3.
Implementação do Sistema

3.1.
Paralelização em CUDA (GPU-PGLIQ)

Aplicação: Aceleração Base

No algoritmo serial de PGLIQ, o maior tempo de processamento está na avaliação da função de aptidão, embora este procedimento seja de poucos indivíduos em cada geração. Outros processos como o operador *P* e o processo de observação do indivíduo quântico têm um custo computacional pequeno em comparação com o tempo de avaliação do programa. Entretanto, paralelizar apenas uma parte do algoritmo implica que se faça transferência de informação do GPU ao CPU e vice-versa, de forma que esta alternativa não é exatamente ótima.

Um dos objetivos desta dissertação é propor a paralelização de todos os processos da PGLIQ para problemas de regressão simbólica sem precisar de transferências de dados ou ocultar a latência destas. Assim, foi implementado um total de 13 Kernels CUDA, dos quais oito são os mais representativos e utilizados em cada geração. Na Figura 20, mostram-se os oito *kernels* que estão presentes em cada geração, Os blocos amarelos são os processos para o cálculo dos individuos clássicos. O *kernel GenerateRandomMatrix* é responsável da geração da matrix de números aleatórios que são utilizados no processo de observação; neste *kernel* são utilizadas as bibliotecas de CURAND. A matrix gerada é um dos parâmetros de entrada para o *kernel Observation* que inicializa os individuos quânticos criando os primeiros programas. Estes novos programas são paralelizados para executar todos os dados de entrada em cada *thread* no *kernell Evaluation.* Posteriormente, se faz a soma dos erros (soma os erros para cada dado de entrada) de cada individuos dentro do *kernel ReduceSum* e, finalmente, é realizado o cálculo de cumprimento dos programas com o *kernel CalcCumprimento*. Este procedimento é idêntico para os individuos observados. Realizado o cálculo dos erros dos individuos observados, todos os individuos são

ordenados em função de seus erros dentro do *kernel Sort*. Como último *kernel*, *P-Operator* realiza a atualização das probabilidades das Indivíduos Quânticos para a próxima geração. No caso que o algoritmo entre na etapa de *reset*, o *kernel Reset* executa a nova atualização dos Indivíduos Quânticos e salva o melhor indivíduo atingido. Existe um *kernel* de transferência *Device–Host* para transfererir o erro do melhor individuos atingido na presente geração ao CPU, onde é conferido se um novo individuo foi calculado ou o número de gerações sem melhorar foi atingido de modo a ordenar ao *kernel Reset* que realize a troca do novo melhor individuo com o melhor anterior (dentro do GPU). Ao mesmo tempo em que se inicia a comparação entre erros de cada individuo no CPU, uma nova geração começa a ser executada assincronamente na GPU, de modo que a nova geração e a comparação dos erros dos individuos clássicos com o melhor indivíduo são realizadas em paralelo.

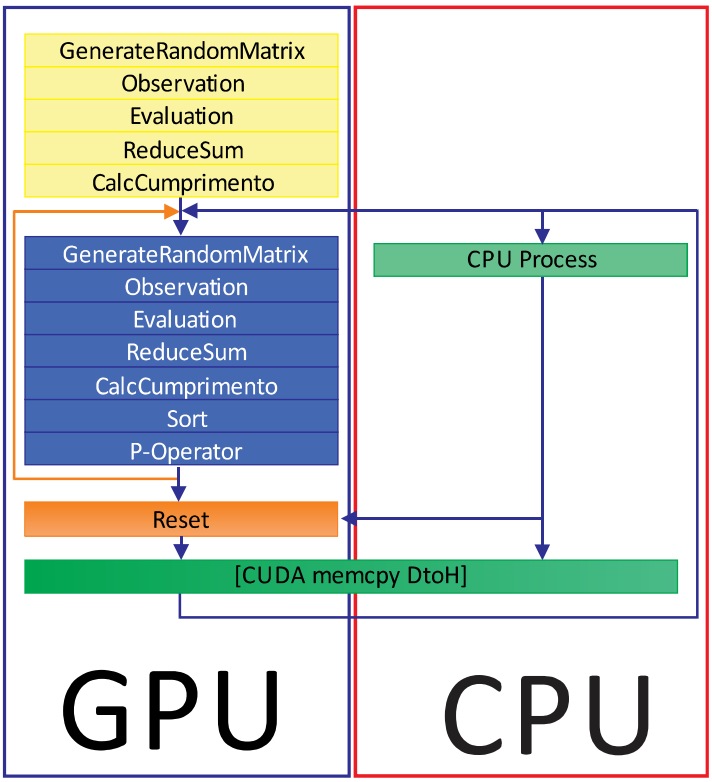

Figura 20. *Kernels* utilizados em cada geração pela GPU-PGLIQ

Pode-se dizer que uma das mudanças importantes é a execução do *kernel Reset* em função dos resultados da comparação com o melhor individuo na transferência

anterior. A mudança permite realizar cálculos no CPU, de modo que se o algoritmo encontra um novo melhor individuos, o valor de *gsm* vira 0 e o programa do melhor individuo ainda está com o primeiro dos individuos clássicos na memória do GPU, realizando a transferência do novo melhor indivíduo clássico como melhor resultado.

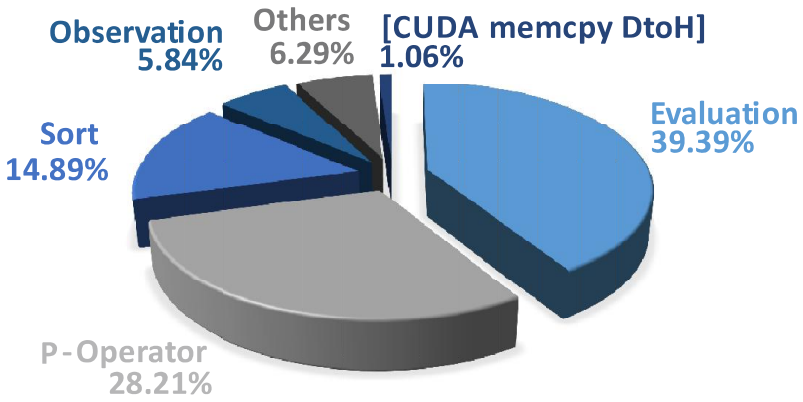

Figura 21. Porcentagem de tempo de processamento de cada *kernel* em uma geração da PGLIQ.

A instrução *cudamemcpy* é utilizada para transferir os erros da população ao *Host*. A porcentagem de tempo utilizada em uma geração é de 1.06%, como mostrado na Figura 21, juntamente com as porcentagens de tempo dos outros *kernels* executados para uma geração da PGLIQ. Se em cada geração o método *cudamemcpy* é chamado para levar a lista de aptidão dos indivíduos clássicos à CPU, temos uma perda de otimização, dado que uma GPU com capacidade de cálculo superior a 1.2 pode acessar a memória global realizando a leitura de blocos de dados de 8, 16 ou 32 palavras de 32 *bits* numa única operação [40] (quantidade maior à quantidade de individuos). A transferência à CPU não será feita em cada geração; primeiro armazena-se uma quantidade de dados que seja ótima para a transferência à CPU.

3.2. Paralelização com OpenMPI (GPU-PGLIQ)

Aplicação: *criação de demes*

A utilização de *demes* para melhorar o desempenho é comum na maioria de algoritmos evolutivos. No caso do modelo GPU-PGLIQ, é possível aproveitar o paralelismo em GPU e utilizar uma segunda placa gráfica para a segunda população. As duas populações têm que interatuar, de modo que o número de gerações ao longo de um intervalo de tempo em processamento seja menor comparado com o de uma só população.

Para um melhor desempenho, cada placa gráfica precisa de um processador para ser controlada. A cada intervalo de gerações, é realizada a transferência de indivíduos de uma população a outra. O esquema de transferência de Indivíduos Clássicos de uma população a outra se encaixa perfeitamente com o conceito de MPI. Assim, para um melhor desempenho, a utilização de uma placa gráfica por cada *deme* facilita a programação e deixa o trabalho da troca de mensagens para o MPI. A exemplificação da estrutura de *demes* trabalhando com MPI é mostrada na Figura 22.

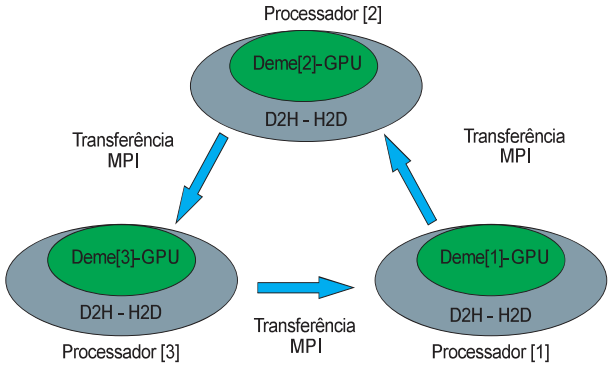

Figura 22. Estrutura da evolução com *demes* utlizando CUDA e OpenMPI

Na ilustração, D2H e H2D são utilizados para representar as transferências *Device to Host* e *Host to Device* respectivamente.

Entre os tipos de comunicação ponto a ponto, são utilizadas as operações não-bloqueantes, dado que cada processador tem que enviar e receber os indivíduos sem esperar receber uma confirmação. Além disso, as operações de envio e recebimento retornam quase que imediatamente, não esperando pelo término de qualquer evento de comunicação, de forma que podem se sobrepor computação e comunicação para

obteção de alguns ganhos de desempenho. A transferência entre populações é realizada enviando o melhor indivíduo da população "*n*" à população "*n+1*", e da última população à primeira população, fechando o círculo de *demes*.

3.3. Paralelização com OpenMP (GPU-PGLIQ)

Aplicação: Problemas multivariáveis

A PGLIQ pode resolver tantos problemas univariáveis como multivariáveis, mas, quando aumenta-se o número de variáveis, a solução geralmente é mais complexa e pode-se precisar de um programa com mais linhas de código que gere uma solução aceitável. Uma técnica alternativa é a utilização de vários programas trabalhando em conjunto, encarregados de otimizar uma parte de todo o problema.

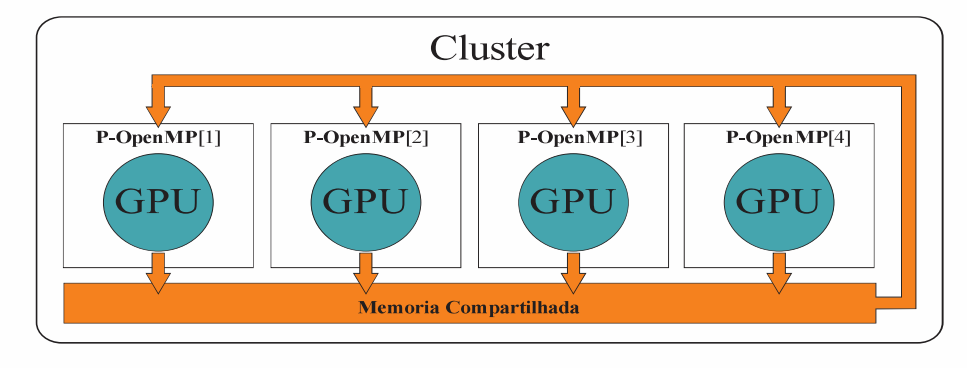

Figura 23. Topologia Multi-GPUs com memória compartilhada no *Host*.

A estrutura mostrada na Figura 23 mostra um sistema onde quatro GPUs trabalham em paralelo gerando quatro programas. A saída é transferida à memória compartilhada (*Host*), onde é avaliada e retorna os valores de aptidão ao GPU para continuar a iteração seguinte. No momento da avaliação, podem-se usar os recursos de OpenMP para melhorar o desempenho. Porém, a utilização de OpenMP não é só para problemas multivariáveis. Um dos objetivos desta dissertação é a visualização remota da evolução que está sendo processada dentro do *cluster*, mas é necessário trabalhar com ajuda de outra ferramenta como OpenMPI criando um sistema hibrido MPI /openMP

3.3.1.
Paralelização Hibrida MPI /OpenMP (GPU-PGLIQ)

Aplicação: Visualização Gráfica

Os sistemas híbridos permitem paralelizar diferentes processos simultaneamente, aproveitando os beneficios de ter uma memória compartilhada entre processadores, ao mesmo tempo em que isola outros processadores, comunicando-se só por troca de mensagens. Nesta dissertação, estes tipos de paralelizações são utilizados para levar os resultados do *cluster* ao um computador remoto e visualizar como é desenvolvido o processo evolutivo.

O sistema que gerencia o desenvolvimento do processo evolutivo e a transferência de informação entre o computador local e o servidor precisa executar diferentes tarefas em paralelo, entre eles um processamento dedicado apenas para recepção de dados trabalhando sincronamente com os processadores que executam a PGLIQ e controlam as placas gráficas. Estes processadores têm que se comunicar com o processador dedicado exclusivamente à comunicação TCP/IP que envia informação a um computador remoto. Para facilitar esta tarefa, oprocessador principal, de recepção de dados e de comunicação TCP/IP são de memória compartilhada. Dado que só um processador escreve e o outro lê, não é necessário colocar barreira alguma.

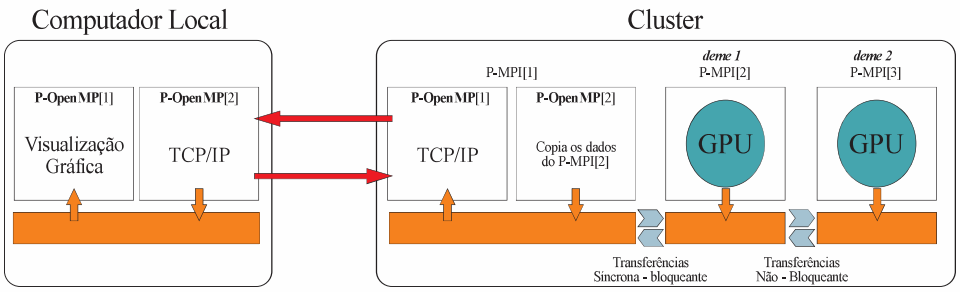

Figura 24. Estrutura Base MPI/OpenMP

A Figura 24 mostra o primeiro esquema deste sistema, conforme explicado anteriormente. Os processadores P-MPI[2] e P-MPI[3] evoluem os *demes* 1 e 2, os quais trocam indivíduos cada intervalo de gerações. O processador P-OpenMP[2] é encarregado de copiar os resultados em cada intervalo que seja programado (não precisa ser cada iteração), e o processador P-OpenMP[1] pega os dados e envia os resultados do *cluster* ao Computador Local.

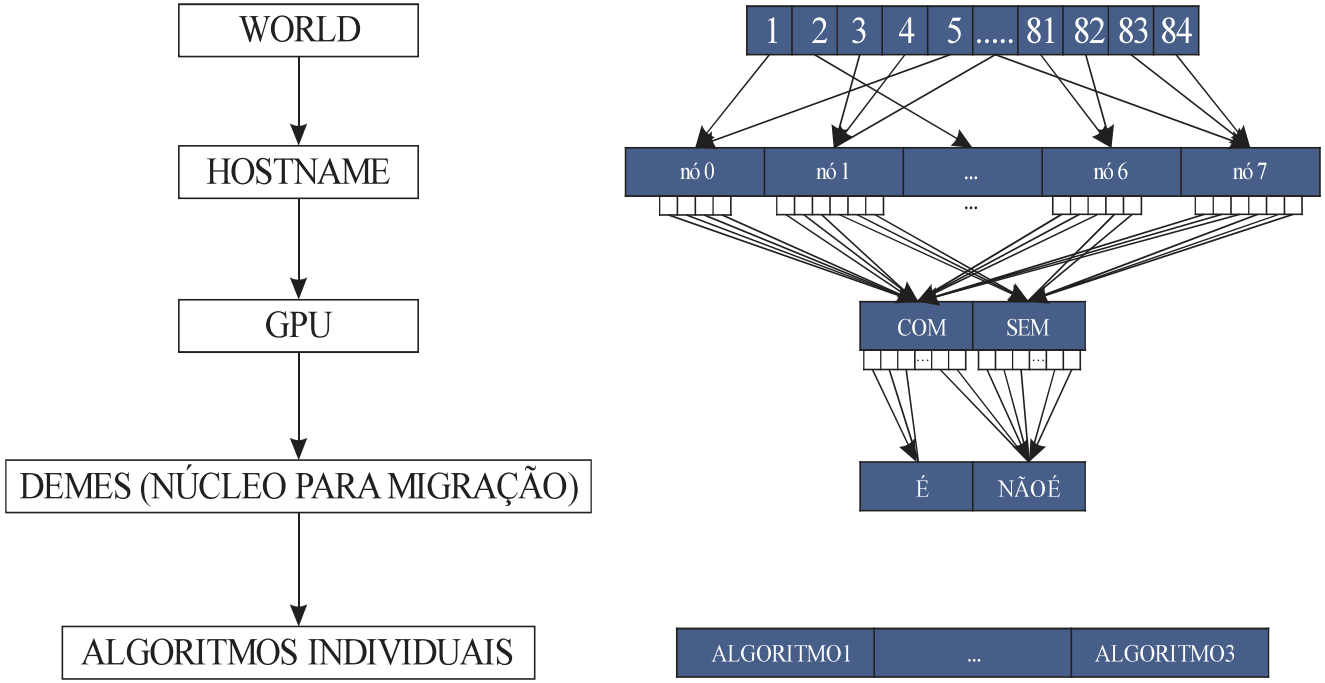

Figura 25. Agrupamentos dos nós dos *clusters*.

Para análise do estudo de caso de robótica aplicada, necessita-se de um poder computacional muito maior, tendo em vista o custo computacional utilizado nas simulações. Assim, foi criada uma topologia MPI a qual apresenta diferentes tipos de agrupamentos entre os processadores. Nesta topologia, pode-se implementar qualquer tipo de algoritmo evolutivo.

A Figura 15 apresenta a topologia implementada nos *clusters* do Departamento de Engenharia Elétrica – ICA. O *cluster* utiliza o gerenciador de recursos Torque PBS, o qual, até o momento, não apresenta suporte para o controle de múltiplos GPUS NVIDIA. Por isso, foi criada uma topologia especial para obter o controle de todas as placas gráficas e paralelizar algoritmos evolutivos.

A Figura 15 apresenta o agrupamento dos nós do *cluster* com a finalidade de obter uma topologia com as características explicadas acima. A topologia apresenta vários níveis de agrupamento os quais possuem as seguintes características:

- Primeiro Nível: Cada agrupamento de algoritmo tem um processador principal (nó 0 dentro do agrupamento), o qual tem o controle de um GPU, além de ter processadores livres que paralelizam as avaliações dos indivíduos. Cada agrupamento é isolado, permitindo assim as operações MPI_Bcast e MPI_Gather entre o processador principal e os processadores livres.
- Segundo Nível: Todos os processadores principais estão agrupados no nível *demes*, o qual tem duas finalidades: ajudar nos processos de migração utilzado por *demes*; e obter os valores de cada *deme* em um processador central (nó 0), o qual tem um processador adicional com memória compartilhada para comunicação e transmissão de resultados em tempo real.
- Terceiro e Quarto Níveis: Estes níveis foram programados pela falta de suporte para o controle de placas gráficas GPU NVIDIA pelo TORQUE. O agrupamento *Hostname* agrupa os processadores pelo nome de computador ao qual pertencem, e o agrupamento GPU separa uma quantidade de processadores igual à quantidade de placas gráficas que possui o computador. Assim, o segundo nível pode obter os processadores para criar seu agrupamento.
- Quinto Nível: WORLD contém todos os processadores numerados sem utilizar nenhuma estrutura padrão.

Os agrupamentos principais são dois níveis (migração e *demes*), os quais são esquematizados na Figura 16 para melhor entendimento. Cada coluna representa um agrupamento de *demes* que evolui isoladamente interatuando entre nós principais dentro do grupo PGLIQ. O nó principal (no extremo superior esquerdo da Figura 16) contém dois processos openMP para a transferência de dados (resultado dos parâmetros do algoritmo) a um ponto remoto. As razões pela que a GPU-PGLIQ-1 utiliza uma estrutura MPI para o caso de estudo de robótica evolutiva são as seguintes:

- A etapa de migração descreve um processo de transferência de informação (indivíduos) de uma população (*deme*) a outra. As bibliotecas de troca de mensagens facilitam este tipo de processos.
- Quando se trabalha em nível de *clusters* (agrupamento de computadores), a utilização de bibliotecas de memória compartilhada como OpenMP são uma forma ineficiente de paralelizar, dado que a programação é sobre uma plataforma que transforma as memórias de cada computador, que estão separadas fisicamente, em uma unica memória. Entretanto, OpenMP tem melhor desempenho quando trabalhamos com processadores de um mesmo computador.
- O funcionamento do simulador *Bullet* apresenta erros quando executamos varias simulações em paralelo com memória compartilhada.

Dentro do algoritmo de GPU-PGLIQ-1, há um processo OpenMP que recebe os resultados de todos os *demes* e envia para o computador local. Tendo em vista que o trabalho do processo é leve e não síncrono com o algoritmo evolutivo, a maior parte do tempo está em estado *idle* (ocioso), aguardando uma conexão com o computador remoto.

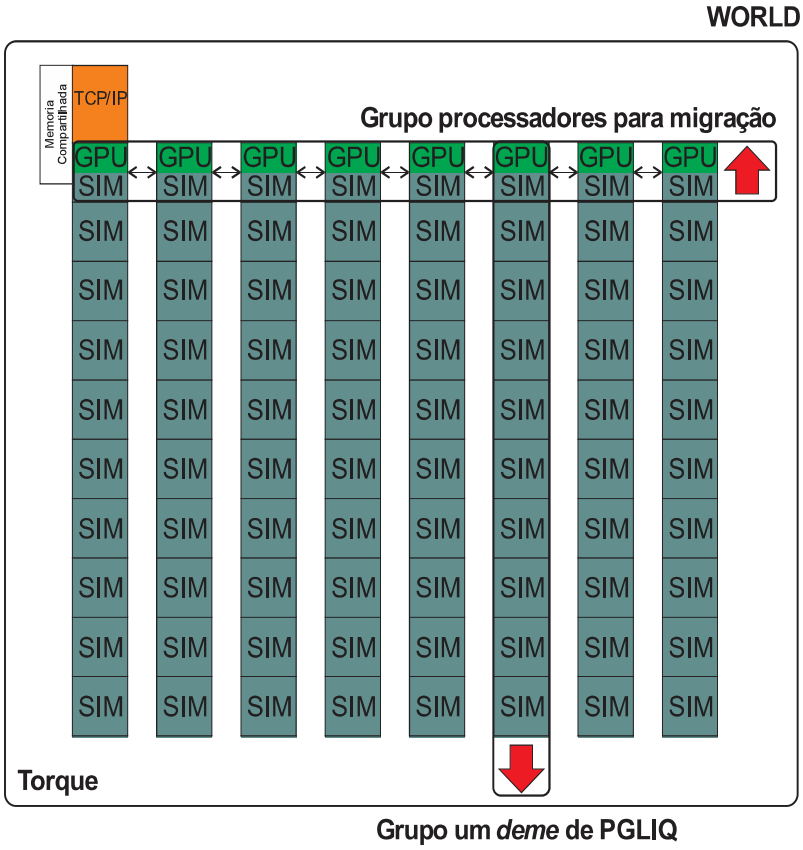

Figura 26. Topologia MPI para Algoritmo Evolutivos

3.4. Visualização Gráfica

A utilização da Visualização Gráfica como ferramenta de apoio, além de fornecer uma forma de compreender o que acontece dentro do processo evolutivo, também ajuda a visualizar novas formas de melhorar o desempenho do algoritmo. Desta forma, nesta seção, são definidas duas visualizações particulares à PGLIQ com a finalidade de melhorar o desempenho. Por outro lado, um estudo de caso desta dissertação é a aplicação da PGLIQ em robótica evolucionaria que, aproveitando a estrutura de comunicação do *cluster* ao computador local, visualiza a comportamento de um robô hexápode que está sendo evoluído dentro do *cluster*.

3.4.1.
Visualização do Melhor Individuo Quântico

Na PGLIQ, podemos observar que o conhecimento que o algoritmo aprendeu em cada geração é armazenado no Indivíduo Quântico cada vez que o Operador *P* é executado. Portanto, é uma variável importante para visualizar. Como definição, o Individuo Quântico é a superposição das "*N*" funções, as quais são possíveis soluções, cada uma com uma probabilidade de ser observada. Então, é possível visualizar esta variável como um plano de "*N*" funções x "*L*" linhas de código que tenha o programa.

Esta visualização permite observar o comportamento do indivíduo quântico através das gerações e, especialmente, quando chega a etapa de *Reset*, onde a PGLIQ apresenta dificuldades de encontrar uma técnica que permita sair do mínimo local eficientemente. Posteriormente, na etapa de Análise da PGLIQ, esta visualização será utilizada como critério da melhora de desempenho proposta.

3.4.2.
Visualização da estrutura de Sub-rotinas

Para a construção do aproximador, os dados foram obtidos usando os próprios simuladores que se quer substituir. Como o custo computacional da utilização dos simuladores integrados é alto, uma técnica que visa minimizar a quantidade de amostras é altamente recomendada.

3.4.3.
Simulação e Animação

Nesta seção, descrevem-se as ferramentas utilizadas tanto para simular e visualizar a dinâmica do ambiente e os resultados do estudo do caso 3. A visualização gráfica é feita na plataforma Linux e utiliza-se suporte da biblioteca *Bullet*, simulando dentro do *cluster* para avaliar os indivíduos em cada iteração, e em um computador local para interaturar com motor gráfico OGRE (*Object-Oriented Graphics Rendering Engine*), usado para o desenvolvimento de simulações em tempo real, como os jogos, tanto para *videogames* ou computadores e encarregado da renderização, visualizando o processo de otimização que está

aconteciendo dentro do *cluster*. Estas ferramentas apresentam diversas funcionalidades e facilidades para programar em C/C + +.

3.5. Análise da PGLIQ

A PGLIQ [2] apresenta um desempenho geral superior na resolução de problemas de regressão simbólica em comparação com o modelo AIMGP, considerado atualmente o modelo de PG mais eficiente na evolução de código de máquina. Mesmo assim existem etapas da PGLIQ nas quais é possível melhorar o desempenho em que o algoritmo se desenvolve. Nesta seção, é apresentada uma extensão da PGLIQ a qual melhora o desempenho com que realiza algumas etapas do processo evolutivo na resolução de problemas. Os processos são estruturados na Figura 6 na seção 2.3.8 de fundamentação teórica.

A extensão da PGLIQ copia o melhor programa obtido antes do *Reset* e coloca uma função dentro do banco de memória da PGLIQ. Assim, o algoritmo, depois de pasar pela etapa de *Reset*, cria um novo programa que pode utilizar a função obtida para continuar evoluindo.

Esta extensão foi desenvolvida observando uma diminução no desempenho para sair do mínimo local após da etapa de *Reset*. A Visualização Gráfica do Indivíduo Quântico é utilizada para entender o que acontece internamente na reinicialização do *qudit* durante a execução do algoritmo. A Etapa de *Reset* da PGLIQ contém os seguintes processos:

- O operador *P* (seção 2.3.6) é aplicado uma vez em todos os indivíduos da população quântica, tomando como referência o melhor indivíduo clássico *Cm* e o melhor valor do tamanho de passo obtido experimentalmente ($s=sr=1,0$).
- O indivíduo *C1* da população clássica recebe uma cópia de *Cm* como semente.

Baseado nos testes realizados, a redução de desempenho acontece pela difilculdade de sair do “minimo local” em que se encontra a solução atual. Depois da estapa de *Reset*, os valores dos *qudits* são direccionados para o melhor indivíduo

encontrado até a última geração. Se na direção no espaco de busca em que os novos indivíduos foram reinicialzados não se encontra um novo melhor programa, a PGLIQ cairá novamente no mesmo mínimo.

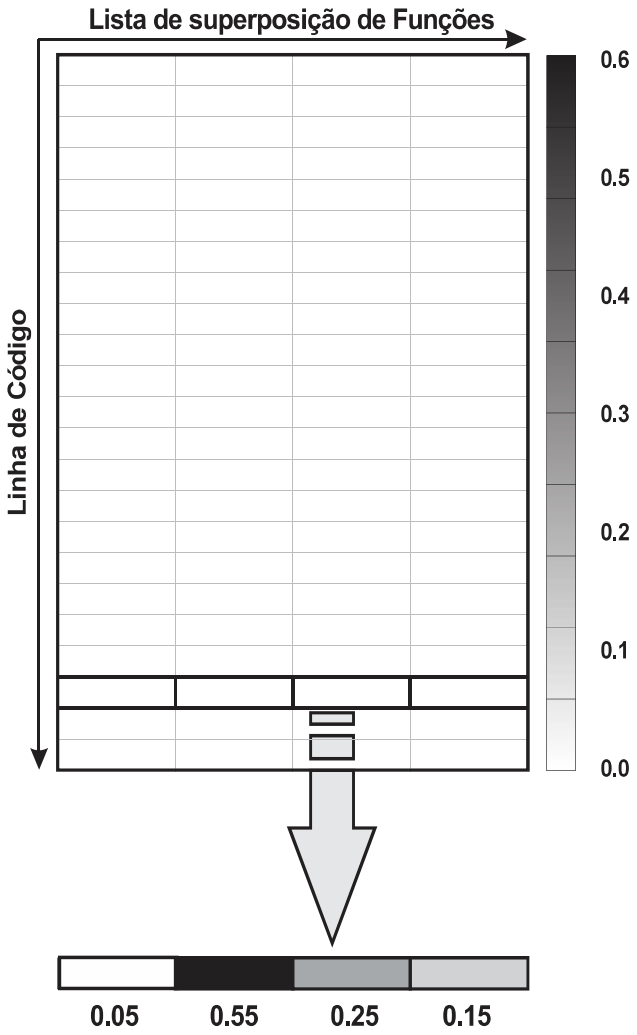

Figura 27. Distribução de probabilidades da PGLIQ

Para evitar a convergência prematura e sair do mínimo local eficientemente, precisa-se de um *reset* com um ponto de vista diferente. Várias alternativas foram avaliadas e a melhor proposta é apresentada nesta seção. Esta técnica tem uma nova etapa de *reset* que, além de fornecer um novo espaço de soluções, cria o conceito de função ou sub-rotina na PGLIQ, similar as ADFs nos modelos de Programação Genética com forma de árvore.

A Figura 27 apresenta a distribução de probabilidades do melhor Indivíduo Quântico. O gráfico é uma matriz de *Número de Linhas de Código x Número de*

Funções. Cada célula em uma linha representa a probabilidade da função de ser escolhida, de forma que a soma de todas as probabilidades em uma linha é igual a 1.

O valor da probabilidade de cada função ser escolhida é representado pela intensidade da cor de cada retângulo pequeno, em um contraste desde preto até branco. Uma coluna de cor preta está predominante quase sempre à esquerda no início do algoritmo, que é o operador NOP. Este operador inicializa com a probabilidade alta (normalmente 80% ou 90%), sendo o primeiro operador da lista as funções. As funções da lista são colocadas de esquerda para a direita. Depois, através das gerações, as probabilidades vão mudando para o melhor programa.

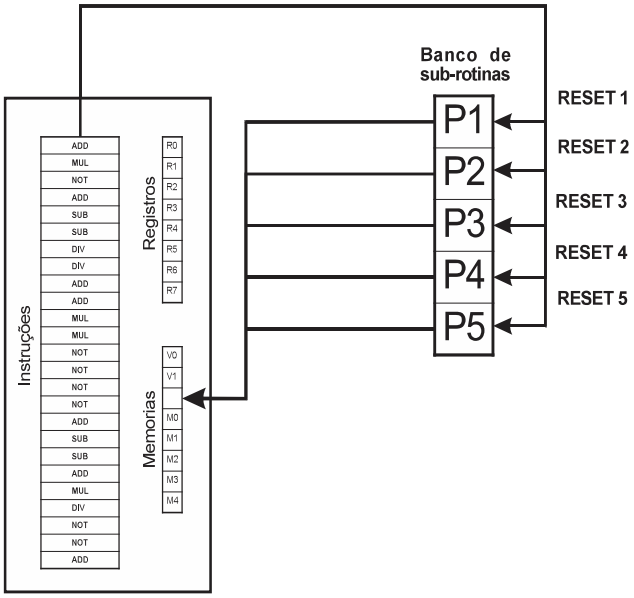

Figura 28. Estrutura nova etapa de *Reset*.

A técnica proposta é esquematizada no gráfico da Figura 28. Cada vez que a PGLIQ atinge a etapa de *Reset*, o melhor programa encontrado é armazenado em um banco de sub-rotinas, que depende estritamente das variáveis de entrada e constantes da memória. Esta sub-rotina é armazenada na memória do novo programa, que começará a evoluir. A Figura 29 mostra um caso típico em que a PGLIQ fica presa e atinge a etapa de *Reset*.

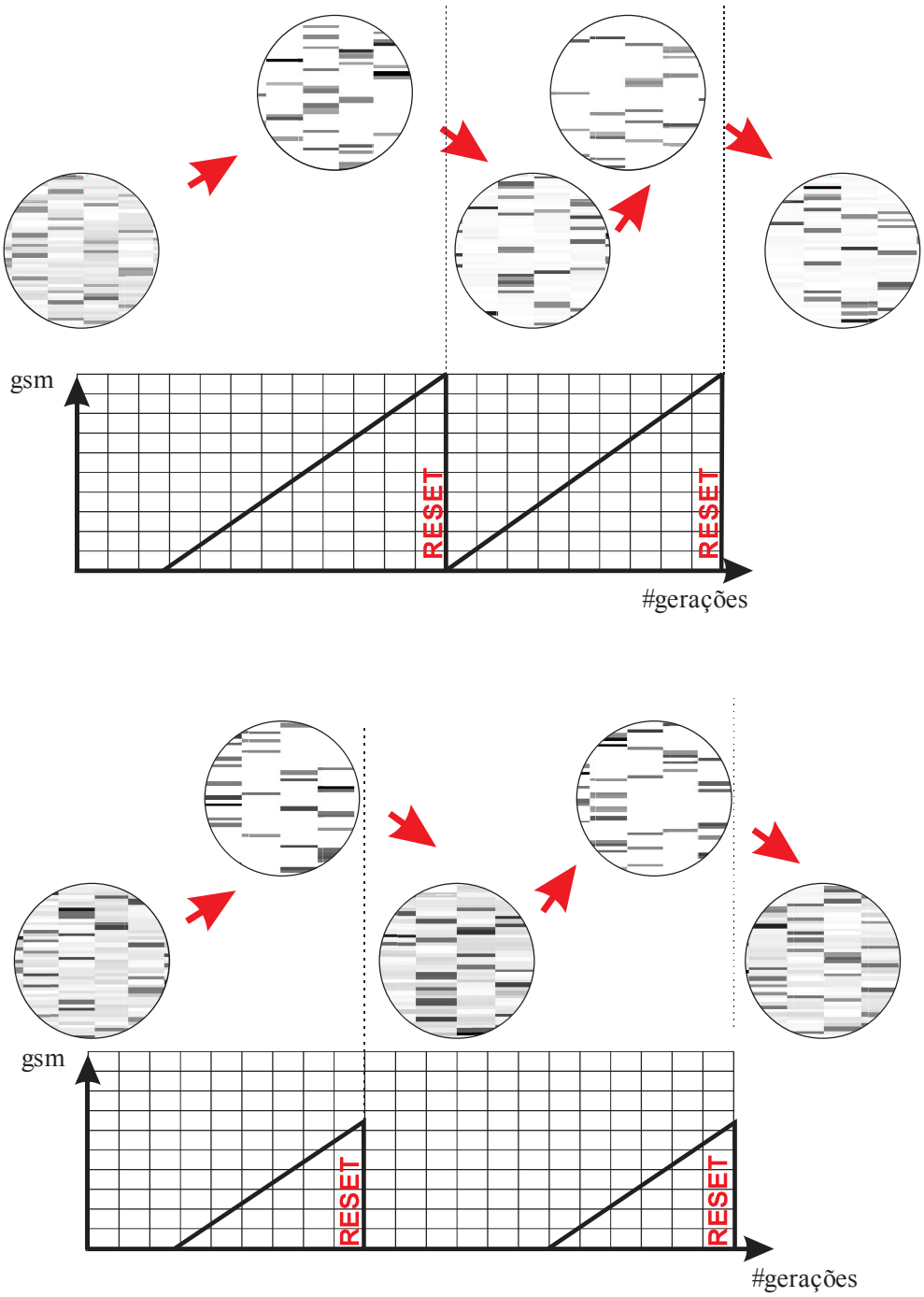

Figura 29. Etapas de Reset. (a) *Reset* padrão. (b) reset proposto.

No caso padrão (*caso a*), o algoritmo tenta sair do mínimo local, mas não consegue e fica preso novamente. Com a etapa de *reset* proposta (*caso b*), o algoritmo obtém o valor da sub-rotina previamente armazenada no banco de memórias e começa a evoluir saindo do mínimo local com maior facilidade. Assim,

a quantidade de gerações de transição até encontrar um novo melhor individuo é menor em comparação ao *reset* padrão explicado na seção 2.3. Contudo, o valor do parâmetro *gsm* (gerações sem melhorar) na PGLIQ padrão está em torno de 20,000 avaliações, dado que se precisa de uma quantidade alta de tentativas para sair do mínimo local que ele atingiu.

Um ponto importante de indicar é que, quando se utiliza a nova etapa de *reset*, o melhor valor obtido até o momento que está armazenado no banco de memórias tem um erro que se torna o novo problema (ou o novo espaço de soluções) na qual a PGLIQ começará a otimizar.

Como consequência, uma caraterística importante desta técnica é a diminuição do valor máximo de *gsm,* o qual é um parâmetro e condição de *reset* do algoritmo. Depois de atingir a etapa de *Reset*, a PGLIQ começa a evoluir e acessar a sub-rotinas armazenadas, mas acessa várias vezes, criando uma dependência entre o programa anterior e o programa atual. Esta dependência é representada em um diagrama, no qual cada seta indica um aceso à sub-rotina. Assim, é possivel saber a estrutura do programa depois da evolução. Na Figura 30, o "*programa_2"* acessa seis vezes o "*programa_1"* e "*programa"* accesa duas vezes o *"programa_2"*.

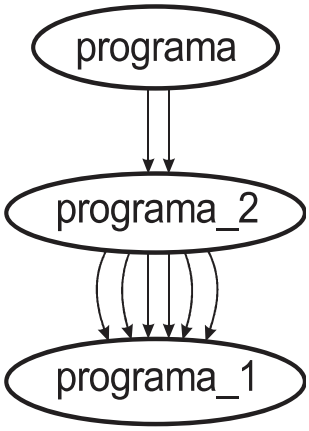

Figura 30. Estrutura de sub-rotinas da PGLIQ

Quando a técnica proposta trabalha com *demes,* o *reset* tem uma dependência de todos os *demes*, de forma que não é possível fazer o *reset* independentemente da evolução de outro *deme.* Tendo em conta uma transmissão

de indivíduos a cada 1000 gerações, a etapa de *reset* pode ser descrita em linhas gerais da seguinte forma:

- Em cada etapa de transmissão de individuos, o erro de cada *deme* é transmitido ao *deme* principal, onde é feita uma comparação entre todos os erros. Se eles foram iguais, o *flag1* é ativado.
- Se na segunda transmissão de indivíduos, os erros são novamente iguais (e iguais ao erro da transmissão anterior), o flag2 é ativado e começa o *Reset*.
- Todos os *demes* salvam o melhor programa (que é o mesmo em todos os *demes*) e é realizada uma reinicialização dos indivíduos quânticos.