UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ESTRATÉGIAS DE PARALELISMO COM GPGPU PARA OTIMIZAÇÃO DO PROCESSAMENTO DO CÁLCULO DO FLUXO DE CARGA EM SISTEMAS ELÉTRICOS DE POTÊNCIA

IGOR MEIRELES DE ARAÚJO

DM:15/2017

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2017

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

IGOR MEIRELES DE ARAÚJO

ESTRATÉGIAS DE PARALELISMO COM GPGPU PARA OTIMIZAÇÃO DO PROCESSAMENTO DO CÁLCULO DO FLUXO DE CARGA EM SISTEMAS ELÉTRICOS DE POTÊNCIA

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2017

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

IGOR MEIRELES DE ARAÚJO

ESTRATÉGIAS DE PARALELISMO COM GPGPU PARA OTIMIZAÇÃO DO PROCESSAMENTO DO CÁLCULO DO FLUXO DE CARGA EM SISTEMAS ELÉTRICOS DE POTÊNCIA

Dissertação submetida à Banca Examinadora do Programa de Pós-graduação em Engenharia Elétrica da UFPA para a obtenção do Grau de Mestre em Engenharia Elétrica na área de Computação Aplicada, elaborada sob a orientação do Prof. Dr. Ádamo Lima de Santana.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2017

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ESTRATÉGIAS DE PARALELISMO COM GPGPU PARA OTIMIZAÇÃO DO PROCESSAMENTO DO CÁLCULO DO FLUXO DE CARGA EM SISTEMAS ELÉTRICOS DE POTÊNCIA

AUTOR: IGOR MEIRELES DE ARAÚJO

DISSERTAÇÃO DE MESTRADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADDA ADEQUADA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE COMPUTAÇÃO APLICADA.

APROVADA EM ____ / ____ / _____

BANCA EXAMINADORA:

Prof. Dr. Ádamo Lima de Santana
(ORIENTADOR – UFPA)

Prof. Dr. Josivaldo de Souza Araújo
(MEMBRO EXTERNO – UFPA-PPGCC)

Prof. Dr. Diego Lisboa Cardoso
(MEMBRO INTERNO– UFPA-PPGEE)

VISTO:

Prof. Dr. Evaldo Gonçalves Pelaes
(COORDENADOR DO PPGEE/ITEC/UFPA)

AGRADECIMENTOS

Primeiramente, agradeço a Deus por todas as oportunidades a mim dadas e principalmente pela minha saúde e a de meus familiares, pois unidos podemos superar os obstáculos impostos pela vida. Obstáculos estes que foram impostos desde meu nascimento, mas que com a fé de todos ao meu redor foram superados.

Agradeço aos meus pais e aos meus irmãos e familiares que confiaram em mim e acreditaram na minha capacidade, me deram suporte e condições de chegar aonde eu cheguei hoje. Também agradeço a minha amiga, companheira e namorada Thamna Maíra, que também me ajudou a enfrentar as dificuldades impostas e compreendeu a importância que este trabalho possui para mim.

Agradeço ao meu orientador, Prof. Dr. Ádamo Lima de Santana, que o considero como um pai no meio acadêmico, pois me acolheu, orientou, enxergou minhas dificuldades e me mostrou o caminho para chegar onde estou hoje, sempre acreditando em mim.

Agradeço aos meus amigos Iago Cavalcante, Gabriel Viana, Alison Rejo e Tiago Rodrigues que desde o início do ensino superior estiveram comigo, cada um apoiando o outro.

Agradeço aos meus colegas de trabalho do LINC que estiveram comigo desde o início desta jornada e aqueles que encontrei ao longo do caminho e, em especial, ao Prof. Dr. Fábio Manoel França Lobato, que também me auxiliou e contribuiu para a realização das minhas conquistas.

Por último, mas não menos importante, agradeço à banca examinadora desta Dissertação de Mestrado, os Professores Dr. Diego Lisboa Cardoso e Dr. Josivaldo de Souza Araújo, que aceitaram em avaliar este trabalho e contribuir para sua excelência. Também a fundação CAPES, a Universidade Federal do Pará e seu Programa e Pós-Graduação em Engenharia Elétrica.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AVR	*Automatic Voltage Regulator*
CDP	*CUDA Dynamic Parallelism*
CPU	Central Processing Unit
CUDA	*Compute Unified Device Architecture*
FACTS	*Flexible Alternating Current Transmission System*
FC	Fluxo de Carga
FDPF	*Fast Decoupled Power Flow*
GS	Gauss-Seidel
GPGPU	*General Purpose Graphics Processing Unit*
GPU	*Graphics Processing Unit*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
MKL	*Math Kernel Library*
MP	Multiprocessador
NLTC	*No Load Tap Changer*
NR	Newton-Raphson
OLTC	*On Load Tap Changer*
PEGASE	*Pan European Grid Advanced Simulation and State Estimation*
SEP	Sistema Elétrico de Potência
ShR	*Shunt Reactor*
ULA	Unidade Lógica e Aritmética

RESUMO

O cálculo do fluxo de carga provê informações fundamentais em um sistema elétrico de potência, informações necessárias para que sejam realizados estudos nos sistemas. No entanto, o fluxo de carga só pode ser realizado em estado de regime permanente. Caso o sistema sofra alguma alteração, seja por variação nas cargas ou modificações dos equipamentos de controle, este cálculo é necessário ser refeito. Por essa necessidade, de constantemente ter que realizar o controle no fluxo de carga, começou-se uma busca por otimizar o tempo necessário desta tarefa. Uma das soluções abordadas para isso foi a utilização de computação paralela, a qual começou a ser utilizada a *General Purpose Graphics Processing Unit* (GPGPU) como uma alternativa de melhor custo benefício para execuções em arquiteturas paralelas, que consiste na utilização de *Graphic Processing Units* (GPU) não somente para processamento gráfico, mas também para propósitos gerais. Diversos trabalhos têm tirado proveito da utilização de GPGPU nos cálculos do fluxo de carga, contudo, não há um consenso sobre qual estratégia utilizar para paralelizar neste tipo de hardware, ficando a cargo de cada autor o trabalho de desenvolver seu próprio método, dificultando a utilização da arquitetura para a implementação desses cálculos, tanto para fins acadêmicos, quanto para o mercado. Pela falta de um consenso e palas divergências encontradas nos trabalhos, esta dissertação visa analisar as etapas do fluxo de carga, identificando quais estão mais aptas a paralelização em GPGPU com o intuito de realizar múltiplos cálculos do fluxo de carga simultâneos e reduzir o tempo necessário para o processamento, difundindo uma estratégia eficiente para sistemas de larga escala no mercado e no meio acadêmico, facilitando a replicação para trabalhos futuros com utilização de metaheurísticas para otimização de sistemas elétricos de potência.

Palavras-chave: *Sistemas Elétricos de Potência, Fluxo de Carga, GPGPU, CUDA*

ABSTRACT

The load flow calculation provides fundamental information for an electric power system. However, the load flow can only be carried out in the steady state, in the event of a system suffering any change, by variation in the loads or modifications of the control equipment, this calculation is necessary to be redone. Because of this need, frequently have to perform the load flow, a research has begun to optimize the time needed for this task. A General-Purpose Graphic Processing Unit (GPGPU) as a cost-effective alternative to parallel architecture runs, which has a GPU not only for graphics purposes but also for general purposes. Several works were taken for the use of GPGPU in load flow calculations, there is no consensus on the content of the material, being in charge of each one of the work of its own method, making it difficult to use the architecture for an implementation of calculations, both for academic purposes and for the market. Due to the lack of consensus and differences found in the work, this dissertation aims to analyze the steps of the load flow, identifying which is more suitable to parallelize in GPGPU in order to perform simultaneous load flow calculations and reduces the time required for the processing, an efficient strategy for large scale systems in the market and not academic environment, facilitate the replication for future works using metaheuristics for optimization of power electrical systems.

Keywords: *Electric Power System, Power Flow, GPGPU, CUDA*

1 INTRODUÇÃO

1.1 MOTIVAÇÃO E DESCRIÇÃO GERAL DO PROBLEMA

O primeiro Sistema Elétrico de Potência com distribuição de energia para consumidores finais surgiu a partir da *Pearl Street Station*, a primeira fonte geradora de energia a vapor, desenvolvida por Thomas Edison em *New York* no ano de 1882. A *Pearl Street Station* inicialmente tinha potência para energizar 3.000 lâmpadas para 59 consumidores, utilizava corrente direta e devido as dificuldades de transporte desse tipo de energia, a estação supria eletricidade apenas em um raio de 800m de distância.

Atualmente, a energia elétrica é um recurso importante que contribui na melhoria da qualidade de saúde, segurança e acesso a informação dos seres humanos. Segundo o departamento de energia dos Estados Unidos, no ano de 2014, foram consumidos 20.7 trilhões de Quilowatt-hora (kWh) no mundo. Esse crescimento na demanda de energia elétrica, tornou os Sistemas Elétricos de Potência cada vez mais complexos.

O Fluxo de Carga é um estudo responsável por fornecer os dados das tensões e potências do Sistema Elétrico de Potência com base nos valores de cargas consumidas. A partir das informações resultantes, podem ser realizados outros estudos e análises nos sistemas, como: planejamentos futuros para o acréscimo de equipamentos ou componentes; analises dos sistemas em caso de contingências no fornecimento de energia elétrica; otimização dos sistemas para redução da perda de potência ativa durante a transmissão.

O Fluxo de carga é realizado em regime permanente (quando não há variação no sistema) devido os cálculos se basearem no consumo de carga. A exemplo, considerando uma indústria que usa de equipamentos pesados com alto consumo de energia, o desligamento das máquinas por motivo de intervalo dos funcionários causa uma variação na carga consumida no Sistema Elétrico de Potência, sendo necessário recalcular o Fluxo de Carga com os valores de consumo atuais. Adicionalmente, durante a transmissão da energia elétrica há diversos equipamentos de controle de tensão e potência reativa (VQC) ajustados para a carga anterior, tornando-se necessário realizar diversos cálculos das tensões e potências com diferentes combinações de valores dos equipamentos de controle, de forma a otimizar o sistema.

Todo esse processamento fora demandado apenas por causa do intervalo dos funcionários de uma indústria. Porém, e se ao término do intervalo na indústria em questão, ou em outra pertencente ao mesmo Sistema Elétrico de Potência, ocorrerem problemas técnicos nos equipamentos e forçarem o desligamento? Neste caso, todas as vezes em que as cargas consumidas são alteradas é necessário refazer esse processamento, para que se diminua o desperdício de energia elétrica e, consequentemente, os custos monetários na sua geração.

1.2 PROBLEMÁTICAS E LIMITAÇÕES ATUAIS

Nas décadas de 1970 e 1980, iniciaram-se os desenvolvimentos de trabalhos de pesquisa com o objetivo de reduzir o tempo de processamento do fluxo carga, onde foram desenvolvidos métodos de solução do Fluxo de Carga menos custosos computacionalmente como o Desacoplado Rápido(STOTT; ALSAC, 1974) e com a abordagem da computação paralela (HOUSOS; OMAR WING, 1979; RAFIAN; STERLING; IRVING, 1985; WANG et al., 1989). Tais trabalhos, em sua maioria, criavam algoritmos paralelos, porém não os testavam em arquiteturas com múltiplas unidades logicas aritméticas.

Na década de 1990, a abordagem mais utilizada no *design* de métodos paralelos para o cálculo do Fluxo de Carga, era a paralelização das tarefas e estruturas contidas nos métodos sequenciais bem consolidados na área (HUANG; ABUR, 1990; LAU; TYLAVSKY; BOSE, 1991), como Gauss-Seidel (HUANG; ONGSAKUL, 1994), Newton-Raphson (AMANO; ZECEVIC; SILJAK, 1996; JUN QIANG WU; BOSE, 1995) e Desacoplado Rápido (EL-KEIB; DING; MARATUKULAM, 1994), embora alguns métodos específicos para ambientes paralelos também tenham sido desenvolvidos (CHEN; BERRY, 1993; WANG; HADJSAID; SABONNADIÈRE, 1999).

Como resultado da busca por ambientes de baixo custo no início do século XXI, as *Graphic Processing Units* (GPU) começaram a ser utilizadas como alternativa de ambiente paralelo para propósitos gerais (GPGPU), dado seu desempenho em operações de ponto flutuante e o baixo custo de aquisição do hardware. Aliado a isso, a busca pela padronização da programação paralela motivou o desenvolvimento de modelos de programação específicos para GPU.

A plataforma *Computer Unified Device Architecture* (CUDA) (NVIDIA CORPORATION, 2016a) desenvolvida pela NVIDIA; surgiu como um novo modelo para a programação em GPU e é considerada uma das mais utilizadas. Tal plataforma passou a ser utilizada, também, na paralelização de métodos de Fluxo de Carga(LI; LI, 2014; MEI YANG et al., 2012), apresentando bom desempenho quando aplicada a Sistemas Elétricos de Potência de larga escala.

Entretanto, não há na presente literatura um padrão de paralelização do Fluxo de Carga na plataforma CUDA, o que dificulta a replicação e utilização de forma eficiente da GPU para este fim, apesar de todos os trabalhos apresentarem um benefício com a utilização do periférico de computação gráfica.

Devido à falta de padronização, cada autor fica encarregado de desenvolver sua própria metodologia, ocasionando ocorrência de algumas divergências entre os trabalhos, como por exemplo, a opção pelo uso de matrizes densas ou esparsas. Pela pouca conectividade em sua estrutura, o Sistema Elétrico de Potência tem uma característica representativa de esparsidade; porém, devido à arquitetura do *hardware* da placa de vídeo, a plataforma CUDA tem um maior aproveitamento com matrizes densas. Diante disso, qual abordagem deve ser utilizada para o Fluxo de Carga? E quais etapas deste devem ser paralelizadas? Todas ou apenas as que consomem maior processamento?

1.3 OBJETIVOS GERAIS

Diante do exposto, este trabalho consiste em um estudo de estratégias de paralelização de execuções múltiplas dos cálculos de Fluxo de Carga em GPGPU, investigando qual abordagem de matrizes é melhor para ser utilizada, visando a realização de tal estudo com testes em sistemas reais (com milhares de barras elétricas), e também analisar quais etapas do Fluxo de Carga melhor se adequam a arquitetura paralela. Como objetivos específicos considera-se: realizar múltiplos cálculos de tensão e potência concorrentemente; obter uma reposta rápida do processamento exigido pelo Fluxo de Carga; testar estratégias em Sistemas Elétricos de Potência de larga escala; e disponibilizar os códigos para que sejam utilizados como base, facilitando, com isso, a replicação deste trabalho, tanto para comunidade acadêmica quanto para o mercado.

1.4 OBJETIVOS ESPECÍFICOS

A realização de múltiplos cálculos dos Fluxos de Carga simultâneos desenvolvidos nesta dissertação, tem o intuito de servir como referência para trabalhos futuros que utilizarão algoritmos baseados em população para otimizar os Sistemas Elétricos de Potência, bem como obter um melhor desempenho usando mais recursos disponíveis pela GPU, ao realizar os cálculos concorrentemente, uma vez que dependendo do tamanho do sistema ou da placa de vídeo que realiza o processamento, uma execução do Fluxo de Carga não irá usufruir de todos os recursos disponíveis.

A resposta rápida do processamento exigido pelo Fluxo de Carga será obtida a partir da paralelização com GPGPU. Para isso, serão realizados testes com bibliotecas e diferentes estratégias, onde busca-se tirar o maior aproveitamento da GPU e também da CPU, quando a tarefa não se adaptar totalmente à plataforma paralela.

Vale ressaltar que, este trabalho tem o intuito de formular uma metodologia padrão para paralelização do Fluxo de Carga em GPU, visto que não existe uma definida disponível para tal finalidade. Além disso, os códigos serão disponibilizados no GITHUB – serviço de *Web Hosting* compartilhado para projetos que usam o controle de versionamento Git – para facilitar a usabilidade e replicação do trabalho no meio acadêmico e comercial.

Outro ponto importante é que as simulações serão realizadas com casos de testes (de domínio público) de Sistemas Elétricos de Potência de larga escala. Visto que há uma tendência no crescimento dos sistemas e também uma agregação de diferentes tipos de geração de energia como hidroelétrica, eólica, fotovoltaica e nuclear.

1.5 ESTRUTURA DA DISSERTAÇÃO

A presente dissertação está organizada como segue:

No Capítulo 2 são apresentados os principais conceitos de um sistema elétrico de potência, quais são seus principais componentes e como é realizado o controle da energia elétrica. Serão fornecidas informações necessárias para o estudo de caso desta dissertação, que são o cálculo de Fluxo de Carga e os componentes de controle que o influenciam.

No Capítulo 3 será apresentado um pouco sobre o contexto histórico do processamento em GPU para propósitos gerais, a plataforma CUDA com suas principais definições e a arquitetura de *hardware* das placas gráficas. Diante o exposto neste capítulo, será possível ter uma compreensão maior acerca da paralelização em GPU, o que contribuirá para o desenvolvimento das estratégias de paralelização do Fluxo de Carga para múltiplas execuções e melhorará a eficiência do processamento, reduzindo o tempo necessário.

No Capítulo 4, alguns trabalhos relacionados a pesquisa são discutidos e assim será possível obter referências das metodologias de paralelização utilizadas e analisar divergências entre os trabalhos. A partir disso, serão realizadas estratégias de paralelização e testes para solucionar as divergências encontradas.

No Capítulo 5 serão apresentadas estratégias de paralelização elaboradas nesta dissertação, bem como suas principais vantagens e desvantagens.

Em seguida, o Capítulo 6 descreve como foram realizados os testes e os resultados obtidos, qual estratégia foi mais eficiente e sua divulgação na plataforma de compartilhamento de códigos, para se difundir no meio acadêmico e industrial. Vale ressaltar que são realizados os testes com bases de larga escala, visto que há uma tendência no crescimento dos sistemas.

Por fim, no Capítulo 7, serão expostas as conclusões, dificuldades e trabalhos futuros.

2 SISTEMAS ELÉTRICOS DE POTÊNCIA

2.1 CONSIDERAÇÕES INICIAIS

Neste capitulo serão apresentados os conceitos de um sistema elétrico de potência, quais são seus principais componentes e como é realizado o controle da energia elétrica. Também serão citados quais estudos são realizados nestes sistemas, entrando em detalhes no estudo do fluxo de carga, mostrando sua importância e como é realizado.

Adicionalmente, será apresentado o histórico que une os principais trabalhos na interseção de computação paralela e fluxo de carga. Sendo assim, esse capitulo irá fornece as informações necessária para o estudo de caso desta dissertação, ou seja, o cálculo de fluxo de carga e os componentes de controle que o influenciam.

2.2 ESTRUTURA DE UM SISTEMA ELÉTRICO DE POTÊNCIA

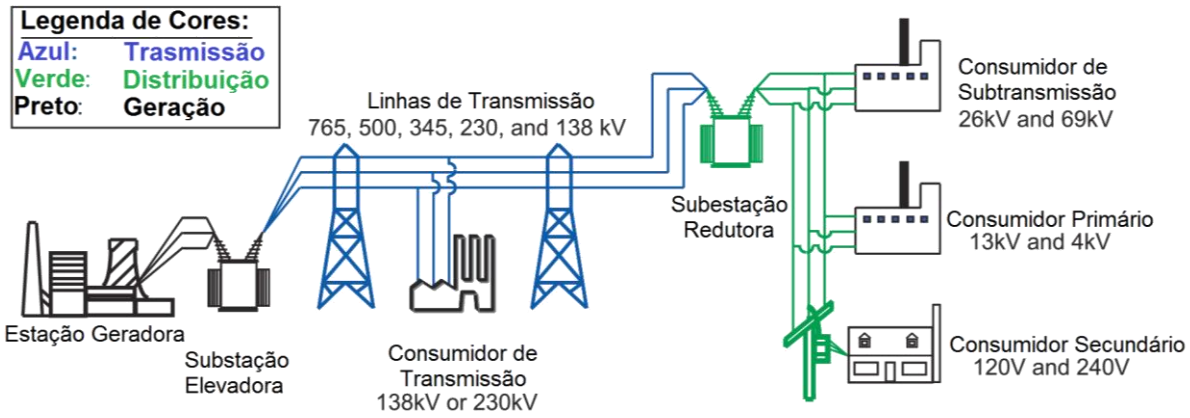
Um Sistema Elétricos de Potência (SEP) é uma rede de componentes interconectados, projetados para converter energia não elétrica contínua para a forma elétrica e transporta-la em distancias potencialmente grandes. Um SEP pode ser dividido em 03 (três) subsistemas: Geração, Transmissão e Distribuição (KUNDUR; BALU; LAUBY, 1994).

A Geração é responsável pela produção e injeção de energia elétrica dentro dos SEPs. De fato, os geradores são dispositivos capazes de converter energia não elétrica na forma elétrica. Tipicamente, a energia é produzida num gerador sob a forma de torque mecânico em um eixo rotativo, como por exemplo, estações hidroelétricas e eólicas. Porém, podem ser produzidas de outras formas, como por exemplo, através de estações térmicas, nucleares e fotovoltaicas.

O subsistema de Transmissão, atende ao requisito de transportar a energia elétrica das estações geradoras para as áreas consumidoras. Este transporte é realizado através das linhas de transmissão, o qual é necessário que seja feita em alta tensão, entre de 110-230 kV, ou extra-alta tensão, acima de 230 kV, para que possa haver menos perda de energia pelo efeito Joule (IEEE INDUSTRY APPLICATIONS SOCIETY. POWER SYSTEMS ENGINEERING COMMITTEE., 1994).

Por fim, o ultimo subsistema, o de Distribuição é responsável pela redução da tensão e fornecer a energia para os consumidores finais. A tensão é reduzida para Media Tensão, valores entre 1 kV e 110kV, afim de atender os consumidores indústrias como também é reduzida para Baixa Tensão, valores inferiores a 1 kV, atendendo a generalidade de consumidores domésticos (IEEE INDUSTRY APPLICATIONS SOCIETY. POWER SYSTEMS ENGINEERING COMMITTEE., 1994). A Figura 2.1 exibe um SEP identificando cada subsistema.

Figura 2.1 - Ilustração de um Sistema Elétrico de Potência. Fonte: Força Tarefa de Contingências de Sistemas Elétricos de Potência USA-Canadá.



A Figura 2.1 demonstra superficialmente o processo de geração, transmissão e distribuição da energia elétrica. Primeiramente, há uma estação geradora, o qual produzirá a energia elétrica, seguida de uma subestação elevadora, contendo os transformadores de energia para aumentar o valor de tensão. Enfim, essa energia é transportada pelas linhas de transmissão, que pode haver ou não consumidores nesta etapa, normalmente grandes fabrica e mineradoras. Ao chegar na área de consumo essa energia e passada novamente por uma subestação com transformadores, mas agora para reduzir o valor de tensão e fornecer a energia para as indústrias e residências.

A seguir será apresentado como um SEP é representado, a utilização de barras elétricas para conexão desses componentes, como são classificadas essas barras e quais

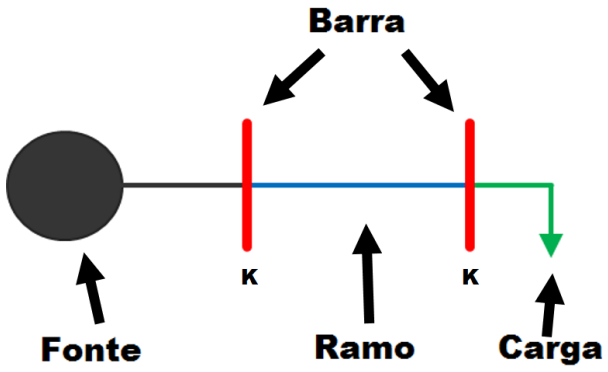
informações do SEP podem ser obtidas partir delas, para que assim possam ser realizados os estudos nestes sistemas.

2.3 REPRESENTAÇÃO DE UM SISTEMA ELÉTRICO DE POTÊNCIA

A maioria da energia transmitida pelo mundo é realizada via sistema trifásico, sendo composto por 03 (três) ondas senoidais, defasadas em 120 graus entre si, de forma a equilibrar o sistema. Embora a estrutura de um SEP siga um sistema trifásico, para simplificação e entendimento o mesmo, via de regra é apresentado utilizando um sistema de diagrama unifilar, onde os elementos elétricos podem ser exibidos por símbolos esquemáticos padronizados e em vez de representar cada fase com uma linha separada, apenas 01 (um) condutor é representado. A partir desse diagrama, pode-se construir um circuito equivalente por fase para então prever o desempenho no sistema trifásico (GROSS, 1986).

A Figura 2.2 demonstra um SEP simples representado por um diagrama unifilar. Percebe-se que existe uma fonte geradora de energia, uma linha de transmissão ou ramo e uma carga que representa um gasto de energia pelos consumidores. Também, pode-se notar, que entre estes componentes há barras elétricas realizando essa interconexão.

Figura 2.2 - Representação simples de Diagrama Unifilar.



As barras elétricas fornecem informação da energia em cada ponto de um SEP, podendo assim ser feita analises e estudos neste sistema. Cada barra k possui o valor de potência ativa P_k, potência reativa Q_k, magnitude da tensão nodal V_k e ângulo da tensão θ_k;

porém, dependendo das ligações existentes na barra, os valores de algumas variáveis serão conhecidos, enquanto outras terão que ser calculadas. A Tabela 2.1 apresenta os tipos de classificação das barras, de acordo com suas ligações, quais valores são conhecidos e quais são desconhecidos.

Tabela 2.1 Classificação das barras elétricas.

Tipos	**Variáveis conhecidas**	**Variáveis desconhecidas**	**Características**
PV	P_k, V_k	Q_k, θ_k	Barras de geração
PQ	P_k, Q_k	V_k, θ_k	Barras de carga
Referência (Vθ, slack, swing)	V_k, θ_k	P_k, Q_k	Barras de geração (geralmente uma unidade geradora de grande capacidade)

As barras do tipo PV possuem as informações da potência ativa P_k e magnitude da tensão nodal V_k, isso por que estão conectadas a fontes geradoras de energia. Já a de referência também conhecida pelos termos Vθ, Slack e Swing assim com PV, está ligada a uma fonte geradora, contudo em todo o SEP só há apenas 01 (uma) barra de referência, geralmente conectada a maior fonte geradora do sistema e por seu ângulo ser referência para as outras, são conhecidas a magnitude V_k e ângulo θ_k da tensão nodal. Por fim as demais barras que não estão ligadas diretamente a componentes produtores de energia, as PQ são conhecidas a potência ativa P_k e reativa Q_k (BORGES, 2005).

Na próxima subseção serão apresentados alguns equipamentos de controle dos SEP presentes no subsistema de transmissão, estes são responsáveis por otimizar o sistema realizando ajustes na tensão e potência transportada. Também, para um melhor entendimento do SEP e seu relacionamento com os equipamentos, será introduzido um exemplo de estudo de caso *benchmark* de um diagrama unifilar de um caso de teste de SEP.

2.4 EQUIPAMENTOS DE CONTROLE EM SISTEMAS ELÉTRICOS DE POTÊNCIA

Modernos sistemas de transmissão têm aumentado o número de dispositivos eletrônicos de potência com o objetivo de aumentar o controle da transmissão de energia. O conjunto desses equipamentos de controle, para corrente alternada, é chamado de sistema de transmissão flexível em corrente alternada ou como conhecida em inglês por *Flexible Alternating Current Transmission System* (FACTS). Os equipamentos podem ser conjugados com soluções tradicionais, como compensações fixas em série ou shunt, de forma a complementar o sistema. A seguir será apresentado alguns desses dispositivos.

2.4.1 Regulador Automático de Tensão

Os reguladores automáticos de tensão, conhecidos em inglês por *Automatic Voltage Regulator* (AVR), estão presentes em todos os geradores e compensadores síncronos do SEP. Estes dispositivos são compostos por diversos componentes como diodos, capacitores, resistores e potenciômetros até mesmo microcontroladores, tudo disposto em uma placa de circuito. Posicionados próximos dos geradores e conectados por um conjunto de fios para realizar as medições e ajuste do gerador. Os AVRs são utilizados para controlar a saída de energia da fonte geradora e manter constante o valor de tensão, porém um regulador não pode gerar energia, o valor de saída do equipamento sempre será menor ou igual ao de entrada.

2.4.2 Transformador com *Tap* Variável

Os transformadores são usados para aumentar ou diminuir as tensões alternadas em SEPs, a partir da relação de espiras presente no enrolamento do transformador. Esses componentes que possuem o mecanismo de variação de *tap*, possibilitam a variação da relação de espiras, tendo vários pontos de acesso ao longo do enrolamento. Em transformadores com o *tap* variável existem dois tipos primários de variação do *tap*. Primeiro, que é necessário o transformador ser desenergizado, chamados de *No Load Tap Changer* (NLTC). Segundo, que pode ser ajustado durante a operação, chamado de *On Load Tap Changer* (OLTC). Com os OLTCs é possível realizar o ajuste da tensão transformada ao

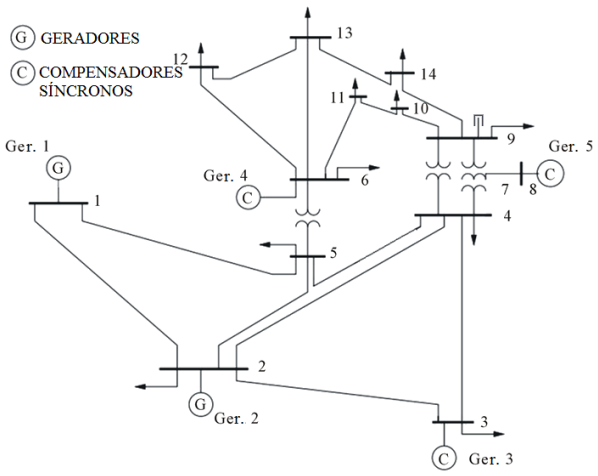
longo do SEP, mantendo os valores de tensão próximos dos valores nominais e então manter a confiabilidade do SEP.

2.4.3 Compensador Estático Fixo

No caso de linhas de transmissão muito longas, acima de 400 km, ocorre o Efeito Ferranti. Isto quer dizer, as linhas de transmissão têm um efeito capacitivo, que faz com que a tensão se eleve. Assim, sem uma compensação reativa, a tensão de regime no final da linha de transmissão é sempre maior do que no início. Os compensadores estáticos fixos, como por exemplo um reator shunt, *Shunt Reactor* (ShR), injetam uma potência reativa Q_k no SEP, já que Q_k é inversamente proporcional ao modulo da tensão V_k, ao aumentar Q_k diminui-se V_k.

A Figura 2.3 é um diagrama unifilar representando um caso de teste para benchmark do Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) com 14 barras elétricas. Percebe-se a presença de geradores e compensadores síncronos nas barras de número 1, 2, 3, 4 e 8, sendo a barra 1 do tipo de referência, pois está ligada a fonte de maior geração do SEP e as demais sendo do tipo PV, pois estão ligadas a fontes geradoras. As barras onde não são injetadas tensão no SEP são do tipo PQ. Nota-se também a presença de transformadores entre as barras 5-6, 4-9 e um transformador com 03 (três) enrolamentos entre as barras 4 ,8 e 9. Por fim, pode-se verificar a presença de um compensador estático fixo na barra 9.

Figura 2.3 Diagrama Unifilar de 14 barras para benchmark do IEEE. Fonte: (UNIVERSITY OF WASHINGTON, 2017)



Na seção 2.3, vimos as diferentes classificações de barras elétricas, quais variáveis são conhecidas em cada tipo e quais são necessários ser calculadas, na próxima seção será apresento o cálculo do fluxo de carga, o qual pode-se obter os outros valores desconhecidos das barras elétricas.

2.5 FLUXO DE CARGA

Fluxo de Carga (FC), chamado também de Fluxo de Potência é um estudo de SEP, realizado em estado estacionário do sistema, quando está operando em regime permanente, com o objetivo de determinar as características de operação estável dos sistemas de geração e transmissão de um SEP, dado um conjunto de barras elétricas de carga. A solução esperada são as informações, o ângulo e magnitude da tensão, assim como a potência ativa e reativa de cada unidade de transmissão, as perdas de potência durante a transmissão e a quantidade de energia reativa gerada/absorvida barra com tensão controlada.

O FC realiza uma modelagem do SEP, em forma de equações algébricas não lineares, mas antes de saber como é feita esta modelagem, é necessário ter uma maior compreensão sobre matrizes de admitância, o que será explicado na seção a seguir.

2.5.1 Matriz de Admitância

A matriz de admitância Y, representa quando o valor de $Y_{ij} \neq 0$, que a barra i possui uma conexão direta com a barra j e vice-versa, seja por linha de transmissão ou por transformador. A admitância y é obtida pelo inverso da impedância Z exibido pela equação (2.1). A partir das informações de resistência r e reatância x das linhas de transmissão, tem-se o valor de impedância Z, logo, o valor de y. A equação (2.2), mostra a relação entre a corrente complexa I^* e tensão complexa E, onde E pode ser representado pela magnitude da tensão V e ângulo da tensão θ. A Figura 2.4 demonstra a equação (2.2) em forma de matriz para um SEP com n barras elétricas.

$$y = Z^{-1} \mid Z = r + jx \tag{2.1}$$

$$I_k{}^* = \sum_{i=0}^{n} Y_{ki} * E_i \mid E_i = V_i \angle \theta_i \tag{2.2}$$

Figura 2.4 - Matriz de Admitância.

$$\begin{bmatrix} I_1{}^* \\ I_2{}^* \\ \vdots \\ I_n{}^* \end{bmatrix} = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1n} \\ Y_{21} & Y_{21} & \cdots & Y_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ Y_{n1} & Y_{n2} & \cdots & Y_{nn} \end{bmatrix} * \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

O cálculo de cada elemento da matriz de admitância é realizada conforme a equação (2.3), onde os elementos cuja os valores de i e j são diferentes, utiliza apenas o valor negativo da admitância entre a barra i e j, no outro caso, realiza-se uma somatória das admitâncias de todas as linhas de transmissão que terminam na barra i.

$$Y_{ij} = \begin{cases} i = j, & \sum_{k=0 \neq i}^{n} y_{ik} \\ i \neq j, & -y_{ij} \end{cases} \tag{2.3}$$

Visto como é feita o preenchimento da matriz de admitância e sua relação com a corrente e tensão, na próxima seção será apresenta a modelagem do SEP para equações algébricas e quais os principais métodos computacionais de resolução.

2.5.2 Modelagem do Fluxo de Carga

Como dito no Capitulo 2.3, existem diferentes tipos de barra e o fluxo de carga irá fornecer as informações a priori desconhecidas dessas barras. A equação para o cálculo das potencias pode ser visto em (2.4) onde cada barra k possui uma potência S_k que pode ser decomposta em potência ativa P_k e reativa Q_k obtendo esse resultado pela multiplicação da tensão E_k e corrente $I_k{}^*$. Fazendo a substituição do $I_k{}^*$ da equação (2.2) em (2.4),depois desenvolvendo o resultado e separando a parte real da imaginaria, adquire-se as equações

(2.5) e (2.6), onde a parte real é a potência ativa P_k e parte imaginaria é a potência reativa Q_k (GROSS, 1986).

$$S_k = P_k * jQ_k = E_k * I_k^* \tag{2.4}$$

$$P_k = \sum_{m=0}^{n} V_k V_m \left(G_{km} \cos\theta_{km} + B_{km} \sin\theta_{km}\right) \tag{2.5}$$

$$Q_k = \sum_{m=0}^{n} V_k V_m \left(G_{km} \sin\theta_{km} - B_{km} \cos\theta_{km}\right) \tag{2.6}$$

Devido à variedade de tipos de barras essas equações (2.5) e (2.6) compõem 02 (dois) subsistemas.

2.5.2.1 Subsistema 1

Este subsistema contém as equações que devem ser resolvidas para se encontrar o módulo e ângulo das tensões nas barras, para isso estipula-se os valores de V_k e θ_k para as barras do tipo PQ e o valor de θ_k para barras do tipo PV que se aproximem de seus conhecidos valores de P_k e Q_k de acordo com as equações deste subsistema.

$$\begin{cases} P_k = P_{k(V,\theta)}, & k \in \{PV, PQ\} \\ Q_k = Q_{k(V,\theta)}, & k \in \{PQ\} \end{cases} \tag{2.7}$$

2.5.2.2 Subsistema 2

Este subsistema contém as equações que devem ser resolvidas para se encontrar a potência ativa e reativa nas barras, para isso utiliza-se os valores obtidos no Subsistema 1.

$$\begin{cases} P_k = P_{k(V,\theta)}, & k \in \{V\theta\} \\ Q_k = Q_{k(V,\theta)}, & k \in \{V\theta, PV\} \end{cases} \tag{2.8}$$

Essa aproximação realizada dos valores de potência a partir do modulo e ângulo da tensão do subsistema 1, pode ser feita de forma analítica, porém para casos de SEPs muito grandes, esta forma se torna impraticável, como também pode ser feita de forma computacional, para isto tem-se algumas técnicas como Gauss-Seidel, Newton-Raphson e Desacoplado rápido (BORGES, 2005).

O método de Gauss-Seidel (GS) é o mais antigo para a solução do fluxo de carga. É simples, confiável e geralmente tolerante a condições de baixas tensões e potência reativa. Apesar de ter um baixo consumo de memória, seu tempo de convergência tem crescimento rápido com relação ao aumento do tamanho do SEP. Este método possui uma lenta taxa de convergência e apresenta um problema de convergência para sistemas com altos níveis de potência ativa transferida.

O método de Newton-Raphson (NR) tem uma boa taxa de convergência. Seu tempo computacional tem um crescimento apenas linear ao tamanho do SEP. Este método, tem problemas de convergência para o caso de valores inicias serem estipulados muito diferentes dos valores reais, por isto não está apto a inicialização de tensão "*flat*", termo usado quando os valores inicias são atribuídos 1 (um) para módulo e 0 (zero) para o angulo da tensão. Além de ser necessário o cálculo da matriz jacobiana em cada iteração. O NR é adequado a sistemas de larga escala, o qual necessitam de soluções precisas.

O método Desacoplado Rápido, também conhecido em inglês por *Fast Decoupled Power Flow* (FDPF), é uma variação do método NR, não sendo necessário ser recalculado e refatorada a matriz jacobiana em cada iteração, assim diminui o tempo de processamento, porém aumento o número de iterações necessárias para obter a solução. Este método é menos sensível as condições de inicialização da tensão. Possui versões XB e BX, onde a primeira é negligenciado o efeito de resistência em série, por isso não é indicado para sistemas com altas taxas de resistência e reatância nas linhas de transmissão, sendo mais indicado para esses sistemas a segunda versão. O FDPF prover uma solução rápida e precisa para a maioria das condições de sistemas. Contudo, para sistemas como altos valores de ângulo de tensão pelas linhas de transmissão e com a presença de equipamentos de controle que influencia na potência ativa e reativa do SEP a utilização do NR pode ser necessária (KUNDUR; BALU; LAUBY, 1994).

Para este trabalho será utilizado o Método de NR, devido a sua adaptação a generalidade de condições de SEPs, tanto de sistemas pequenos e de grande escala, quanto a presença de equipamentos de controle, além de outros motivos que serão apresentados logo mais à frente no Capitulo 4, onde será exposto os trabalhos correlatos. A seguir será apresentado o método de NR para o problema de fluxo de carga.

2.5.3 Solução do Fluxo de Carga pelo Método de Newton-Raphson

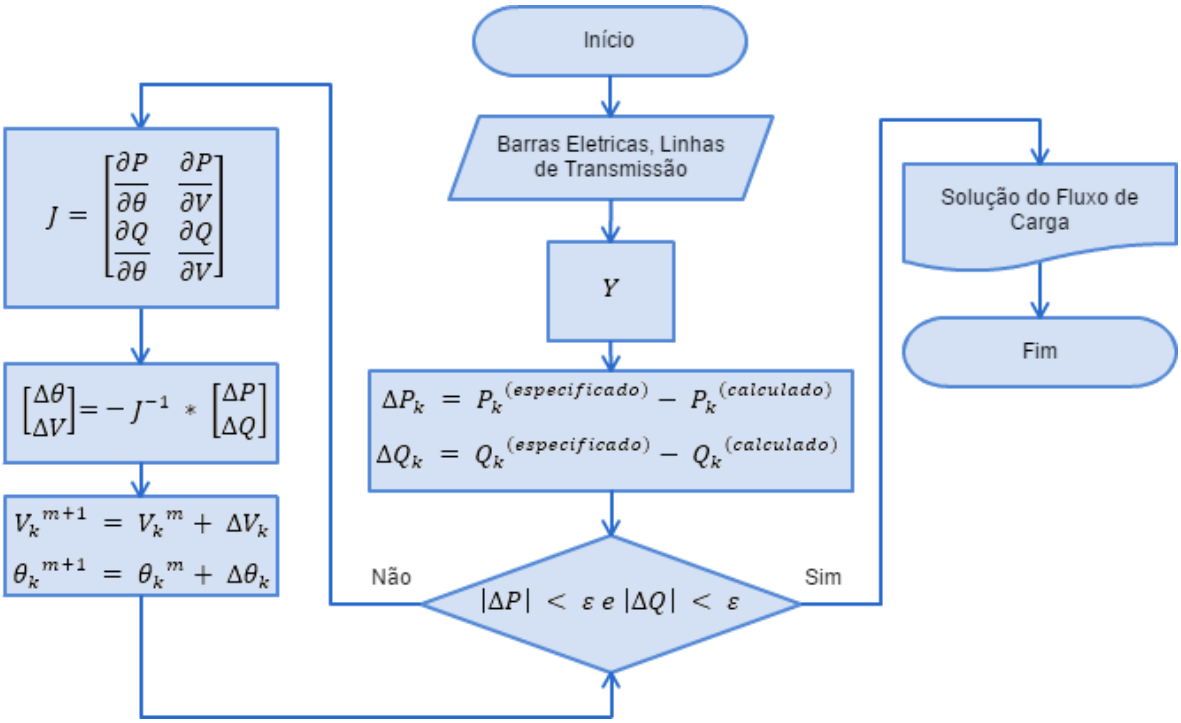
O método conforme mencionando na seção anterior, é um método mais genérico o qual aceita diversas condições do SEP. Um dos seus principais problemas está na inicialização dos valores de modulo e ângulo de tensão, para isso, usufrui-se de um seguinte princípio. Para o fluxo de carga, o sistema tem que estar em regime permanente, caso haja alguma alteração no SEP, seja nas cargas ou nos equipamentos de controle, o cálculo do FC tem que ser realizado novamente. Então, em vez de estipular os valores inicias completamente aleatórios ou pela inicialização de tensão "*flat*", e dado um direcional para esses valores, apropriando-se dos valores obtidos no cálculo do fluxo de carga anterior, os quais não vão esta corretos, mais estão mais próximos dos valores reais neste novo estado de regime permanente (KUNDUR; BALU; LAUBY, 1994).

Em um SEP novo, seja gerado do zero ou apenas realizando uma modificação em um sistema antigo, como por exemplo adicionar uma nova fonte geradora, neste caso não se tem os dados de um cálculo anterior, então é utilizado um outro método como GS ou FDPF para o 1º cálculo do FC e futuramente utilizar as informações anteriores no NR (KUNDUR; BALU; LAUBY, 1994).

A Figura 2.5 exibe o fluxograma de execução do método NR. Primeiramente é realizada a leitura das informações de todas as barras elétricas e linhas de transmissão do SEP. Logo mais é realizado o cálculo da matriz de admitância, conforme mostrada na seção 2.5.1. Então, utilizando as equações (2.5) e (2.6) para o subsistema 1 apresentado na seção 2.5.2.1, calcula-se os valores de potência ativa ${P_k}^{calculado}$ e reativa ${Q_k}^{calculado}$ e comparados os valores dessas potências já conhecidas ${P_k}^{especificado}$ e ${Q_k}^{especificado}$, obtendo assim os valores de desajustes das potências ΔP_k e ΔQ_k.

Com os valores de desajustes, verifica se o os critérios de parada foram satisfeitos, para isso, caso os valores máximo absoluto de desajuste da potência ativa $|\Delta P|$ e reativa $|\Delta Q|$ sejam menores que um valor de erro ε previamente definido, considera como o método convergido, inicia-se o processo de uma atualização dos valores de módulo e ângulo da tensão com base nesse desajuste das potências.

Figura 2.5 - Fluxograma do Newton-Raphson para Solução do Fluxo de Carga.



A matriz jacobiana é composta por quadrantes, onde em cada quadrante, tem-se uma derivada da potência em função de variável de modulo ou ângulo da tensão. Depois do cálculo da jacobiana, inverte-se a matriz para encontrar a variação de módulo ΔV e ângulo $\Delta\theta$ da tensão. A inversão pode ser realizada por uma resolução de sistemas lineares. Com as variações das variáveis da tensão, atualiza-se os valores com uma soma do valor atual mais a variação.

Com os novos valores de tensão, calcula-se novamente os desajustes de potência e verifica-se também o critério de parada, caso ainda não satisfeito, repete-se o processo até que se seja satisfeito. É comum também utilizar um outro critério de parada para não deixar executando muito tempo o método, que é o número máximo de iterações, os desajustes não satisfaçam o erro, mas quando o número de iterações máxima é atingida este método é

finalizado com os valores de tensão incorretos para o erro predefinido. Quando os critérios de parada são satisfeitos, o NR é finalizado com uma solução precisa das informações do SEP.

Como o FC provê somente as informações básicas de um SEP, e só pode ser realizado em estado estacionário, sendo necessário a realização do cálculo novamente quando houver qualquer alteração no sistema, buscou-se formas para que seu tempo de processamento fosse reduzido, para isso iniciaram estudos com o intuito de paralelizar os métodos de FC e então diminuir seu tempo. Na próxima seção será apresentado um breve histórico do paralelismo destes cálculos.

2.6 PARALELISMO DO FLUXO DE CARGA

Com o surgimento da computação paralela como uma ferramenta de otimização, vislumbrou-se a oportunidade de aplicar esta abordagem no cálculo de FC, a fim de reduzir o custo em SEPs mais complexos. De acordo com (J. TYLAVSKY; BOSE, 1992), a motivação para a utilização de paralelismo neste problema está simplesmente no desejo de tornar a resolução do problema mais rápida, pois não há paralelismo inerente na estrutura matemática do FC, exceto para alguns dos métodos utilizados em sua solução. Neste âmbito, nas décadas de 1970 e 1980, iniciaram-se os desenvolvimentos de trabalhos com o intuito de se aproveitar da computação paralela para o cálculo do FC (HOUSOS; OMAR WING, 1979; RAFIAN; STERLING; IRVING, 1985; WANG et al., 1989). Tais trabalhos, em sua maioria, criavam algoritmos paralelos, porém não os testavam em arquiteturas paralelas, apenas teorizavam seus *speedups*, métrica de avalição de paralelização.

Na década de 1990, a abordagem mais comum no design de métodos paralelos para o cálculo do FC era a paralelização das tarefas e estruturas contidas nos métodos (sequenciais) bem consolidados na área (HUANG; ABUR, 1990; LAU; TYLAVSKY; BOSE, 1991) – como GS (HUANG; ONGSAKUL, 1994), NR (AMANO; ZECEVIC; SILJAK, 1996; JUN QIANG WU; BOSE, 1995) e FDLF (EL-KEIB; DING; MARATUKULAM, 1994), embora alguns métodos específicos para ambientes paralelos também tenham sido desenvolvidos (CHEN; BERRY, 1993; WANG; HADJSAID; SABONNADIÈRE, 1999). Esta abordagem facilita no designe e implementação paralela de

tais algoritmos, pois possibilita a reutilização de trabalhos e bibliotecas que já realizaram a paralelização das tarefas e estruturas necessárias para estes métodos.

Enquanto isso, esforços para fortalecer a área de computação paralela levaram a definições e padrões em 1992, no *Workshop on Standards for Message Passing in a Distributed Memory Environment* (BARNEY, 2016), com a direção para arquiteturas paralelas mais acessíveis para diversos problemas, inclusive no FC, como clusters de computadores, ao invés de supercomputadores (BALDICK et al., 1999; LI; BROADWATER, 2003).

Como resultado da busca por ambientes de baixo custo, no início do século XXI, as *Graphic Processing Units* (GPU) começaram a ser utilizadas como alternativa de ambiente paralelo, dada seu desempenho em operações de ponto flutuante e o baixo custo para adquirir o hardware. Aliado a isso, a busca por padronização da programação paralela motivou o desenvolvimento de modelos de programação paralela para GPU. Como um novo modelo para a programação em GPU, a plataforma *Computer Unified Device Architecture* (CUDA) (NVIDIA CORPORATION, 2016a) desenvolvida pela NVIDIA; ainda hoje em dia como uma das mais utilizadas em ambientes de computação em GPU. Tal plataforma passou a ser utilizada, também, na paralelização de métodos de FC (LI; LI, 2014; MEI YANG et al., 2012), apresentando bom desempenho quando aplicada a SEPs de larga escala.

2.7 CONSIDERAÇÕES FINAIS

Neste capitulo, vimos o que é um sistema elétrico de potência, quais seus componentes, equipamentos de controle e como representá-los. Também, foi apresentado o cálculo de fluxo de carga e seus diferentes métodos computacionais de solução, em mais detalhes, o método de Newton-Raphson, o que suporta a maioria de sistemas com diferentes condições. Adicionalmente, relatou-se um pequeno contexto histórico sobre o paralelismo do fluxo de carga, apontando diversos ambientes de computação paralela utilizadas e a mais recente, computação em GPU. No próximo capitulo, se mostra em mais detalhes a plataforma CUDA desenvolvida pela NVIDIA, para computação em placas de vídeo.

3 COMPUTAÇÃO PARALELA EM GPU

3.1 CONSIDERAÇÕES INICIAIS

Neste capitulo será apresentado um pouco sobre o contexto histórico do processamento em GPU para propósitos gerais. Adicionalmente, será apresentado a plataforma CUDA que até o presente momento é uma das mais utilizadas nos trabalhos de paralização em GPU, por esta razão, será usada nesta dissertação. Complementarmente será exposto as principais definições, o fluxo de execução e arquitetura das placas gráficas com suporte a esta plataforma.

Diante no exposto neste capitulo, será possível ter uma compreensão maior da paralelização em GPU, o que contribuirá para o desenvolvimento das estratégias de paralelização na plataforma CUDA do Fluxo de Carga para múltiplas execuções e melhorar a eficiência do processamento, reduzindo o tempo necessário.

3.2 CONTEXTO HISTÓRICO

No surgimento das placas de vídeo, seu principal objetivo era a realização de aceleração gráfica, permitindo apenas pipelines de função fixa específica. No final da década de 90, o hardware passou a ser mais programável, despertando o interesse não apenas de artistas e desenvolvedores de jogos, mas também, de pesquisadores, buscando aproveitar a desempenho de ponto flutuante desses periféricos. Assim, deu-se início ao termo de GPU de Propósitos Gerais (GPGPU – *General Purpose Graphics Processing Unit*). Entretanto, nesta época, a GPGPU não era simples de ser programável para propósitos gerais, pois, era necessário que os cálculos científicos fossem mapeados, e representados na forma de triângulos e polígonos.

Um grupo de pesquisadores da universidade de Stanford, liderados por Ian Buck, se reuniu para reformular a GPU como um *Streaming Processor*. Resultando em 2003, na anunciação da linguagem de programação Brook, um modelo de programação em linguagem de alto nível para GPGPU. Onde além de sua codificação ser bem mais simples, também era 7x mais rápido do que códigos similares existentes naquele período.

Em 2004 a NVIDIA, umas das principais empresas em computação gráfica, sabendo do potencial das placas gráficas, convidou Ian Buck a se juntar a sua equipe e começar a desenvolver uma solução para executar a linguagem de programação C na GPU de forma fluida, ocasionando em 2006 no anuncio da plataforma CUDA, a primeira solução do mundo para computação de propósito geral em GPUs e uma das mais utilizada até os dias de hoje em trabalhos de paralelismo com GPU (NVIDIA CORPORATION, 2016a).

A seguir será apresentado com mais detalhas a plataforma CUDA, as principais definições para o compreendimento do fluxo de execução e arquitetura das GPUs com suporte a CUDA.

3.3 PLATAFORMA CUDA

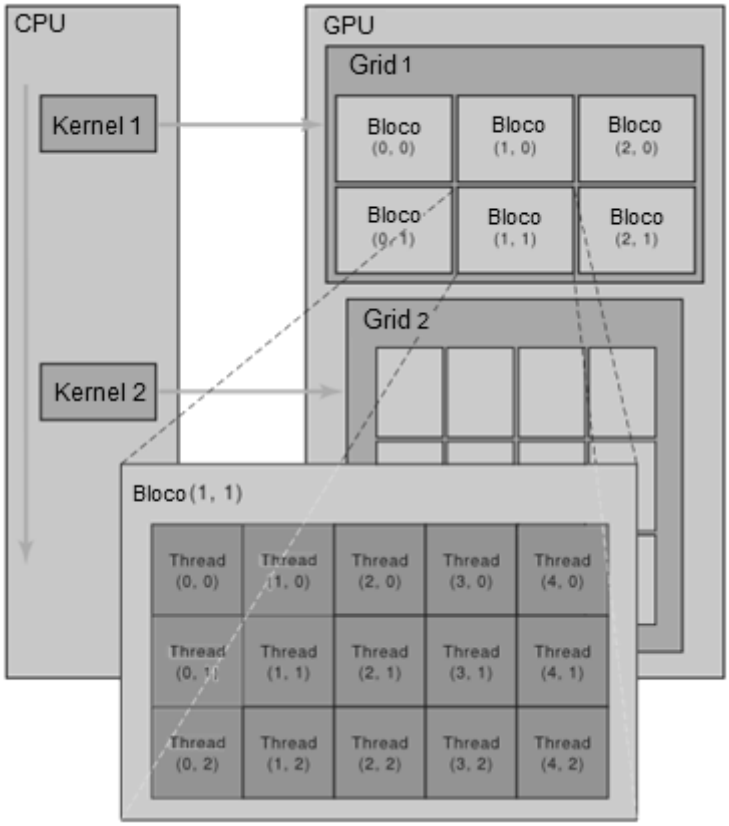
A CUDA oferece suporte para linguagens de alto nível como Fortran, Java, Python, C#, MATLAB porém seu principal kit de desenvolvimento utiliza C/C++ como linguagem de programação. A plataforma vem sendo explorada por desenvolvedoras de grandes empresas como Adobe, ANSYS, Autodesk, MathWorks e Wolfram Research para executar computação científica de propósito geral e de engenharia em uma ampla variedade de softwares (NVIDIA CORPORATION, 2016a). A seguir será apresentado algumas definições utilizada pela plataforma.

3.3.1 Principais definições

No desenvolvimento em CUDA, a CPU e a memória principal do computador são chamadas *host*, enquanto a GPU é chamada *device*, em um mesmo programa pode ser colocado instruções a serem executados por ambos. Entretanto, a execução deste programa começa no *host*, sendo necessário que se faça uma chamada para a execução no *device*. Esta chamada é feita por uma função especial, denominada *kernel*.

Quando é feita uma chamada para a execução de um *kernel*, constrói-se uma estrutura chamada *grid*. Esta estrutura é formada pelo conjunto de *threads* que executarão o código contido no *kernel* chamado. Estas, por sua vez, são separadas em blocos de *threads* com números iguais de *threads*. A Figura 3.1 demonstra o processo acima descrito de execução de instrução em GPU.

Figura 3.1 – Ilustração de chamada a execução de instrução em GPU. Fonte (NVIDIA CORPORATION, 2016b)



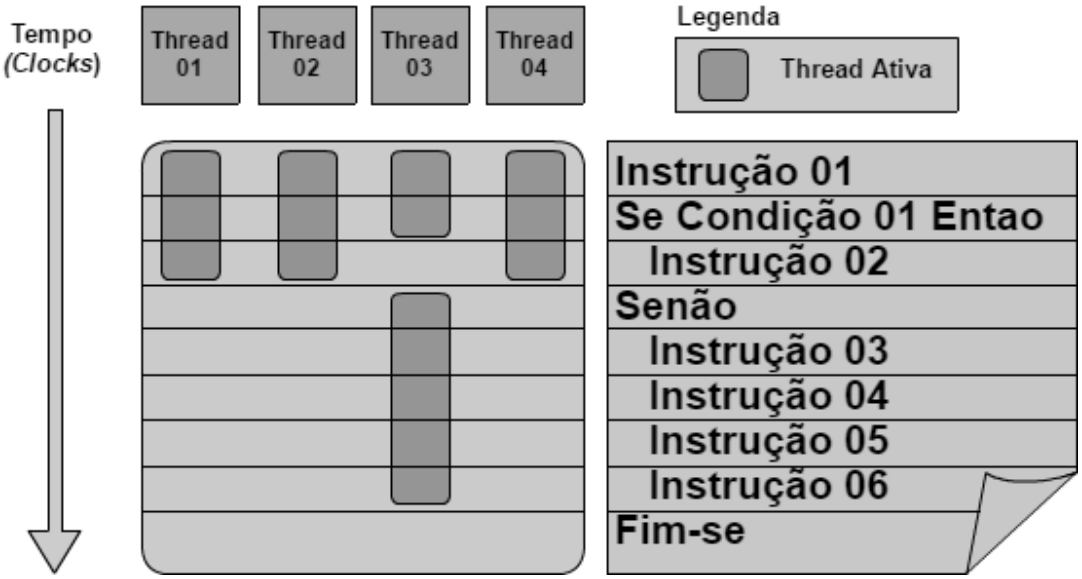
Percebe-se que na Figura 3.1 que é realizado dois *kernels* e cada um tem seu próprio *grid* de execução. No caso do primeiro *grid* são construídos 6 blocos de *thread*, onde cada um possui 15 *threads*, totalizando a execução de 90 *threads* para o primeiro *kernel*. Vale ressaltar que tanto o *grid* quanto o bloco de *threads* estão dispostos em duas dimensões, podendo ser em até três. Também é importante mencionar que a distribuição do número de *threads* por blocos, quantidades de blocos e suas dimensões, que totalizam o número de *threads* a ser executada pelo *kernel* é definida pelo desenvolvedor durante a chamada do *kernel*.

O modelo de execução em computação paralela pela taxonomia de Flynn (MONTEIRO, 2007) adotado na plataforma CUDA é *Single Instruction, Multiple Data* (SIMD), onde uma mesma instrução e realizada com múltiplos dados concorrentemente. No

caso das GPUs com suporte a essa plataforma são executadas 32 *threads* simultaneamente, esse conjunto é denominado *warp.*

O conhecimento de *warps* é importante para prevenir dois problemas que podem diminuir o desempenho, são esses: a ociosidade de *threads* e a inconsistência de *warps*. O primeiro, se refere a não utilização de um número de *threads* suficientes para completar um *warp*, isto quer dizer que terão threads do *warp* ociosas, indicando que não está sendo aproveitado o máximo que a GPU pode oferecer. O segundo, é quando existem instruções condicionais a ser executa em que uma condição é realizada por parte das *threads* de um *warp* e a outra condição pelas *threads* restantes. A Figura 3.2 demonstra um exemplo de inconsistência de *warp*.

Figura 3.2 – Exemplo de inconsistência de warp



A Figura 3.2 mostra apenas 4 *threads*, por meio de sua análise é possível perceber que por causa da *thread* 03 houve muita ociosidade das outras *threads*. Pelo fato de todas as *threads* de um *warp* executarem baseados em SIMD, essas threads terminaram o processamento no mesmo instante de tempo. Por esta razão, deve-se tentar manipular o desenvolvimento para que as condições sejam executadas em mesmo *warp* ao máximo.

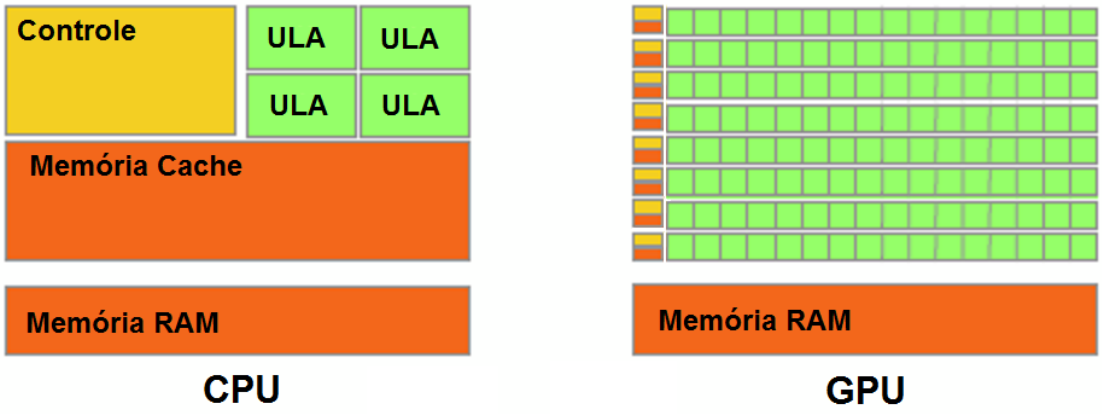
Na próxima seção será apresentada a arquitetura das placas de vídeo com suporte a plataforma, mostrando as principais diferenças com a CPU e citando pontos onde pode ser obtido um melhor aproveitamento do *hardware*.

3.3.2 Arquitetura do Hardware

A filosofia de *design* das GPUs, foi moldada pelo rápido crescimento das indústrias de jogos eletrônicos, que exerceram uma grande pressão econômica, pela capacidade de realizar massivos cálculos de números de ponto flutuante, por *frame* de vídeo nos jogos mais avançados. Esta demanda motivou as fabricantes de placas gráficas a aumentarem seus esforços para criar uma nova arquitetura do hardware, reduzindo o consumo de energia e maximizando a área dos chips para cálculos de pontos flutuantes, dessa forma otimizando a taxa de transferência em execução de grandes quantidades de *threads* (KIRK; HWU, 2013).

Por outro lado, as CPUs foram projetadas para diminuir a latência de uma única *thread*, com uma grande área do chip voltada para memória cache que armazena os dados mais frequentes afim de reduzir o tempo de acesso a essa informação. A Figura 3.3 ilustra uma representação das duas arquiteturas.

Figura 3.3 – Modelo de projeto das arquiteturas da CPUs e GPUs. Fonte: (NVIDIA CORPORATION, 2016b)

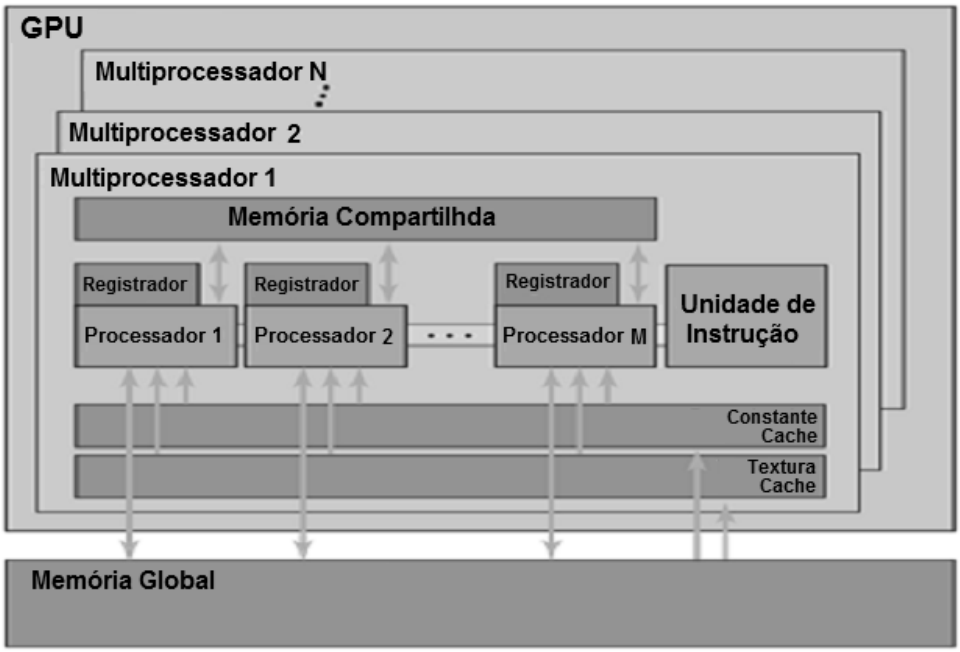


Pode-se observar na Figura 3.3 que a CPU possui uma sofisticada Unidade de Controle (UC) e de Unidades Logicas Aritméticas (ULAs), além de reservar boa parte de seu chip para memória cache. Enquanto a GPU diminui as ULA, tornando-as mais simples e reduz o tamanho da UC e memória cache, com a finalidade de aumentar o número de ULAs.

As GPUs com suporte a CUDA são compostas por um conjunto de N Multiprocessadores (MPs), onde cada MP possui um conjunto de M processadores chamados

de núcleos CUDA. A Figura 3.4 apresenta a arquitetura do hardware das GPU e sua organização quanto aos MPs e dos núcleos CUDA.

Figura 3.4 - Modelo do hardware e sua hierarquia de memória. Fonte: (NVIDIA CORPORATION, 2016b)



Na Figura 3.4 percebe-se a hierarquia de memória da placa gráfica constituída por 3 níveis: memória global, memória interna ao MP e memória interna ao núcleo CUDA. A memória global é acessível por todos os MPs; memórias internas ao MP que são manipuláveis por cada núcleo CUDA de um mesmo MP e são divididas em 2 tipos, a memória compartilhada e as memorias de cache (de variável constante e texturas) que armazenam cópias do que há na memória global. A memória interna ao núcleo CUDA são os registradores, controlado apenas por seus próprios núcleos.

As Memórias Globais são consideradas de longa latência, cerca de centenas de ciclos de *clocks* são necessários para seu acesso. Contudo, a transferência de dados da memória da CPU com a GPU só é realizada por este nível de memória. Em vista disso, uma boa gerencia da utilização dos níveis de memória é fundamental para um maior benefício da paralização neste hardware, já os outros níveis de memórias são de baixa latência por serem memórias internas. Sendo assim, uma boa prática para reduzir esse impacto da latência da memória

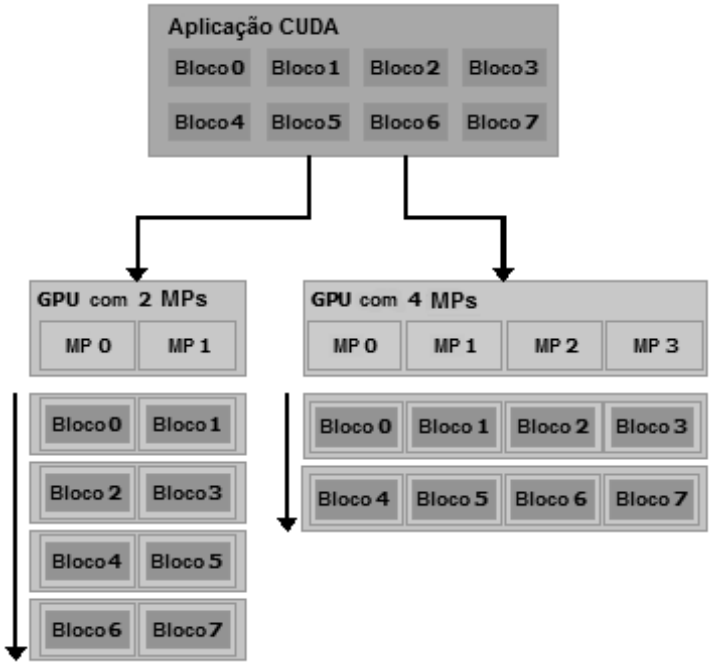
global é realizar uma cópia do dado em uma das memórias internas, caso for utilizar esse dado mais de uma vez (KIRK; HWU, 2013).

Esse modelo de hardware baseado em MP e núcleos CUDA, contribui para uma escalabilidade automática da plataforma, permitindo que uma aplicação desta plataforma seja executada em diferentes placas de vídeo. Na seção a seguir será explicado com mais detalhe a escalabilidade proporcionada pela plataforma.

3.3.3 Escalabilidade da Plataforma

Os limites de *threads* por blocos e blocos por MP da placa de vídeo possibilita a otimização da aplicação para a determinada GPU, fazendo o balanceamento de número de *threads* por blocos e quantidade de blocos do *kernel*. Todavia essas configurações do *kernel* podem ser generalizadas afim de tornar a aplicação executável em qualquer outro dispositivo periférico gráfico. Isto é possível devido ao modelo de hardware e software adotados pela plataforma, número de MPs e núcleos CUDA, número de blocos e quantidade de *threads* por bloco. A Figura 3.5 exibe um exemplo dessa escalabilidade onde uma aplicação CUDA é executada por 2 GPU com diferentes propriedades.

Figura 3.5 - Escalabilidade da plataforma CUDA. Fonte: (NVIDIA CORPORATION, 2016b)



A aplicação da Figura 3.5, o *kernel* foi configurado para a execução de 8 blocos e testada em duas placas gráficas com diferentes quantidades de MP, mas com o mesmo limite de 1 bloco por MP. Na primeira são necessários 4 ciclos de execução de blocos, já que são agrupados de forma a ocupar os 2 MP disponíveis. Enquanto na segunda, por ter uma maior quantidade de MP, o dobro da primeira, a quantidade de ciclos de execução de blocos e reduzida pela metade.

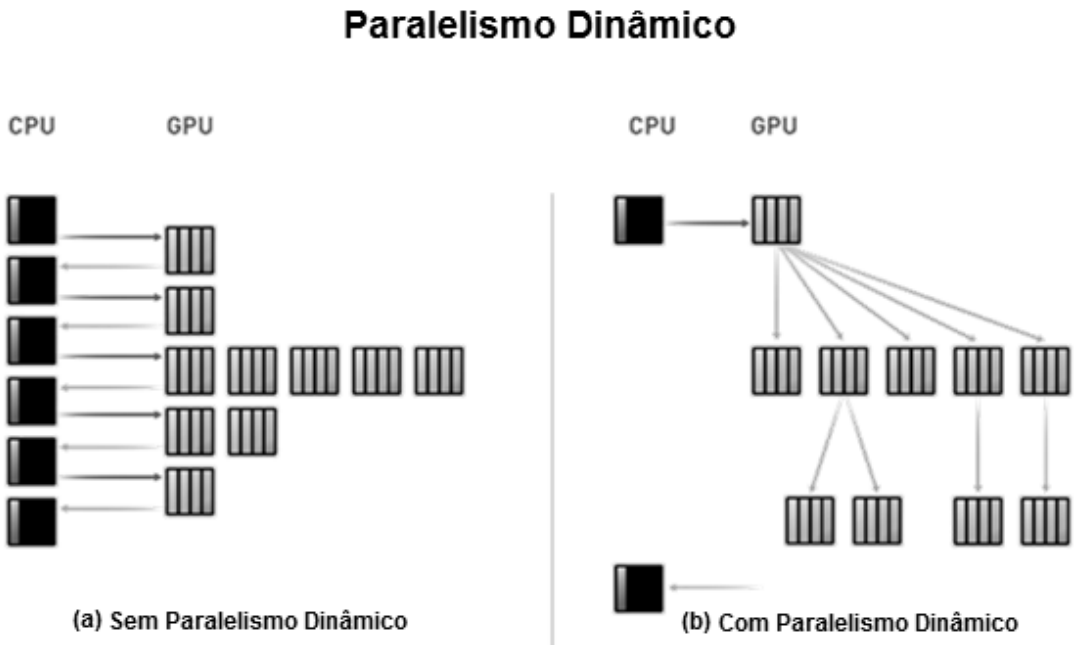
Quando um *kernel*, não é suficiente para ocupar todos os MPs da GPU, para ter benefício máximo da GPU, pode-se executar múltiplos *kernels* concorrentemente, mas só pode ser feito em placa gráficas com arquitetura a partir da geração Kepler. Além disso, essa geração proporciona também a utilização de paralelismo dinâmico CUDA, uma outra forma de tentar aproveitar ao máximo a GPU, reduzindo a troca de informação entre CPU e GPU. Na próxima seção será apresentado com mais detalhes o paralelismo dinâmico CUDA.

3.3.4 Paralelismo Dinâmico CUDA

O paralelismo dinâmico em CUDA, *CUDA Dynamic Parallelism* (CDP), foi incluído a partir da arquitetura Kepler e é uma extensão ao modelo de programação CUDA, possibilitando a chamada de *kernels* a partir do *device* e não mais apenas do *host*, resultando em uma redução da necessidade de sincronismo com a CPU (NVIDIA CORPORATION, 2016b).

Com essa nova inclusão na plataforma, houve uma melhora da desempenho em algoritmos executados em GPU que utilizam recursão, estruturas irregulares de repetição, variação na memória consumida durante o tempo de execução. (WANG; YALAMANCHILI, 2014) realizaram um estudo sobre a utilização de CDP em GPU avaliando como métrica o potencial benefício de desempenho e no comportamento do fluxo de controle e de acesso a memória em um conjunto de aplicações de benchmark não estruturadas. Os testes mostraram que a utilização de CDP pode ser 1,13x-2,73x mais rápido do que sem a utilização, porém a sobrecarga do grande número de lançamentos de *kernels* pode negar esse benefício. A Figura 3.6 exibe a mudança de padrão de chamada de novos *kernels* com a inclusão do paralelismo dinâmico CUDA.

Figura 3.6 - Paralelismo Dinâmico CUDA. Fonte: (NVIDIA, 2012)



Percebe-se que na Figura 3.6(a) sem a presença do CDP, apenas a CPU pode realizar a chamada de *kernels*. Em contrapartida na Figura 3.6(b) a presença do CDP permite que *threads* em GPU façam a chamada de novos *kernels*. Além disso, é notório a diminuição da comunicação entre CPU e GPU.

3.4 MÉTRICAS DE AVALIAÇÃO

Em arquitetura de computador, o *speedup* é a métrica de desempenho entre dois sistemas que processam o mesmo trabalho, representando o ganho em velocidade de execução da determinada tarefa em duas arquiteturas similares com diferente arquitetura. O termo *speedup* foi definido por lei de Amdahl, a qual foca particularmente o processamento paralelo(HENNESSY; PATTERSON; ASANOVIĆ, 2012). Esta métrica pode ser obtida pela Equação

$$Speedup = \frac{T_s}{T_n} \qquad (3.1)$$

Onde o T_s é o tempo de execução da versão sequencial e o T_n é o tempo de execução da versão paralelizada por n processos/processadores. O resultado varia entre 0 e n, tendo

um desempenho positivo para a versão paralela para valores acime de 1 (um). Caso o resultado seja inferior a 1 (um), o algoritmo paralelo é dito com menor eficiente do que sua versão sequencial, motivo pelo o qual dever ser analisado possíveis gargalos ou má utilização dos recursos disponíveis.

3.5 CONSIDERAÇÕES FINAIS

Neste capitulo, foi apresentado a plataforma CUDA, uma da mais utilizadas e bem-conceituada plataforma para paralelismo em GPU, fato este que contribuiu para sua utilização nesta dissertação. Também, foram expostos alguns conhecimentos técnicos a respeito da arquitetura do hardware utilizado de forma a tirar vantagem ao máximo da placa gráfica. Por fim, foi mostrado um novo recurso disponíveis nas arquiteturas mais atuais, o paralelismo dinâmico CUDA, o que o torna ideal para múltiplas execuções simultâneas afim de usufruir de todos os MP da GPU e reduzir o sincronismo com a CPU.

O conhecimento a respeito da plataforma e da arquitetura do hardware, irão contribuir para o desenvolvimento das estratégias de paralelismo em GPU de múltiplos cálculos do Fluxo de Carga, mas ainda, é necessário obter informações de trabalhos acadêmicos similares, ou de interseção ao assunto de forma a refinar a base das estratégias que serão desenvolvidas. Tendo isso em vista, no próximo capitulo, serão apresentados os trabalhos correlatos do assunto deste trabalho.

4 TRABALHOS CORRELATOS

4.1 CONSIDERAÇÕES INICIAIS

Neste capítulo serão apresentados alguns trabalhos que fazem uso do Fluxo de Carga, também que realizam a paralelização, seja por cluster ou *multithread*, e aqueles que utilizaram apenas a plataforma CUDA para paralelizar o problema.

O conhecimento aprendido nesse capítulo contribuirá para o desenvolvimento das estratégias para múltiplas execuções simultâneas do Fluxo de Carga, como também colaborará para a aquisição de uma resposta rápida do processamento exigido. Na próxima seção serão apresentados alguns trabalhos que utilizam apenas o Fluxo de Carga sem a presença da computação paralela.

4.2 FLUXO DE CARGA

Zhang *et al*. (2016) faz uso do Fluxo de Carga, porém utilizando um intervalo de confiança para a solução em vez de aproximar os valores de tensão e potência. Desta forma, pretende evitar possíveis erros de medições devido incertezas durante a geração e carga do Sistema Elétrico de Potência. Foi posposto uma combinação entre duas formas de solução em intervalos do Fluxo de Carga, retangular e polar, tirando proveito da precisão do intervalo do ângulo e potência ativa da forma retangular, como também, da precisão do intervalo da magnitude de tensão da forma polar. O método proposto mostrou-se, a partir dos testes com os casos do IEEE de 30 e 118 barras elétricas, mais preciso do que os métodos convencionais.

Tamimi *et al.* (2016) devido ao elevado custo para aquisição de equipamentos de controle como compensadores estático fixo este trabalho, propõe uma utilização híbrida desses equipamentos, com controle de fluxo de carga unificados com baixo custo, tipo bancos de capacitores. Foi realizado uma modelagem, controle e aplicação em uma rede do Canadá levando em consideração o custo e as limitações físicas dos equipamentos. Como conclusão, o método proposto é uma opção atraente para alívio de congestionamento em redes de energia reais.

Naresh *et al.* (2016) faz uso do controle do Fluxo de Carga e de métodos para melhorar a qualidade de energia em sistemas de distribuição com geração residencial, prática cada vez mais comum com a utilização de placas fotovoltaica em estabelecimento de consumidores. Utilizando conversor dc-dc, conversor dc-ac e modulação por largura de pulso o autor aprimorou a qualidade de energia elétrica para o modelo testado.

Os trabalhos mencionados acima mostram o interesse na comunidade em aprimorar o Fluxo de Carga e/ou fazer inclusão de novas tendências que surgiram, como a produção de energia elétrica no sistema de distribuição, dessa forma há uma propensão no aumento da complexidade do Fluxo de Carga. Por isso, reforça-se a necessidade de uma resposta rápida

Com exceção do trabalho de Tamimi *et al.* (2016) que foi realizado em um sistema real do Canadá, os demais realizam teste de caso de pequena escala, visto que há uma tendência no crescimento do SEP, torna-se ideal a realização de testes de larga escala, com milhares de barras elétricas, de forma a ter uma noção do desempenho do trabalho ao ser aplicado em sistemas reais. A seguir, será apresentado trabalhos que realizam a paralelização do Fluxo de Carga, não necessariamente em GPU.

4.3 PARALELIZAÇÃO DO FLUXO DE CARGA

Bing Liang *et al.* (2016) realizam uma modelagem de forma que o fluxo de carga possa ser resolvido paralelo por *Map-Reduce*, através da plataforma de computação distribuída *Hadoop*. A proposta foi testada para o sistema de distribuição de 33 barras elétricas do IEEE mostrando viabilidade e eficácia.

Fukuyama (2015a), (2015b) e Iwata *et al.* (2016) utilizaram metaheurística afim de otimizar os sistemas elétricos de potência, executando múltiplos cálculos do fluxo de carga, modificando as variáveis de controle de operação reguladores automáticos de tensão, transformadores com *tap* variável e compensadores estáticos fixo. Foi realizado a paralelização dos cálculos por *multithread* em CPU com o *OpemMP*. Cada thread realizava todo o cálculo do fluxo de carga. Como resultado conseguiu um *speedup* de aproximadamente 4x para 8 threads.

Ye (2015) com foco na estabilidade de transitórios, utilizando otimização por Fluxo de Carga com o NSGA-II paralelizado em *cluster* de computadores no padrão mestre-escravo

pela a biblioteca *Open MPI*. Com 8 computadores possuindo processadores *Quad-Core Xeon* atingiu um speedup de aproximadamente 30x para o caso de 678 barras elétrica do IEEE.

Os trabalhos previamente apresentam paralelizações se valendo de diferentes plataformas/bibliotecas, nota-se a com exceção do trabalho Bing Liang *et al.* (2016), todos os trabalhos utilizam a aplicação de metaheurísticas com o intuito de otimizar os sistemas. Em razão disto a realização de múltiplos cálculos do Fluxo de Carga simultâneos contribuirá para o aumento de desempenho.

O trabalho de Ye (2015) obteve uma bom desempenho de *speedup* atingindo 30x, contudo para isso foram utilizados 8 servidores, o que eleva o custo para sua replicação. Por esta razão, buscou-se a paralelização por GPU, visto que possui um ótimo custo benefício, como apresentados nos trabalhos que paralelizam o Fluxo de Carga exclusivamente em GPU discutidos na próxima seção.

4.4 FLUXO DE CARGA EM GPU

Guo *et al.* (2012) realizou um comparativo dos métodos do cálculo do fluxo de carga Gauss-Seidel, Newton-Raphson e Desacoplamento Rápido, paralelizados na plataforma CUDA. Neste trabalho, os autores focaram as análises na identificação de qual método apresenta maior ganho de desempenho em sua versão paralela, obtendo assim, maior aproveitamento da plataforma supradita. Os resultados apresentados, mostraram que mesmo o método do Desacoplamento Rápido consumir menos tempo de execução, o de Newton-Raphson possui um maior ganho ao ser utilizado na plataforma paralela em GPGPU.

Błaśkiewicz *et al.* (2015) utilizaram o método de Newton-Rapshon para o cálculo do fluxo de carga, para redes de alta tensão DC, também utilizando a plataforma CUDA. Também comparou dois algoritmos para a etapa da resolução de sistemas lineares, a Decomposição LU e o Gauss-Jordan. Por meio das análises de casos de testes de sistemas elétricos de potência de até 900 barras, concluíram que o maior tempo consumido pelo cálculo do fluxo de carga está na etapa de resolução de sistemas lineares. Por isso, nesta dissertação, testou-se algoritmos paralelizados em GPU pela biblioteca CUBLAS para serem utilizados nessa etapa. Os resultados mostraram que a utilização de Decomposição LU pela

biblioteca testada, mostraram uma melhora de desempenho para casos de testes de larga escala.

Li *et al.* (2013) exploraram o método multifrontal com GPU, visando obter um melhor desempenho na etapa de resolução de sistemas lineares do cálculo do fluxo de carga. Este método transforma uma matriz esparsa em pequenas matrizes densas, visto que as matrizes presentes nos cálculos têm uma característica de esparsidade e a GPU tem um melhor aproveitamento com matrizes densas. Seus resultados apontaram um bom desempenho para umas bases de testes sintáticas, o qual eram geradas matrizes aleatórias esparsas. Porém, para os casos de testes de *benchmark* de sistemas elétricos de potência, está estratégia apresentou um baixo desempenho.

Li (2014) e Li *et al*. (2016) realizaram uma abordagem de métodos de resolução de sistemas lineares indiretos para o cálculo do fluxo de carga. Nestes estudos os autores afirmam que o método de decomposição LU é ideal para a utilização em CPU, mas para GPU, deve-se buscar alternativas como o método indireto Conjugado Gradiente. Os testes mostram um ganho de desempenho em GPU para SEPs de larga escala. Contudo, vale ressaltar que os autores realizam a comparação entre implementações no MATLAB para a versão sequencial, o que pode causar uma comparação injusta com a de GPU, visto que os códigos em MATLAB, na maioria dos casos são mais lentos do que quando convertidos para o C/C++.

Roberge *et al.* (2015) fazem uma comparação entre os métodos do cálculo de fluxo de carga, Gauss-Seidel e Newton-Raphson, realizando múltiplos cálculos concorrentemente. A etapa de resolução de sistemas lineares foi resolvida por multithread em CPU pela biblioteca Eigen. Os testes mostraram um melhor resultado para o Newton-Raphson, sendo paralelizado por apenas um grande *kernel* em cada etapa do cálculo do fluxo de carga. Além de afirmar que a resolução de sistema lineares esparsos em GPU ainda é um desafio, os autores também dizem que a utilização de matrizes esparsas tem um melhor resultado do que matrizes densas, apesar de não justificarem.

Rakai *et al.* (2014) pelo método preditor-corretor de Mehrotra, realizaram múltiplos cálculos do fluxo de carga, paralelizando em GPU a etapa de fatoração de matriz. Simulou-se com ponto flutuantes de precisão simples e dupla, apresentou ganhos de *speedup* de aproximadamente 4x.

Os trabalhos acima mostraram o Newton-Raphson como método de solução do Fluxo de Carga que tem um maior aproveitamento da GPU, apresentaram, também, algumas divergências quanto ao uso de matrizes esparsas ou densas. Adicionalmente, apontam que a resolução de sistemas lineares é a etapa que consome maior processamento, havendo um maior foco para a paralelização desta etapa apenas. Em vista disso, averígua-se as divergências encontrada para elaborar uma estratégia de paralelização que há uma maior eficiência tanto do hardware da GPU quanto da CPU, de maneira a difundir a proposta, bem como, no meio acadêmico quando no mercado.

A Tabela 4.1 exibe um resumo dos trabalhos correlatos apresentados com o objetivo de cada um, se foi executado de forma sequencial ou paralela, com qual plataforma e também quais foram os casos de testes utilizados.

Tabela 4.1 – Resumo dos trabalhos correlatos.

Autor	Execução	Objetivo	Sistema Testado*
(Zhang, C. et al, 2016)	Sequencial	Precisão Fluxo de Carga	[30 –118]
(Tamimi, B. et al, 2016)	Sequencial	Equipamentos de Controle	--
(Naresh, M. et al, 2016)	Sequencial	Geração Residencial	--
(Bing Liang et al, 2016)	Hadoop	Tempo de Processamento	[33]
(Fukuyama et al, 2015b)	OpenMP	Otimização por Metaheurísticas	[14 – 57]
(Iwata et al, 2016)	OpenMP	Otimização por Metaheurísticas	[118]
(Ye et al, 2015)	Open MPI	Otimização por Metaheurísticas	[39 – 678]
(Guo et al., 2012)	GPGPU	Comparação de Métodos	[9 – 300]

(Błaśkiewicz et al, 2015)	GPGPU	Sistemas de Corrente Contínua	[400 – 900]
(X. Li et al, 2013)	GPGPU	Otimizar Etapa de Sistema Linear	[2383 – 3375]
(X. Li et al , 2014)	GPGPU	Otimizar Etapa de Sistema Linear	[30 –1138]
(Rakai et al, 2014)	GPGPU	Otimizar Etapa de Sistema Linear	[30 – 3120]
(Roberge et al ,2015)	GPGPU	Múltiplas Execuções	[4 – 2383]

* Intervalo do número de Barras Elétricas dos sistemas testados

Alguns não há a informação do caso de teste por não serem utilizados casos de acesso público e sim privado. O trabalho de Roberge *et al.* (2015), foi o primeiro ensaio para realizações de múltiplos cálculos simultâneos, contudo concentrou a realização de todos os cálculos de cada etapa em um único kernel.

4.5 CONSIDERAÇÕES FINAIS

Neste capítulo, observou-se a tendência na complexidade do Fluxo de Carga, mostrando a importância de se obter uma resposta rápida do processamento. Também, notou-se a utilização de metaheurísticas com a paralelização do FC, dessa forma, a realização de múltiplos cálculos simultâneos contribuirá para trabalhos futuros que utilizarem essa abordagem de otimização. As divergências encontradas nos trabalhos de GPU devem ser analisadas, dessa maneira, poder difundir estratégias mais eficientes pode facilitar sua utilização e replicação. Por fim, os trabalhos têm realizado testes com casos pequenos, porém há uma necessidade de testar com casos de larga escala como os sistemas reais. No capítulo seguinte serão exibidas as estratégias de paralelização de múltiplos Fluxo de Carga simultâneos com GPGPU realizadas nesta dissertação.

5 ESTRATÉGIAS DE PARALELISMO DO FLUXO DE CARGA

5.1 CONSIDERAÇÕES INICIAIS

Neste capítulo serão apresentadas as principais estratégias elaboradas por esta dissertação para a paralelização de múltiplas execuções simultâneas em GPGPU. Primeiramente, desenvolveu-se uma versão para execução sequencial do Fluxo de Carga, depois, todas as etapas foram paralelizadas em CUDA de forma a comparar as versões, tanto com matrizes esparsas quanto densas. Dessa forma, pode-se confrontar as divergências relacionadas à quais etapas devem ser paralelizadas ou não em GPU e da abordagem de utilização de diferentes características de matrizes. Com base nisso, define-se as etapas que mais se adequam a arquitetura da plataforma CUDA e experimenta-se uma versão híbrida executando em CPU as que não se adaptaram a GPU. Por fim, verifica-se a aplicabilidade do paralelismo dinâmico CUDA para o estudo de caso de múltiplos cálculos do Fluxo de Carga.

Ao longo deste capítulo serão apresentadas as estratégias desenvolvidas e, em seguida, testes realizados para cada uma de forma a contribuir para o desenvolvimento das próximas. Em razão disto, a seguir serão exibidos casos de testes utilizados e as configurações de hardware do ambiente de teste.

5.2 BASE DE DADOS DOS TESTES E CONFIGURAÇÕES DE HARDWARE

Os casos de testes possuem origem do IEEE e do *Pan European Grid Advanced Simulation and State Estimation* (PEGASE). Ambos foram obtidos pelo pacote *open-source* MATPOWER para simulações com sistemas elétricos de potência em MATLAB (ZIMMERMAN; MURILLO-SANCHEZ; J, 2011).

No total foram utilizados 8 (oito) casos de testes, variando o tamanho de 14 até 9241 números de barras. A Tabela 5.1 exibe os detalhes de cada caso de teste quanto ao número de barras, linha de transmissão ou ramos, *Automatic Voltage Regulator*, *On Load Tap Canger* e *Shunt Reactor*, e também informa a origem do caso de teste.

Tabela 5.1 - Casos de Testes de Sistemas Elétricos de Potência.

Barras	Ramos	AVRs	OLTCs	ShRs	Origem
14	20	5	3	1	IEEE
30	41	6	4	2	IEEE
57	80	7	17	3	IEEE
118	186	54	9	14	IEEE
300	411	69	107	14	IEEE
1354	1991	260	234	1082	PEGASE
2869	4582	510	496	2197	PEGASE
9241	16049	1445	1319	7327	PEGASE

Para a realização dos testes sob as bases de dados de SEPs, utilizou-se um servidor com sistema operacional CentOS 6.5. Suas configurações de *host* e *device* são apresentadas a seguir:

- Processador: Intel Xeon E5-2620 v2, 6 núcleos, 15 MB Cache;
- Memória RAM: 16 GB;
- GPU: NVIDIA Tesla K20c, 2496 núcleos CUDA, 13 MP, 5 GB de memória;
- Versão do CUDA: 7.5.

Tendo conhecimento do ambiente e das bases dos testes, na próxima seção serão apresentadas estratégias para paralelizar o FC, começando pela versão sequencial a qual terá o desempenho comparado às versões paralelas.

5.3 VERSÃO SEQUENCIAL

Foi implementada em C++ a versão sequencial do método de solução do Fluxo de Carga Newton-Raphson com base no MATPOWER, um pacote de código aberto para simulações de sistemas elétricos de potência baseado em MATLAB. A Tabela 5.2 contém o pseudocódigo da versão sequencial em C++ baseada no código em MATLAB.

Tabela 5.2 - Pseudocódigo da versão sequencial.

Pseudocódigo
1: LeituraCasodeTeste
2: InicializarEquipamentodeControle
3: Alocaçãode memória
Fluxo de Carga
4: CalcularTensãodasBarras
5: MatrizAdmitancia
6: NNZ Matriz Jacobina
Newton-Raphson
7: CalcularCorrentedasBarras
8: MatrizJacobiana
9: Resolução do SistemaLinear
10: AtualizaçãoDasTensões
11: DesajustesPotência
Fim(Newton-Raphson)
12: Cacular Perda
Fim(Fluxo de Carga)

No passo 1 é realizada a leitura do caso de teste no formato do MATPOWER versão 2 e, por ser considerada uma constante no tempo de execução em relação ao caso de teste, não é levado em consideração o tempo deste passo.

No passo 2 é feita a inicialização de uma estrutura que conterá os valores das variáveis dos equipamentos de controle, de forma que, futuramente, seja modificada pela metaheurísticas, indicando qual equipamento (OLTC, ShC, ARV) e seu valor deseja avaliar. Por causa disso, pode-se executar diversas execuções de um mesmo sistema, porém variando apenas os equipamentos de controle em cada execução, evitando armazenar redundância de informação referente ao sistema. Atualmente os valores dos equipamentos de controle são os mesmos presente no caso de teste.

No passo 3 é realizada a alocação de memória necessária para os cálculos. A concentração em um único ponto reduz o impacto do tempo necessário para esta tarefa em GPU e por motivos de comparações a versão sequencial também realizará a alocação em apenas um ponto.

No passo 4 são realizados os cálculos das tensões das barras e, no passo 5, os da Matriz de Admitâncias, onde, em seguida, pode-se verificar quantos elementos não nulos a matriz jacobiana terá, bem como alocar o espaço de memória necessário, sendo este o passo 6.

No passo 7, as correntes das barras elétricas são calculadas de forma a auxiliar no cálculo da matriz jacobiana realizado no passo 8. Faz-se, no passo 9, a resolução de sistemas lineares de forma a encontrar os ajustes necessários das tensões afim de reduzir os desajustes das potências.

No passo 10, os valores de tensão são atualizados a partir da solução encontrada no passo anterior e no passo 11, verifica-se se a solução encontrada é satisfatória. A partir da utilização de dois critérios, o erro do desajuste das potências ser inferior a 10^{-8} ou o atingir o número máximo de 10 iterações.

No caso de o resultado não satisfazer os critérios, realiza-se o processo novamente desde o passo 7, caso contrário, finaliza-se o newton-Raphson e se prossegue para o passo 12, onde é realizado o cálculo da perda de potência ativa de todo o Sistema elétrico de potência.

Caso apenas o critério do número máximo de iterações seja satisfeito, considera-se que a solução encontrada não convergiu, por isso atribui-se o valor de perda como o pior possível. No caso do C++, o maior número de uma variável de precisão dupla é 10^{37}. Já quando a solução converge a perda é calculada conforme a equação (5.1).

$$perda = \sum_{k=1}^{n-1} \sum_{m=k+1}^{n} P_{km} + P_{mk} \tag{5.1}$$

A equação (5.1) calcula a potência ativa no sentido das barras *k* para *m* e no sentindo inverso de *m* para *k*, em seguida, soma-se as duas potências. Caso a soma das duas potências seja diferente de zero, quer dizer que houve uma perda durante a transmissão e que a potência gerada por uma barra não foi totalmente consumida pela outra, já que uma possui sinal negativo por estar no sentindo inverso da corrente.

Para obter perda total de todo o SEP esse procedimento é realizado para todas as *n* barras do sistema, de forma facilitar a validação dos resultados obtidos nos cálculos com um único valor, em vez de todos os valores de tensão e potência resultantes do Fluxo de Carga serem comparados.

Após a validação da versão sequencial, buscou-se otimizar o código, visto que foi desenvolvido com base em códigos do MATLAB, que é otimizado para operações com

matrizes. Para isso, fez-se o uso de bibliotecas para operações de matrizes e resolução de sistemas lineares.

A Intel *Math Kernal Library* (MKL) é bastante utilizada em trabalhos de comparações de CPU com GPU, tanto de operação de matrizes quanto de resolução de sistema lineares, mostrando sua eficiência e consolidação no meio acadêmico. Além disso, há a biblioteca *Eigen*, que também promove a resolução de sistemas lineares e apresenta bons resultado no trabalho de Roberge *et al.* (2015) discutido no capitulo 4.4.

A Tabela 5.3 exibe o comparativo entre as duas bibliotecas para resolução de sistemas lineares, onde são exibidos os tempos médios em segundos e o desvio padrão de 5 simulações com 1 e 200 cálculos simultâneos do FC para as bases de testes com 14, 30, 300 e 9241 barras elétricas.

Tabela 5.3 - Teste com Eigen e MKL.

Caso de teste		***Cálculos Simultâneos***			
		1		***200***	
	Biblioteca	***Eigen***	***MKL***	***Eigen***	***MKL***
14	***AVG***	0,032	**0,013**	**0,110**	0,125
	STD	0,008	0,020	0,013	0,007
30	***AVG***	0,049	**0,005**	0,261	**0,208**
	STD	0,010	0,001	0,022	0,002
300	***AVG***	0,478	**0,014**	12,053	**1,998**
	STD	0,117	0,001	0,328	0,077
9241	***AVG***	283,516	**0,403**	15053,656	**79,065**
	STD	5,534	0,005	179,832	0,577

Percebe-se que a *Eigen* obteve o melhor tempo apenas para a combinação do caso de teste de 14 Barras com 200 cálculos simultâneos, mostrando que a biblioteca tem uma melhora de desempenho quando aumentada a quantidade de cálculos simultâneos, porém é inibida ao se aumentar a complexidade do SEP.

Nas demais combinações a MKL apresentou um melhor desempenho atingindo um *speedup* máxima de 703,51x para o caso de 9241 barras com apenas um cálculo do FC. Por esse motivo adotou-se a biblioteca MKL para o desenvolvimento deste trabalho.

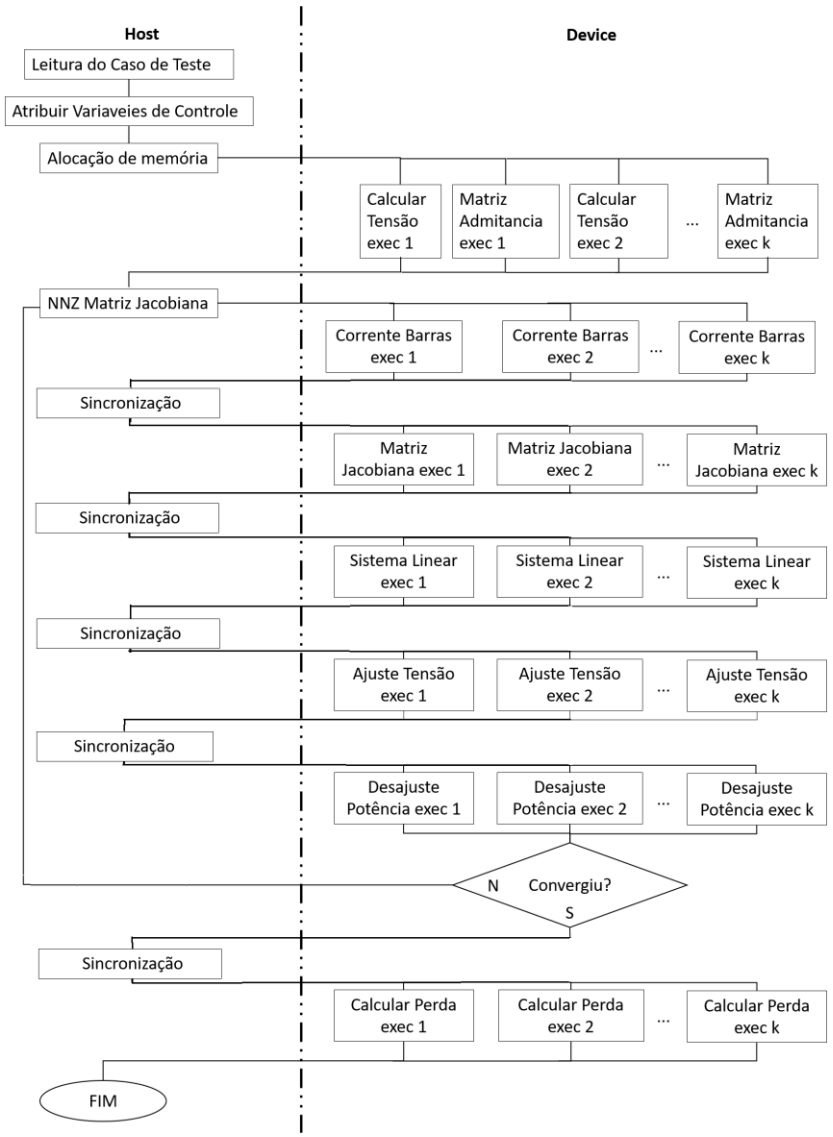
5.4 VERSÃO TOTALMENTE PARALELIZADA ESPARSA

Com a versão sequencial pronta e otimizada, realizou-se a paralelização em CUDA de todas as etapas do FC, utilizando matrizes esparsas através da biblioteca cuSparse e a resolução de sistemas lineares pela cuSolver, ambas oferecidas pela própria NVIDIA.

Um *kernel* é gerado para cada execução do FC, onde cada *thread* é uma barra elétrica e é atribuído o valor de 1024 *threads* por bloco, visto que esse é o limite máximo da placa utilizada nos testes.

A Figura 5.1 apresenta o fluxograma da versão paralelizada em CUDA.

Figura 5.1 – Fluxograma de execução de múltiplos cálculos do FC paralelo.



Na Figura 5.1, percebe-se que todas as etapas são realizadas em GPU, mas que em cada etapa é criado um *kernel* para cada execução do Fluxo de Carga, de forma a calcular *k* execuções simultâneas.

Também, nota-se que as etapas de cálculo das tensões das barras e matriz de Admitância estão sendo executadas no mesmo instante, isso porque não há uma sincronização ente as etapas devido não existir uma dependência de dados entre estas. Desta maneira reduz-se a ociosidade gerada por uma sincronização.

Por último identifica-se a execução da verificação de números não zeros (NNZ) da matriz jacobiana a ser executada em CPU e não em GPU, isso porque a GPU trabalha com memória estática, sendo necessário já ter sido alocada antes da chamada do *kernel*, e esta etapa é um preparatório para a execução do *kernel* que calcula a jacobiana.

A paralelização da etapa do cálculo da perda de potência foi diferente das demais, pois além de ser calculada a perda de cada linha de transmissão por uma *thread*, foi necessário realizar o somatório de todos os valores presentes em diferentes *threads*. Para isso, fez-se uso de uma *reduction*, um padrão de computação paralela no qual os valores contidos em vários processadores são “reduzidos” para apenas um valor ou um processador por meio de operações algébricas ou booleanas como máximo, mínimo, somatória.

Testou-se bibliotecas que realizam essa operação como a *Thrust API*, também oferecida suporte pela NVIDIA. Contudo, o resultado obtido foi mais lento do que a execução sequencial, devido ser chamada várias vezes, já que são realizados múltiplos cálculos do fluxo de carga e haver muita sincronização interna na biblioteca. Portanto, utilizou-se uma modificação do código de Mark Harris, desenvolvedor da NVDIA, que disponibilizou uma apresentação sobre otimizações de *reductions* em CUDA. A modificação foi para execução de *reductions* concorrentemente.

A Tabela 5.4, exibe o pseudocódigo, com os tempos em segundos de CPU e GPU de cada etapa do FC, juntamente com o *speedup*, para 200 execuções simultâneas com o caso de teste de 300 barras elétricas.

Tabela 5.4 - Pseudocódigo, com speedup e os tempos em segundos de CPU e GPU de cada etapa do FC para 200 execuções simultâneas com o caso de teste de 300 barras elétricas.

Pseudocódigo	CPU	GPU	Speedup
1: LeituraCasodeTeste	--	--	--
2: InicializarEquipamentodeControle	0,007	0,009	0,78
3: Alocaçãode memória	0,001	1,566	0,00
Fluxo de Carga	4,658	23,025	0,20
4: CalcularTensãodasBarras	0,025	--	--
5: MatrizAdmitancia	0,114	0,128	1,09
6: NNZ Matriz Jacobina	0,001	0,002	0,50
Newton-Raphson	4,505	22,889	0,20
7: CalcularCorrentedasBarras	0,045	0,012	3,75
8: MatrizJacobiana	0,114	0,031	3,68
9: Resolução do SistemaLinear	4,072	22,796	0,18
10: AtualizaçãoDasTensões	0,066	0,008	8,25
11: DesajustesPotência	0,241	0,040	6,03
Fim(Newton-Raphson)	--	--	--
12: Cacular Perda	0,012	0,006	2,00
Fim(Fluxo de Carga)	--	--	--
Total	**4,666**	**24,600**	**0,19**

Nota-se que a etapa com mais consumo de processamento é a de resolução do sistema linear, que para GPU apresentou um baixo desempenho sendo mais de 5 vezes mais lenta do que em CPU. Isso ocorre devido a utilização de matrizes esparsas em GPU ainda ser um desafio, pois o padrão de acesso a memória não é uniforme. Contudo, as demais etapas paralelizadas apresentaram um ganho no desempenho.

O *speedup* do cálculo das tensões não é exibido devido na versão paralelizada não haver uma sincronização entre a etapa e da matriz jacobina, impossibilitando a coleta do tempo apenas de uma etapa. Por isso os *speedup* das duas etapas são exibidos na etapa da matriz de admitâncias realizando a soma dos tempos de ambas as etapas na versão sequencial.

Para tentar melhorar a eficiência da etapa de resolução de sistema lineares e também solucionar a divergência encontrada na literatura sobre a utilização de matriz esparsa e densa, utilizou-se esta mesma versão totalmente paralelizada só que com a abordagem de matrizes densas, que será apresentada na próxima seção.

5.5 VERSÃO TOTALMENTE PARALELIZADA DENSA

O problema da utilização de matrizes esparsas está no padrão de acesso a memória. Como a memória global possui uma latência alta e devido o foco ser para sistemas de larga escala, impossibilitando o uso inteiramente dos outros níveis de memória presentes na arquitetura da GPU, a etapa de resolução de sistema lineares foi prejudicada. Com a utilização de matrizes densas, o padrão de acesso a memória será melhorado, porém a quantidade de elementos será aumentada.

A utilização de matrizes densas tem como objetivo tentar melhorar o desempenho da etapa de sistemas lineares, mais também de solucionar a divergência encontrada nos trabalhos e determinar qual a abordagem é melhor de se utilizar para o estudo de caso de múltiplos fluxo de carga, densa ou esparsa.

A Tabela 5.5 exibe o pseudocódigo, com os tempos em segundos de CPU, GPU com matrizes esparsas e GPU com matrizes densas, de cada etapa do FC, juntamente com o *speedup* da versão sequencial com paralelizada com matrizes densas, para 200 execuções simultâneas com o caso de teste de 300 barras elétricas.

Vale ressaltar que como as matrizes densas estão sendo utilizadas, não há necessidade de verificar a quantidade de números não zeros (NNZ) da matriz jacobina.

Tabela 5.5 - Pseudocódigo, com o speedup e os tempos em segundos de CPU, GPU com matrizes esparsas e densas de cada etapa do FC para 200 execuções simultâneas com o caso de teste de 300 barras elétricas

Pseudocódigo	**CPU**	**Esparsa**	**Densa**	**Speedup**
2: InicializarEquipamentodeControle	0,007	0,009	0,442	0,02
3: Alocaçãode memória	0,001	1,566	1,282	0,00
Fluxo de Carga	4,658	23,025	9,473	0,49
4: CalcularTensãodasBarras	0,025	--	--	--
5: MatrizAdmitancia	0,114	0,128	0,287	0,48
6: NNZ Matriz Jacobina	0,001	0,002	0,000	--
Newton-Raphson	4,505	22,889	8,412	0,54
7: CalcularCorrentedasBarras	0,045	0,012	0,070	0,64
8: MatrizJacobiana	0,114	0,031	1,164	0,10
9: Resolução do SistemaLinear	4,072	22,796	6,395	0,64
10: AtualizaçãoDasTensões	0,066	0,008	0,008	8,25
11: DesajustesPotência	0,241	0,040	0,776	0,31
Fim(Newton-Raphson)	--	--		--
12: Cacular Perda	0,012	0,006	0,338	0,04
Total	**4,666**	**24,600**	**10,764**	**0,43**

Notou-se uma melhora na etapa de resolução de sistemas lineares, utilizando matrizes densas, porém não foi o suficiente para superar a versão sequencial. Além disso, as demais etapas que haviam apresentado um melhor desempenho foram prejudicadas, com exceção da atualização das tensões, onde as matrizes não estão presentes nos cálculos.

O mais impactante foi o aumento do consumo da memória, que inviabilizou a execução de sistemas de larga escala. Não se pode realizar nem uma execução do Fluxo de Carga para o maior caso de teste presente, de 9241 barras elétricas.

Definindo-se a utilização de matrizes esparsas com melhor abordagem para a realização de múltiplos cálculos do fluxo de carga, ainda há o problema de que a etapa de resolução de sistema lineares esparsos não se adaptou a arquitetura da GPU, devido ao padrão de acesso a memória.

Diante disso, será realizada uma versão híbrida com a etapa que não se adaptou a plataforma em CPU e as demais em GPU. Esta versão será apresentada com mais detalhes na próxima seção.

5.6 VERSÃO HÍBRIDA PARALELIZADA

A etapa de resolução de sistemas lineares em matrizes esparsas não se adaptou a arquitetura da plataforma CUDA, devido ao acesso a memória. Esse problema não será relevante caso seja executado em CPU, por causa do projeto de hardware o qual a CPU foi projetada, com a ideia de reduzir a latência, voltando mais área de seu chip para memória cache.

A execução da etapa de resolução de sistema lineares em CPU consumirá mais tempo na versão paralelizada pois as informações que devem ser calculadas estão na GPU e é necessário a transferência pelo barramento para a memória do *host* e ao término do processamento da etapa, enviar os resultados para a memória do *device* para dar continuidade aos cálculos do fluxo de carga. Por esta razão, com o intuito de amenizar o tempo consumido pela comunicação entre as memórias, é realizada a paralelização em CPU por *multithread*.

A paralelização desta etapa está nos cálculos efetuados e não na execução múltipla, sendo assim as k execuções do FC são solucionadas sequencialmente, porém cada uma

paralela apenas na etapa de resolução de sistema lineares. A execução por *multithread* do método de resolução de sistema lineares esparsos é oferecido pela biblioteca MKL.

A Tabela 5.6, exibe o pseudocódigo, com os tempos em segundos de versão sequencial e da versão hibrida paralelizada das etapas do FC, juntamente com o *speedup*, para 200 execuções simultâneas com o caso de teste de 300 barras elétricas.

Tabela 5.6 - Pseudocódigo, com speedup e os tempos em segundos de versão sequencial e da versão hibrida paralelizada das etapas do FC para 200 execuções simultâneas com o caso de teste de 300 barras elétricas.

Pseudocódigo	**CPU**	**Híbrida**	**Speedup**
1: LeituraCasodeTeste	--	--	
2: InicializarEquipamentodeControle	0,007	0,009	0,78
3: Alocaçãode memória	0,001	1,5	0,00
Fluxo de Carga	4,658	2,156	2,16
4: CalcularTensãodasBarras	0,025	--	--
5: MatrizAdmitancia	0,114	0,062	2,24
6: NNZ Matriz Jacobina	0,001	0,001	1,00
Newton-Raphson	4,505	2,086	2,16
7: CalcularCorrentedasBarras	0,045	0,013	3,46
8: MatrizJacobiana	0,114	0,020	5,70
9: Resolução do SistemaLinear	4,072	1,998	2,04
10: AtualizaçãoDasTensões	0,066	0,008	8,25
11: DesajustesPotência	0,241	0,045	5,36
Fim(Newton-Raphson)	--	--	--
12: Cacular Perda	0,012	0,008	1,50
Fim(Fluxo de Carga)	--	--	--
Total	**4,666**	**3,665**	**1,27**

Percebe-se que a paralelização do passo 9 em CPU por *multithread* não só compensou o tempo da troca de informação das memórias como beneficiou, tornando mais eficiente que a versão sequencial. A versão híbrida apresentou resultados eficientes para todas as etapas do FC, tendo apenas um gargalo na alocação de memória e na inicialização dos valores dos equipamentos de controle, pois é necessário enviar as informações para a memória da GPU. Contudo, esses problemas não impactaram no resultado final deste caso de teste, tendo um *speedup* final de 1,27x.

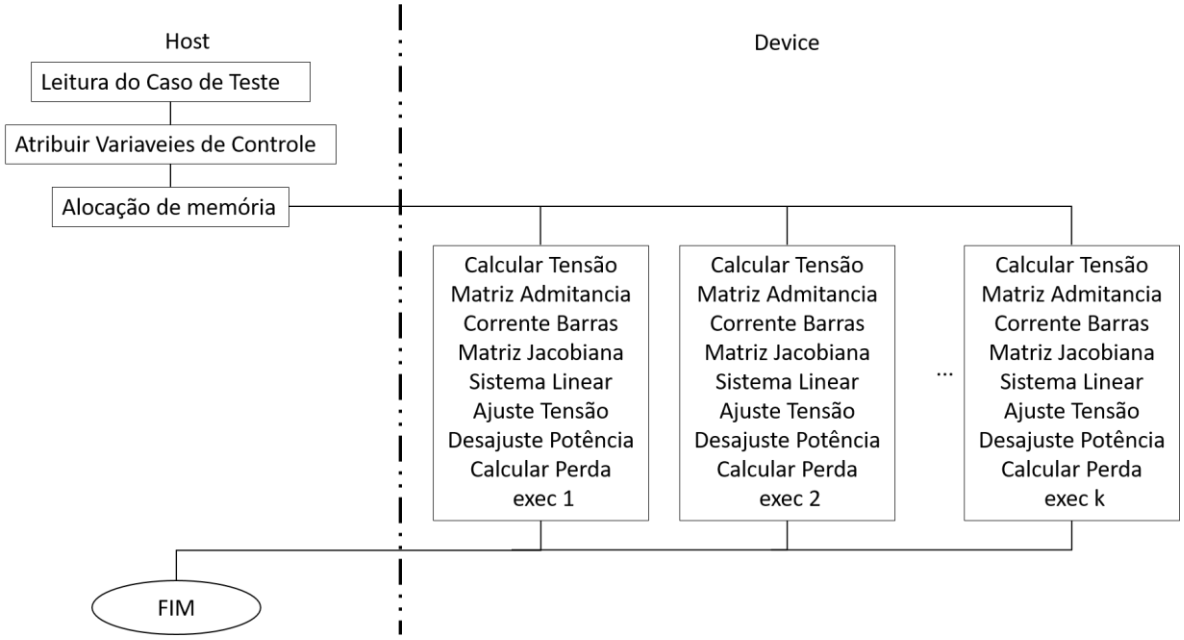
Alguns trabalhos cogitam em uma melhora no desempenho na paralelização do FC, por meio do recurso de paralelismo dinâmico CUDA. Como não há trabalhos que fazem uso no estudo de caso de cálculos de tensão e potência dos SEP, será desenvolvida uma versão

para avaliação da viabilidade deste recurso no estudo de caso. Na próxima seção será apresentado melhor como foi feita essa análise.

5.7 VERSÃO COM PARALELISMO DINÂMICO CUDA

A principal vantagem no CDP é a execução de novos *kernels* a partir de *kernels* já existentes, ou seja, chamar um processamento em GPU de uma função que já está sendo executada em GPU. A partir disso, pode-se criar uma independência entre as execuções do FC, evitando uma sincronização entre as execuções simultâneas. A Figura 5.2 exibe o fluxograma de execução da versão com paralelismo dinâmico CUDA.

Figura 5.2 - Fluxograma de execução da versão com paralelismo dinâmico CUDA.



Percebe-se na Figura 5.2 a ausência de sincronização entre os *kernels* criados para cada execução do FC e cada *kernel* realiza a chamada para todas as etapas do FC em diferentes *kernel*, dessa forma caso uma execução termine uma etapa, não será necessário esperar o processamento de outra execução para dar continuidade.

Um problema identificado antes da realização dos testes, foi que a resolução de sistema lineares iria ocorrer em GPU e que só poderia ser realizada com matriz densa, devido a única biblioteca (cuBLAS) que oferece suporte para essa operação com esse recurso do

paralelismo, utilizar apenas a abordagem de matriz densa. Contudo, espera-se que a falta de sincronismo compense este problema.

Outro ponto que deve ser mencionando é que pelo fato das etapas do FC estarem executando de forma assíncrona entre os cálculos simultâneos, não há como coletar os tempos de processamento de cada etapa como foi realizado nas outras versões, pode-se apenas verificar o tempo total exigido.

A Tabela 5.7, exibe o tempo em segundos da execução total das versões sequencial, híbrida e com CDP, juntamente com o *speedup* da sequencial pelo CDP, para 200 execuções simultâneas com o caso de teste de 300 barras elétricas.

Tabela 5.7 - Tempo em segundos da execução total da versão sequencial, hibrida e com CDP, juntamente com o *speedup* da sequencial pelo CDP, para 200 execuções simultâneas com o caso de teste de 300 barras elétricas.

	Sequencial	**Híbrido**	**CDP**	**Speedup**
Total	4,666	3,665	37,292	0,12

Como pode-se perceber, a versão com CDP não apresentou bons resultados, provavelmente o que a prejudicou foi a etapa de resolução de sistemas lineares, que precisou ser realizada com outra biblioteca, pois a utilizada nas outras versões não possui suporte para o recurso de paralelismo dinâmico CUDA.

O teste mostrou que a utilização de CDP para o estudo de caso desta dissertação não é viável atualmente, devido ao baixo desempenho da etapa de resolução de sistemas lineares em GPU e principalmente por oferecer suporte apenas para matrizes densas, impossibilitando a execução para casos de testes de larga escala, visto que há uma tendência no crescimento dos SEP.

Por fim, a Tabela 5.8 apresenta os resultados de tempo e speedup para todas as estratégias paralelas da versão sequencial para o caso de 300 barras eletricas com 200 avaliações simultâneas.

Tabela 5.8 – Tempo(s) e Speedup de todas as estratégias para o caso de 300 barras elétricas com 200 execuções simultâneas.

	Sequencial	**Híbrido**	**Denso**	**Esparso**	**CDP**
Tempo(s)	4,67	3,67	10,76	24,60	37,29
Speedup	--	1,27	0,43	0,19	0,12

5.8 CONSIDERAÇÕES FINAIS

Neste capitulo, foram apresentadas as estratégias para paralelização de múltiplos cálculos do FC simultâneos e se identificou que as etapas do FC estão mais adequadas à plataforma paralela da NVIDIA. Também, constatou-se que a utilização de matrizes densas inviabiliza a execução de casos de testes de larga escala, tornando-se um problema, visto que a tendência dos sistemas é aumentar.

Adicionalmente foi verificada a viabilidade da utilização de CDP para este estudo, comprovando que sua utilização atualmente não melhora a eficiência para o problema de execuções do FC como cogitado por alguns trabalhos.

Foram apresentados testes a exemplo apenas do caso de 300 barras elétricas. Testes com demais casos e outras análises com relação ao *speedup* serão apresentados no próximo capitulo.

6 TESTES E RESULTADOS

6.1 CONSIDERAÇÕES INICIAIS

Neste capítulo serão apresentados os testes entre as versões sequencial e paralela do fluxo de carga visando a realização de múltiplos cálculos de um mesmo SEP, com valores dos equipamentos de controle deferentes.

Os testes são realizados com sistema de larga escala de forma a analisar o desempenho nesses sistemas, visto que os sistemas possuem tendência de crescimento. Além disso, as versões paralelas serão comparadas de forma a definir a mais eficiente para difundir no meio acadêmico e comercial por meio de plataforma de compartilhamento de códigos, facilitando, assim, a replicação para outros trabalhos relacionados.

6.2 RESULTADOS

Foram realizados testes com as categorias de execuções simultâneas 1, 10, 50,100 e 200 combinadas com as 8 bases de dados de SEP, apresentados na Tabela 5.1, totalizando 40 testes onde foram realizadas cinco simulações para cada combinação com a finalidade de definir a média aritmética do tempo, em segundo, do cálculo do FC e da execução total com a inclusão do tempo de alocação de memória. Os testes completos podem ser vistos no anexo A.

Por fins justos de comparação, os fluxos de carga simultâneos utilizaram os mesmos parâmetros dos equipamentos de controle (AVR, ShC, OLTC) presentes na base de dados. Adicionalmente foram utilizados como parâmetros de convergência um erro das potências inferior a 10^{-8} e número máximo de 10 iterações.

A Figura 6.1 apresenta o gráfico de *speedup* das 4 versões paralelas do fluxo de carga variando o caso de teste de 14 até 9241 barras elétricas para 200 execuções simultâneas. Valer ressaltar que nas versões Densa e CDP, por utilizarem o preenchimento completo das matrizes, não foi possível executar testes com as bases a partir de 1354 barras, devido ao consumo de memória superar o disponível. Também pode ser visto na Tabela 6.1 os tempo em segundos utilizados no cálculo do speedup da Figura 6.1.

Figura 6.1- Speedup das 4 versões paralelas do fluxo de carga variando o caso de teste de 14 até 9241 barras elétricas para 200 execuções simultâneas.

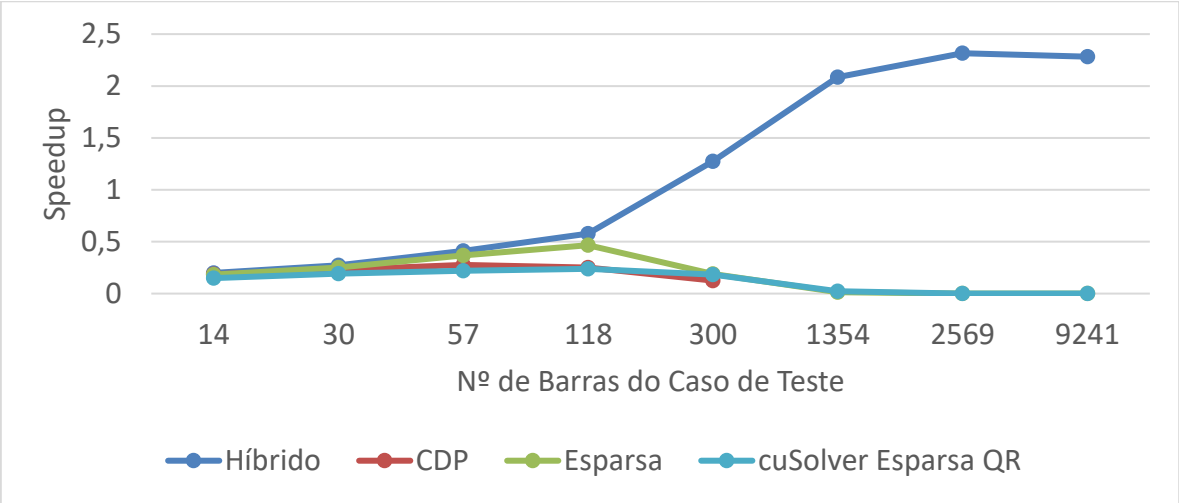


Tabela 6.1 - Tempo(s) das 4 versões paralelas do fluxo de carga e da versão sequencial, variando o caso de teste de 14 até 9241 barras elétricas para 200 execuções simultâneas.

Nº Barras	Sequencial	Híbrida	CDP	Esparsa	Densa
14	0,338	1,696	1,823	1,841	1,726
30	0,487	1,784	2,121	1,933	1,865
57	0,832	2,025	3,01	2,269	2,289
118	1,222	2,116	4,886	2,621	3,335
300	4,666	3,665	37,292	24,6	10,764
1354	17,665	8,473	--	1.302,09	--
2569	57,314	24,749	--	18.882,432	--
9241	197,247	86,46	--	801.548,994	--

A Figura 6.1 mostra que a versão híbrida teve um melhor desempenho a partir do caso de 300 barras, chegando a um *speedup* de quase 2,5x. O desempenho nos casos menores está diretamente ligado ao gargalo da GPU quanto a alocação de memória. Para demonstrar isso, a Tabela 6.2 apresenta o pseudocódigo, com os tempos em segundos de todas as versões das etapas do FC, para 1 execução com o menor caso de teste de 14 barras elétricas.

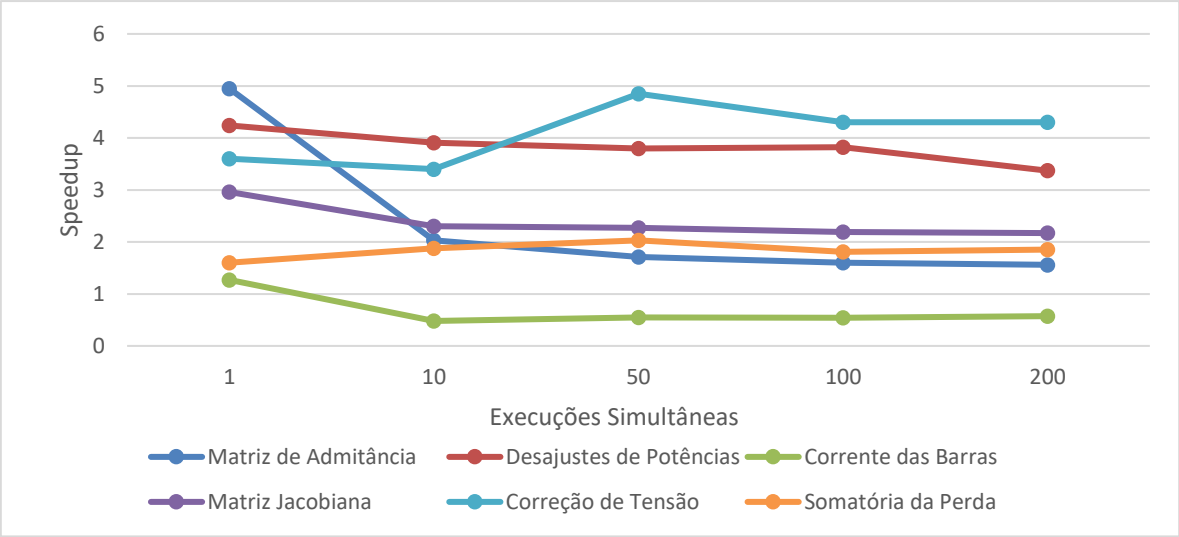
Tabela 6.2 - Pseudocódigo com os tempos em segundos de todas as versões das etapas do FC para 1 execução com o caso de teste de 14 barras elétricas.

Pseudocódigo	CPU	Híbrido	Esparsa	Densa	CDP
1: LeituraCasodeTeste	--	--	--	--	--
2: InicializarEquipamentodeControle	0,001	0,001	0,001	0,001	0,01
3: Alocaçãode memória	0,001	1,51	1,517	1,285	1,504
Fluxo de Carga	0,114	0,018	0,141	0,242	0,007
4: CalcularTensãodasBarras	0,015	--	--	--	--
5: MatrizAdmitancia	0,059	0,003	0,002	0,105	--
6: NNZ Matriz Jacobina	0,001	0,001	0,001	--	--
Newton-Raphson	0,037	0,015	0,138	0,134	--
7: CalcularCorrentedasBarras	0,008	0,001	0,001	0,001	--
8: MatrizJacobiana	0,008	0,001	0,001	0,001	--
9: Resolução do SistemaLinear	0,001	0,013	0,135	0,131	--
10: AtualizaçãoDasTensões	0,008	0,001	0,001	0,001	--
11: DesajustesPotência	0,012	0,001	0,01	0,002	--
Fim(Newton-Raphson)	--	--			--
12: Cacular Perda	0,003	0,001	0,001	0,001	--
Total	**0,115**	**1,528**	**1,658**	**1,527**	**1,512**

Percebe-se na Tabela 6.2 que apesar de o tempo para o cálculo do fluxo de carga ser mais rápido nas versões Hibrida e CDP, atingindo um *speedup* de 6,33x e 16,28, respectivamente, na execução total tiveram um baixo desempenho devido o tempo de alocação de memória para esse caso consumir mais de 98% do tempo. Ressaltando que para a versão CDP não se pode coletar os tempos de cada etapa, apenas do cálculo do FC e alocação de memória.

A versão híbrida possui melhor desempenho por executar a etapa de resolução de sistemas lineares em CPU, tirando proveito da baixa latência da arquitetura, é necessário fazer a análise das demais etapas do FC em GPU, para isso observa-se na Figura 6.2 e a Tabela 6.3 o *speedup* das etapas do FC em GPU da versão híbrida para o caso de 9241 barras elétricas variando o número de execuções simultâneas.

Figura 6.2 - Speedup das etapas do FC em GPU da versão híbrida para o caso de teste de 9241 Barras variando o número de execuções simultâneas.

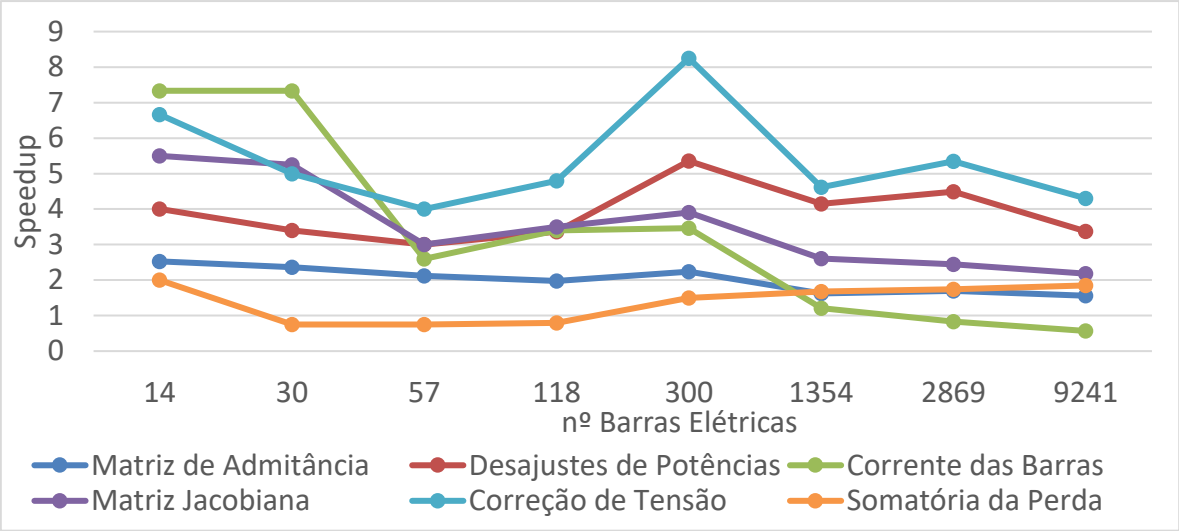


Percebe-se que em todas as etapas, ao crescer os números de execuções simultâneas o *speedup* estabiliza com valores acima de 1x, com exceção do cálculo da corrente, que obteve um *speedup* médio de 0,68x. Contudo este baixo desempenho está ligada ao padrão de conexões do caso de teste. Isso pode ser identificado quando é fixado o número de execuções simultâneas e realizada a variação dos números de barras dos casos de testes, como pode ser visto na Figura 6.3 e Tabela 6.4.

Tabela 6.3 - Speedup das etapas do FC em GPU da versão híbrida para o caso de teste de 9241 Barras variando o número de execuções simultâneas.

Etapas do FC	**1**	**10**	**50**	**100**	**200**	**AVG**
Matriz de Admitância	4,95	2,03	1,71	1,60	1,56	**2,37**
Desajustes de Potências	4,24	3,91	3,80	3,82	3,37	**3,83**
Corrente das Barras	1,27	0,48	0,55	0,54	0,57	**0,68**
Matriz Jacobiana	2,96	2,30	2,27	2,19	2,17	**2,38**
Correção de Tensão	3,60	3,40	4,85	4,30	4,30	**4,09**
Somatória da Perda	1,60	1,88	2,03	1,81	1,85	**1,83**

Figura 6.3 - Speedup das etapas do FC em GPU da versão híbrida para 200 execuções simultâneas variando o número de barras elétricas.



Nota-se que a etapa de cálculo da corrente das barras só apresentou baixo desempenho, abaixo de 1x, para os casos de 2869 e 9241 barras e um *speedup* médio de 3,34x. Adicionalmente, não se pode perceber um padrão do *speedup* com relação ao número de barras do sistema, reforçando a ideia de que as conexões dos SEP's influenciam no desempenho. Também se constata que todas as etapas resultaram em um *speedup* médio acima de 1x mostrando que estão adequas a plataforma CUDA.

Tabela 6.4 - Speedup das etapas do FC em GPU da versão híbrida para 200 execuções simultâneas variando o número de barras elétricas.

Etapas do FC	**14**	**30**	**57**	**118**	**300**	**1354**	**2869**	**9241**	**AVG**
Matriz de Admitância	2,53	2,37	2,12	1,98	2,24	1,63	1,70	1,56	**2,02**
Desajustes de Potências	4,00	3,40	3,00	3,37	5,36	4,15	4,49	3,37	**3,89**
Corrente das Barras	7,33	7,33	2,60	3,40	3,46	1,21	0,83	0,57	**3,34**
Matriz Jacobiana	5,50	5,25	3,00	3,50	3,90	2,60	2,45	2,18	**3,55**
Correção de Tensão	6,67	5,00	4,00	4,80	8,25	4,62	5,34	4,30	**5,37**
Somatória da Perda	2,00	0,75	0,75	0,80	1,50	1,68	1,74	1,85	**1,38**

A plataforma CUDA apresentou ser benéfica na realização de múltiplos cálculos do FC, com exceção da etapa de resolução de sistema lineares que tem uma maior eficiência na arquitetura da CPU. Por isso a versão híbrida obteve melhores resultados e seus códigos estão disponíveis para acesso público no repositório do GitHub pelo link https://goo.gl/4jvbxO de forma a difundir a melhor estratégia elaborada por este trabalho.

6.3 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os testes e resultados das estratégias de paralelização. Conclui-se que a melhor estratégia de paralelização do FC é a versão híbrida, onde a resolução de sistemas lineares é realizada em CPU, apresentando melhor desempenho na utilização de matrizes esparsas quando comparadas as densas.

Foram realizados testes com sistemas de larga escala, analisou-se o desempenho das etapas do FC e os códigos da estratégia híbrida foram disponibilizados para acesso público. No próximo capítulo serão apresentadas as conclusões dos resultados deste trabalho com mais detalhes.

7 CONCLUSÃO

A presente dissertação teve como o objetivo a realização de um estudo de estratégias de paralelização de múltiplos cálculos simultâneos de Fluxo de Carga com GPGPU, mais especificamente, a instigação do uso de matrizes esparsas e densas, bem como avaliar quais etapas do FC são mais adequadas ou não a arquitetura da GPU e examinar a utilização do paralelismo dinâmico CUDA para este estudo. Também teve como objetivo a aquisição de uma resposta rápida para o processamento do FC, testar e difundir com sistemas de larga escala a estratégia mais eficiente de forma a contribuir para trabalhos futuros.

A utilização de matrizes densas apresenta uma vantagem em relação as matrizes esparsas em GPU, principalmente a abordagem com CDP para sistemas de pequena escala. Porém, a utilização dessas matrizes apresentou um consumo de memória bastante elevado que impossibilitou a execução de 200 cálculos simultâneos para os casos de testes maiores que 300 barras elétricas, não sendo suficiente os 5GB disponíveis da placa gráfica, tornando, desta forma, a utilização de matrizes densas inviável, o que torna a abordagem com esparsas a melhor opção.

Devido ao *overhead* da alocação de memória em GPU, os testes constataram que a utilização de GPGPU só se tornará vantajosa para casos de testes a partir de 300 barras, o que para casos pequenos a paralelização em GPU não se mostrou uma boa opção. Pode ser que paralelizar por *multithread* em CPU seja uma boa alternativa neste caso. Contudo, a tendência é um crescimento da complexidade e tamanho dos SEP, tornando o ambiente ideal para a paralelização em placa de gráficas.

Muitos trabalhos focam na paralelização da etapa de maior consumo de processamento, a etapa de resolução de sistemas lineares. Entretanto esta etapa possui muita dependência de dados, onde só pode ser realizado um processamento quando já terminou outro. Por esta razão, a utilização de GPU sofre desvantagens, além de que as matrizes esparsas possuem um padrão de acesso a memória não uniforme, sendo este outro ponto em que a utilização dessa arquitetura será prejudicial.

Os testes mostram que a arquitetura baseada em redução de latência focando na memória cache como a de CPU obtém benefícios na resolução de sistemas lineares. Adicionalmente, percebe-se que possui um maior proveito da GPU paralelizando as demais

etapas do Fluxo de Carga, atingindo um *speedup* médio 5,37 para a correção das tensões de cada barra elétrica.

Por fim a utilização de paralelismo dinâmico cogitado por alguns trabalhos como melhoria no *speedup* não apresentou resultados satisfatórios para sua utilização, até o momento, devido a biblioteca cuBlas só apresentar métodos de resolução de sistemas lineares com matrizes densas, inviabilizando sua utilização para casos grandes, além de ser a única biblioteca oferecida para suporte para o CDP em GPU pela NVIDA, desenvolvedora da Plataforma CUDA.

Nos testes o CDP obteve resultados positivos para 14 Barras com uma execução do Fluxo de Carga, possibilitando que seja utilizada futuramente apenas para casos pequenos, caso haja uma redução no tempo de alocação de memória nas arquiteturas de GPU futuras.

Os testes para sistemas de larga escala apresentaram bons resultados, mostrando a viabilidade de aplicação em sistemas reais, além de obter uma resposta mais rápida com a realização da paralelização em GPU.

Os códigos da estratégia com execução híbrida do fluxo de carga foram disponibilizados para acesso público através da plataforma GitHub de forma a facilitar a replicação deste trabalho no mercado ou em pesquisas em que os autores não são da área de computação paralela mas buscam essa área de forma a otimizar o tempo dos testes a serem realizados.

De acordo com o exposto, conclui-se que a melhor estratégia de paralelização do Fluxo de Carga com múltiplas execuções é realizar a etapa de resolução de sistema lineares paralelizada por *multithread* em CPU e as demais etapas em GPU.

Como trabalhos futuros pretende-se aplicar o algoritmo de metaheurísticas para otimizar os SEP's através dos equipamentos de controle de forma a analisar o comportamento de tais técnicas paralelizadas na mesma plataforma, bem como investigar a utilização de matrizes esparsas na arquitetura das GPU de maneira a melhorar sua eficiência, visto que a utilização destas em GPU é um grande desafio devido ao padrão de acesso a memória dessas estruturas não ser uniforme.

Adicionalmente, esta pesquisa visa a realização de testes com placas gráficas mais comuns como da família *GeForce*, encontrada mais comumente em placas gráficas da NVIDIA e verificar a viabilidade da execução deste trabalho em máquinas de maior

acessibilidade, em vez de maquinas próprias para o *High- performance computing* como no caso de placas da família *Tesla*, onde foram realizados nos testes apresentados.

Por fim, é válido ressaltar que houve 03 (três) trabalhos publicados em conferências: "Algoritmo de Otimização por Enxame de Partículas Paralelo para Minimização de Perdas de Potência Ativa em Sistemas Elétricos de Potência" In: XIV Workshop em Desempenho de Sistemas Computacionais e de Comunicação, 2015, Recife; "Comparative analysis of GPU Approaches for Power Flow Calculation in CUDA". In: IEE Japanese Annual Meeting, 2016, Sendai e "GPU Performance Evaluation for Concurrent Power Flow Calculations". In: IEE Japanese Annual Meeting, 2017, Toyama. Além de 01 (uma) publicação para período em aguardo de aceitação até o momento da conclusão desta dissertação: "Simultaneous Parallel Power Flow Calculations Using GPGPU". Electric Power Systems Research, 2017.

REFERÊNCIAS

AMANO, M.; ZECEVIC, A. I.; SILJAK, D. D. An improved block-parallel Newton method via epsilon decompositions for load-flow calculations. **IEEE Transactions on Power Systems**, v. 11, n. 3, p. 1519–1527, 1996.

BALDICK, R. et al. A fast distributed implementation of optimal power flow. **IEEE Transactions on Power Systems**, v. 14, n. 3, p. 858–864, 1999.

BARNEY, B. **Message Passing Interface (MPI)**. Disponível em: <https://computing.llnl.gov/tutorials/mpi/>. Acesso em: 23 jan. 2017.

BING LIANG et al. **A parallel algorithm of optimal power flow on Hadoop platform**. 2016 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC). **Anais**...IEEE, out. 2016Disponível em: <http://ieeexplore.ieee.org/document/7779568/>. Acesso em: 17 fev. 2017

BŁAŚKIEWICZ, P. et al. An application of GPU parallel computing to power flow calculation in HVDC networks. **Proceedings - 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015**, p. 635–641, 2015.

BORGES, C. L. T. **Análise de Sistemas de Potência**Rio de JaneiroEE-UFRJ, Departamento de Eletrotécnica, , 2005.

CHEN, H. M.; BERRY, F. C. Parallel load-flow algorithm using a decomposition method for space-based power systems. **IEEE Transactions on Aerospace and Electronic Systems**, v. 29, n. 3, p. 1024–1030, jul. 1993.

EL-KEIB, A. A.; DING, H.; MARATUKULAM, D. A parallel load flow algorithm. **Electric Power Systems Research**, v. 30, n. 3, p. 203–208, set. 1994.

FUKUYAMA, Y. Verification of Dependability on Parallel Particle Swarm Optimization Based Voltage and Reactive Power Control. **IFAC-PapersOnLine**, v. 48, n. 30, p. 167–172, 2015a.

FUKUYAMA, Y. Parallel particle swarm optimization for reactive power and voltage control verifying dependability. **IEEE Intelligent System Applications to Power Systems**, p. 3–9, 2015b.

GROSS, C. A. **Power system analysis**. [s.l.] Wiley, 1986.

GUO, C. et al. Performance comparisons of parallel power flow solvers on GPU

system. **Proceedings - 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2012 - 2nd Workshop on Cyber-Physical Systems, Networks, and Applications, CPSNA**, p. 232–239, 2012.

HENNESSY, J. L.; PATTERSON, D. A.; ASANOVIĆ, K. **Computer architecture : a quantitative approach**. [s.l.] Morgan Kaufmann, 2012.

HOUSOS, E. C.; OMAR WING. **Solution Of The Load Flow Problem By A Parallel Optimization Method**. IEEE Conference Proceedings Power Industry Computer Applications Conference, 1979. PICA-79. **Anais**...IEEE, 1979Disponível em: <http://ieeexplore.ieee.org/document/720085/>

HUANG, G.; ABUR, A. **A multilevel REI-based aggregation/disaggregation method for fast parallel power flow computation**. Proceedings of the 32nd Midwest Symposium on Circuits and Systems. **Anais**...IEEE, 1990Disponível em: <http://ieeexplore.ieee.org/document/101784/>

HUANG, G.; ONGSAKUL, W. Managing the bottlenecks in parallel Gauss-Seidel type algorithms for power flow analysis. **IEEE Transactions on Power Systems**, v. 9, n. 2, p. 677–684, maio 1994.

IEEE INDUSTRY APPLICATIONS SOCIETY. POWER SYSTEMS ENGINEERING COMMITTEE. **IEEE recommended practice for electric power distribution for industrial plants**. [s.l.] Institute of Electrical and Electronics Engineers, 1994.

IWATA, S.; FUKUYAMA, Y. Dependability verification of parallel differential evolutionary particle swarm optimization based voltage and reactive power control. **2016 IEEE International Conference on Power System Technology (POWERCON)**, p. 1–6, 2016.

J. TYLAVSKY, D.; BOSE, A. Parallel processing in power systems computation. **IEEE Transactions on Power Systems**, v. 7, n. 2, p. 629–638, maio 1992.

JUN QIANG WU; BOSE, A. Parallel solution of large sparse matrix equations and parallel power flow. **IEEE Transactions on Power Systems**, v. 10, n. 3, p. 1343–1349, 1995.

KIRK, D.; HWU, W. **Programming massively parallel processors : a hands-on approach**. 2nd. ed. [s.l.] Elsevier/Morgan Kaufmann, 2013.

KUNDUR, P.; BALU, N. J.; LAUBY, M. G. **Power system stability and control**. [s.l.] McGraw-Hill, 1994.

LAU, K.; TYLAVSKY, D. J.; BOSE, A. Coarse grain scheduling in parallel triangular factorization and solution of power system matrices. **IEEE Transactions on Power Systems**, v. 6, n. 2, p. 708–714, maio 1991.

LI, F.; BROADWATER, R. P. Distributed algorithms with theoretic scalability analysis of radial and looped load flows for power distribution systems. **Electric Power Systems Research**, v. 65, n. 2, p. 169–177, maio 2003.

LI, X. et al. GPU-based Fast Decoupled Power Flow with Preconditioned Iterative Solver and Inexact Newton Method. **IEEE Transactions on Power Systems**, p. 1–1, 2016.

LI, X.; LI, F. **GPU-based power flow analysis with Chebyshev preconditioner and conjugate gradient methodElectric Power Systems Research**, 2014.

LI, X.; LI, F.; CLARK, J. M. Exploration of multifrontal method with GPU in power flow computation. **IEEE Power and Energy Society General Meeting**, 2013.

MEI YANG et al. **An improved sparse matrix-vector multiplication kernel for solving modified equation in large scale power flow calculation on CUDA**. Proceedings of The 7th International Power Electronics and Motion Control Conference. **Anais**...IEEE, jun. 2012Disponível em: <http://ieeexplore.ieee.org/document/6259153/>

MONTEIRO, M. A. **Introdução ... organização de computadores**. [s.l.] Grupo Gen - LTC, 2007.

NARESH, M.; TRIPATHI, R. K. **Power flow control and power quality issues in distributed generation system**. 2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES). **Anais**...IEEE, jul. 2016Disponível em: <http://ieeexplore.ieee.org/document/7853558/>. Acesso em: 17 fev. 2017

NVIDIA. **Kepler {GK110} Whitepaper**NVIDIA, , 2012. Disponível em: <http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf>

NVIDIA CORPORATION. **CUDA Parallel Computing Platform**. Disponível em: <http://www.nvidia.com.br/object/cuda_home_new_br.html>. Acesso em: 30 dez. 2016a.

NVIDIA CORPORATION. **CUDA C Programming Guide**, 2016b. Disponível em:

<http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf>. Acesso em: 30 dez. 2016

RAFIAN, M.; STERLING, M. J. H.; IRVING, M. R. Decomposed load-flow algorithm suitable for parallel processor implementation. **IEE Proceedings C Generation, Transmission and Distribution**, v. 132, n. 6, p. 281, 1985.

RAKAI, L.; ROSEHART, W. GPU-Accelerated Solutions to Optimal Power Flow Problems. **System Sciences (HICSS), 2014 47th Hawaii International Conference on**, p. 2511–2516, 2014.

ROBERGE, V.; TARBOUCHI, M.; OKOU, F. Parallel Power Flow on Graphics Processing Units for Concurrent Evaluation of Many Networks. **IEEE Transactions on Smart Grid**, p. 1–10, 2015.

STOTT, B.; ALSAC, O. Fast Decoupled Load Flow. **IEEE Transactions on Power Apparatus and Systems**, v. PAS-93, n. 3, p. 859–869, maio 1974.

TAMIMI, B.; CANIZARES, C.; BATTISTELLI, C. Hybrid Power Flow Controller Steady-state Modeling, Control, and Practical Application. **IEEE Transactions on Power Systems**, p. 1–1, 2016.

UNIVERSITY OF WASHINGTON, E. E. **Power Systems Test Case Archive**. Disponível em: <http://www2.ee.washington.edu/research/pstca/index.html>. Acesso em: 19 jan. 2017.

WANG, F. Z.; HADJSAID, N.; SABONNADIÈRE, J. C. Power system parallel computation by a transputer network. **Electric Power Systems Research**, v. 52, n. 1, p. 1–7, out. 1999.

WANG, J.; YALAMANCHILI, S. Characterization and analysis of dynamic parallelism in unstructured GPU applications. **IISWC 2014 - IEEE International Symposium on Workload Characterization**, p. 51–60, 2014.

WANG, L. et al. Parallel reduced gradient optimal power flow solution. **Electric Power Systems Research**, v. 17, n. 3, p. 229–237, nov. 1989.

YE, C.-J.; HUANG, M.-X. Multi-Objective Optimal Power Flow Considering Transient Stability Based on Parallel NSGA-II. **IEEE Transactions on Power Systems**, v. 30, n. 2, p. 857–866, mar. 2015.

ZHANG, C. et al. A Mixed Interval Power Flow Analysis under Rectangular and

Polar Coordinate System. **IEEE Transactions on Power Systems**, p. 1–1, 2016.

ZIMMERMAN, R. D.; MURILLO-SANCHEZ, C. E.; J, T. R. Matpower: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education. **Power Systems, IEEE Transactions**, v. 26, n. 1, p. 12–19, 2011.

ANEXO A – RESULTADO COMPLETO DOS TESTES

Nº Barras	Cálculos Simultâneos		Sequencial	Híbrido	CDP	Esparso	Denso
14	1	Total	0,115	1,528	1,512	1,658	1,527
		FC	0,114	0,018	0,008	0,141	0,243
	10	Total	0,175	1,533	1,519	1,671	1,538
		FC	0,174	0,019	0,018	0,154	0,253
	50	Total	0,221	1,564	1,587	1,707	1,573
		FC	0,220	0,052	0,082	0,187	0,291
	100	Total	0,275	1,616	1,688	1,750	1,625
		FC	0,274	0,104	0,179	0,232	0,342
	200	Total	0,338	1,696	1,823	1,841	1,726
		FC	0,337	0,185	0,312	0,325	0,443
30	1	Total	0,143	1,522	1,519	1,690	1,534
		FC	0,142	0,011	0,012	0,148	0,248
	10	Total	0,110	1,538	1,541	1,703	1,551
		FC	0,109	0,028	0,032	0,160	0,265
	50	Total	0,261	1,588	1,646	1,747	1,612
		FC	0,260	0,076	0,133	0,204	0,328
	100	Total	0,319	1,655	1,862	1,825	1,696
		FC	0,318	0,141	0,344	0,285	0,409
	200	Total	0,487	1,784	2,121	1,933	1,865
		FC	0,485	0,271	0,599	0,382	0,576
57	1	Total	0,155	1,527	1,539	1,695	1,545
		FC	0,154	0,016	0,027	0,150	0,265
	10	Total	0,198	1,547	1,589	1,719	1,582
		FC	0,197	0,035	0,075	0,175	0,297
	50	Total	0,262	1,652	1,861	1,832	1,726
		FC	0,260	0,137	0,337	0,289	0,442
	100	Total	0,516	1,767	2,299	1,983	1,909
		FC	0,514	0,254	0,776	0,431	0,628

Nº Barras	Cálculos Simultâneos		Sequencial	Híbrido	CDP	Esparso	Denso
	200	**Total**	0,832	2,025	3,010	2,269	2,289
		FC	0,830	0,507	1,477	0,719	1,002
118	1	**Total**	0,134	1,526	1,565	1,657	1,574
		FC	0,133	0,015	0,055	0,146	0,294
	10	**Total**	0,237	1,553	1,681	1,707	1,655
		FC	0,236	0,043	0,163	0,191	0,371
	50	**Total**	0,437	1,674	2,284	1,894	2,005
		FC	0,435	0,161	0,759	0,378	0,719
	100	**Total**	0,649	1,821	3,575	2,142	2,452
		FC	0,647	0,305	2,030	0,620	1,163
	200	**Total**	1,222	2,116	4,886	2,621	3,335
		FC	1,218	0,595	3,306	1,099	2,047
300	1	**Total**	0,179	1,532	2,039	1,805	1,720
		FC	0,178	0,025	0,523	0,261	0,441
	10	**Total**	0,384	1,633	3,177	2,825	2,114
		FC	0,383	0,126	1,644	1,282	0,832
	50	**Total**	1,252	2,047	10,098	7,392	3,925
		FC	1,249	0,539	8,479	5,847	2,643
	100	**Total**	2,481	2,579	19,111	13,135	6,196
		FC	2,476	1,069	17,383	11,588	4,911
	200	**Total**	4,666	3,665	37,292	24,600	10,764
		FC	4,658	2,156	35,347	23,025	9,473
1354	1	**Total**	0,232	1,576	25,774	8,176	2,619
		FC	0,230	0,073	24,199	6,663	1,340
	10	**Total**	1,047	1,897	77,642	66,682	7,187
		FC	1,043	0,390	75,647	65,164	5,904
	50	**Total**	4,661	3,283	--	326,734	--
		FC	4,646	1,761	--	325,204	--
	100	**Total**	9,346	5,040	--	653,597	--
		FC	9,317	3,501	--	652,053	--

Nº Barras	Cálculos Simultâneos		Sequencial	Híbrido	CDP	Esparso	Denso
	200	**Total**	17,665	8,473	--	1302,090	--
		FC	17,608	6,908	--	1300,516	--
2869	1	**Total**	0,390	1,655	337,381	96,044	6,158
		FC	0,388	0,155	335,534	94,533	4,879
	10	**Total**	3,029	2,690	--	946,386	--
		FC	3,022	1,182	--	944,864	--
	50	**Total**	14,253	7,373	--	4759,486	--
		FC	14,224	5,838	--	4757,928	--
	100	**Total**	28,268	13,147	--	9453,890	--
		FC	28,211	11,583	--	9452,303	--
	200	**Total**	57,314	24,749	--	18882,432	--
		FC	57,201	23,127	--	18880,810	--
9241	1	**Total**	1,150	1,967	--	4087,750	--
		FC	1,147	0,457	--	4084,929	--
	10	**Total**	9,982	5,815	--	39575,554	--
		FC	9,964	4,289	--	39574,008	--
	50	**Total**	49,895	22,972	--	84999,500	--
		FC	49,806	21,374	--	84998,150	--
	100	**Total**	99,411	44,161	--	392609,727	--
		FC	99,235	42,478	--	392606,541	--
	200	**Total**	197,247	86,460	--	801548,994	--
		FC	196,897	84,602	--	801547.128	--