_DOUBLE,0,MPI_COMM_WORLD);
                printf(" ok.\n");fflush(stdout);

                /* Amortecimento proporcional às massas. */
                printf("Leitura e transferencia do
amortecimento...");fflush(stdout);
                fscanf(ENT,"%lf", &cam);
                MPI_Barrier(MPI_COMM_WORLD);
                MPI_Bcast (&cam, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
                printf(" ok.\n");fflush(stdout);

                /* Mapeamento dos cabos. */
                printf("Mapeamento dos cabos...");

                for (n = 0; n <= nc ; n++)
                        {
                        fscanf(ENT,"%lf %lf %lf %lf", &ni[n], &nf[n],
&ea[n], &dl[n]);
                /* Calculo da rigidez K = EA(i)/L(i). */
                                ea[n]=ea[n]/dl[n];
                        }
                printf(" ok.\n");fflush(stdout);

        /* Imprime a posição inicial no arquivo. */
                fprintf(SAI," %lf",xax);
                fprintf(SAI,"\n");

        /* Cálculo da rigidez e forças restaurados iniciais. */
                printf("Forca restauradora inicial...");
                for (n = 0; n <= nc ; n++)
                {
        /* FR = K(i)*(C(i)-L(i)) */
        /* Adquire informações para o cálculo da força de restauração
        *  de um determinado cabo. */
                        nin=ni[n];
                        nfi=nf[n];
                        ak=ea[n];
                        dli=dl[n];

        /* Verifica se o nó em questão está ligado à terra. */
                        if ((nin == nn) || (nfi == nn))
                                {
        /* O nó em questão é a terra, logo, c = r(nfi) - r0. */
                                        c=r[nfi]-r0;
                                        calfa=1;
                                        cbeta=-1;
                                }

                        /* Não está ligado, é um nó normal. */
                        else
                                {
                                        a=r[nfi]*sin(teta[nfi]-
teta[nin]);
                                        b=r[nin]-r[nfi]*cos(teta[nfi]-
teta[nin]);
                                        c=sqrt(a*a+b*b);
                                        calfa=b/c;
                                        alfa=acos(calfa);
                                        cbeta=cos(pi-alfa-
teta[nfi]+teta[nin]);
```

```
                                        }

                                /* Verifica se houve esticamento. */
                                if (c < dli)
                                        frn=0;
                                else
                                        {
/* Se houve esticamento, é calculado uma nova força de restauração
normal. */
                                                frn=ak*(c-dli);
                                        }
                                fr[nin]+=frn*calfa;
// Decomposição para encontrar a força de restauração na direção radial.
                                fr[nfi]+=frn*cbeta;  /* Mesmo para nó final. */
                        }
                        printf(" ok.\n");fflush(stdout);

                        /* Transferencia do vetor Fr atualizado. */
                        MPI_Barrier(MPI_COMM_WORLD);
                        MPI_Bcast (&fr, nn+1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

                        /* Inicialização para o laço principal */
                        printf("Inicializacao para o laco
principal...");fflush(stdout);
                        fcen=omega*omega*mx*rx;
                        peso=mx*g0*(r0/rx)*(r0/rx);
       /* Altera a força gravitacional, dependendo de sua localização. */
                        fam=cam*mx*xpx;                                         /*
Força de amortecimento dependente do nó. */
                        acel=(fcen-peso-fr[myid]-fam)/mx;
                        x=xax+xpx*dt*0.5*dt*dt*acel; /* Da inicialização do
sistema, oriundo do projeto anterior. */

                        /* Constantes multiplicativas dos membros da EDO. */
                        ck=mx*(1/dt/dt+cam/2/dt);
                        cat=2*mx/dt/dt;             /* Const. do X atual. */
                        can=mx*(1/dt/dt-cam/2/dt); /* Const. do X anterior.
*/

                        /* Escreve as posições iniciais das massas, e pula de
linha no arquivo de saída. */
                        fprintf(SAI,"%lf %lf\n",dt,x);
                        printf(" ok.\n");fflush(stdout);

                        /* Integração das diferenças centrais */
                        /* Sessão do loop principal do programa */
                        printf("Integracao por diferencas
finitas...\r");fflush(stdout);

                        t=dt;

                        /* Começa a contagem do tempo. */
                        t_comeco = time(NULL);

                        /* Sincronizador */
                        MPI_Barrier(MPI_COMM_WORLD);

                        for (np = 1; np <= npas; np++)
                                {
       /* Cálculo das forças restauradoras para cada passo. */
       /* Zerar vetor de forças restauradoras a cada passo. */
                                        for (n=0; n <= nn; n++)
```

```
                                        fr[n]=0;

      /* Recebimento de todos os raios de todas as massas do sistema. */

                                    /* Sincronização */
                                    if (!(np % 20))
                                          {
                                          MPI_Barrier(MPI_COMM_WORLD);
                                          }

      MPI_Gather(&rx,1,MPI_DOUBLE,&r,1,MPI_DOUBLE,0,MPI_COMM_WORLD,statu
s);

                                    for (n=0; n <= nc; n++)
                                          {
                                                nin=ni[n];
                                                nfi=nf[n];
                                                ak=ea[n];
                                                dli=dl[n];

                        /* Checa se o cabo em questão está ligado à terra. */

                                    if ((nin == nn) || (nfi == nn))
                                                {
                  /* O nó em questão é a terra, logo, c = r(nfi) - r0. */
                                                      c=r[nfi]-r0;
                                                      calfa=-1;
                                                      cbeta=1;
                                                }
                                    /* Não está ligado, é um nó normal */
                                    else {
                                    /* Não está ligado, é um nó normal */

      a=r[nfi]*sin(teta[nfi]-teta[nin]);
                                                      b=r[nin]-
r[nfi]*cos(teta[nfi]-teta[nin]);

      c=sqrt(a*a+b*b);
                                                      calfa=b/c;


      alfa=acos(calfa);

      cbeta=cos(pi-alfa-teta[nfi]+teta[nin]);
                                                }
                                                if (c < dli)
                                                      frn=0;
                                                else
                                                {
                                                      frn=ak*(c-
dli);
                                                }
                                                fr[nin]+=frn*calfa;
                                                fr[nfi]+=frn*cbeta;
                                          }

                                    /* Envio de Fr para todos os nós. */
                                    MPI_Bcast (&fr, nn+1, MPI_DOUBLE, 0,
MPI_COMM_WORLD);

                  /* Cálculo da força centrifuga, peso e cálculo de X */
                  /* Atualização da dinâmica do sistema para a atualização das
posições das massas */
```

```
                                    fcen=omega*omega*mx*rx;
                                    peso=mx*g0*(r0/rx)*(r0/rx);
/* Altera a força gravitacional, dependendo de sua localização. */
                                    pcha=fcen-peso-fr[myid]+cat*x-can*xax;
/* Numerador da equação principal. */
                                    xax=x;
                                    x=pcha/ck;
                                    rx=rx+x-xax;
                                    t=t+dt;

/* Escreve na saida as posições atualizadas das massas, e pula linha no
arquivo de saída. */
                                    fprintf(SAI,"%lf %lf\n",t,x);

            /* Informa o progresso do processamento ao usuário */
                                    if (!(np % 1000))
                                    {
                                          printf("Integracao por
diferencas finitas... %d%%\r",np);fflush(stdout);
                                    }

                              }
                        printf("%Integracao concluida com sucesso!
\n\n");
                        printf("-------------------------------------------
-----------------------------\n\n");
                        printf("O tempo total de processamento: %d
segundo(s).\n",time(NULL)-t_comeco);
                        fclose(SAI);

                        /* Transferencia dos arquivos para o nó principal. */
                        MPI_Finalize();
              }
              }
              }
       else
              {
              /* Definição de entrada e saída. */
              itoa(myid,indice_arquivo_saida,10);
              strcpy(nome_arquivo_saida,"massa_");
              strcat(nome_arquivo_saida,indice_arquivo_saida);
              strcat(nome_arquivo_saida,".dat");

           /* Abrindo arquivos. */
           if ((SAI = fopen(nome_arquivo_saida,"w"))== NULL)
              {printf("Nao pude abrir arquivo de saida. Disco protegido
contra gravacao?"); fflush(stdout);}
           else
           {
                        /* Recebimento do pacote de dados de Entrada. Domínio
do programa. */
                        MPI_Barrier(MPI_COMM_WORLD);
                        MPI_Bcast (&dominio, 4, MPI_INT, 0, MPI_COMM_WORLD);
                        MPI_Barrier(MPI_COMM_WORLD);
                        MPI_Bcast (&dt, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
                        nn=dominio[0];
                        nc=dominio[1];
                        npas=dominio[2];

                        /* Inicialização de vetores. */
                        for (n=0 ; n <= nn ; n++)
```

```
                        m[n]=r[n]=teta[n]=xa[n]=xp[n]=fr[n]=0;

                    for (n=0 ; n <= nc ; n++)
                        ni[n]=nf[n]=ea[n]=dl[n]=0;

                    /* Recebimento do mapeamento das massas. */
                    /* Escreve primeiro passo.*/
                    fprintf(SAI,"0.00000");

                    /* A massa recebida é armazenada na variavel massa.
*/
                    MPI_Barrier(MPI_COMM_WORLD);

      MPI_Scatter(&m,1,MPI_DOUBLE,&mx,1,MPI_DOUBLE,0,MPI_COMM_WORLD);

      MPI_Scatter(&r,1,MPI_DOUBLE,&rx,1,MPI_DOUBLE,0,MPI_COMM_WORLD);

      MPI_Scatter(&teta,1,MPI_DOUBLE,&tetax,1,MPI_DOUBLE,0,MPI_COMM_WORL
D);

      MPI_Scatter(&xa,1,MPI_DOUBLE,&xax,1,MPI_DOUBLE,0,MPI_COMM_WORLD);

      MPI_Scatter(&xp,1,MPI_DOUBLE,&xpx,1,MPI_DOUBLE,0,MPI_COMM_WORLD);

                    /* Recebimento do amortecimento proporcional às
massas. */
                    MPI_Barrier(MPI_COMM_WORLD);
                    MPI_Bcast (&cam, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

                    /* Imprime as posições iniciais no arquivo. */
                    /* A terra não é impressa. */
                    fprintf(SAI," %lf",xax);
                    fprintf(SAI,"\n");

                    /* Recebe o vetor Fr atualizado. */
                    MPI_Barrier(MPI_COMM_WORLD);
                    MPI_Bcast (&fr, nn+1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

                    /* Inicialização para o laço principal */
                    fcen=omega*omega*mx*rx;
                    peso=mx*g0*(r0/rx)*(r0/rx);/* Altera a força
gravitacional, dependendo de sua localização. */
                    fam=cam*mx*xpx;
/* Força de amortecimento dependente do nó. */
                    acel=(fcen-peso-fr[myid]-fam)/mx;
                    x=xax+xpx*dt*0.5*dt*dt*acel;
/* Da inicialização do sistema, oriundo do projeto anterior. */

                    /* Constantes multiplicativas dos membros da EDO. */
                    ck=mx*(1/dt/dt+cam/2/dt);
                    cat=2*mx/dt/dt;                          /* Const. do
X atual. */
                    can=mx*(1/dt/dt-cam/2/dt);
                    /* Const. do X anterior. */
                    /* Escreve as posições iniciais das massas, e pula de
linha no arquivo de saída. */
                    fprintf(SAI,"%lf %lf \n",dt,x);

                    /* Integração das diferenças centrais */
                    /* Sessão do loop principal do programa */
                    t=dt;
```

```
                        /* Sincronizador */
                        MPI_Barrier(MPI_COMM_WORLD);

                        for (np = 1; np <= npas; np++)
                                {
                /* Envio de todos os raios de todas as massas do sistema. */
                                /* Sincronização */
                                if (!(np % 20))
                                        {
                                        MPI_Barrier(MPI_COMM_WORLD);
                                        }

        MPI_Gather(&rx,1,MPI_DOUBLE,&r,1,MPI_DOUBLE,0,MPI_COMM_WORLD,statu
s);

        /* Cálculo da força centrifuga, peso e cálculo de X */
        /* Atualização da dinâmica do sistema para a atualização das
        posições das massas */

                                fcen=omega*omega*mx*rx;
                                peso=mx*g0*(r0/rx)*(r0/rx); /* Altera a
força gravitacional, dependendo de sua localização. */

                            /* Recebimento do vetor Fr atualizado. */
                                MPI_Bcast (&fr, nn+1, MPI_DOUBLE, 0,
MPI_COMM_WORLD);

                                pcha=fcen-peso-fr[myid]+cat*x-can*xax;
/* Numerador da equação principal. */
                                xax=x;
                                x=pcha/ck;
                                rx=rx+x-xax;
                                t=t+dt;

/* Escreve na saida as posições atualizadas das massas, e pula linha no
arquivo de saída. */
                                fprintf(SAI,"%lf %lf\n",t,x);
                            }
                        fclose(SAI);

                        /* Transferencia dos arquivos para o nó principal. */
                        MPI_Finalize();
                }
                }
                return 0;
}
```