Uma Ferramenta para Desenvolvimento de Aplicações SPMD com Suporte para Balanceamento de Carga

Alexandre Plastino
Departamento de Ciência da Computação
Universidade Federal Fluminense - Niterói - RJ
plastino@dcc.ic.uff.br

Celso C. Ribeiro
celso@inf.puc-rio.br

Noemi Rodriguez
noemi@inf.puc-rio.br

PUC-RioInf.MCC24/99
Novembro 1999

Abstract - We present in this work the tool SAMBA for parallel applications development. SAMBA (Single Application, Multiple Load Balancing) supports the development of SPMD applications, offering several load balancing strategies. The tool facilitates the identification and selection of appropriate load balancing algorithms for applications under development. We describe SAMBA's functionality and architecture, as well as its programming model. We also describe the algorithms implemented in the load balancing library.

Keywords: SPMD, Load Balancing, Parallel Applications.

Resumo - Apresenta-se a ferramenta SAMBA de apoio ao desenvolvimento de aplicações paralelas. SAMBA (Single Application, Multiple Load Balancing) auxilia a construção de aplicações SPMD, oferecendo diferentes opções de balanceamento de carga. A ferramenta facilita a identificação e a seleção de uma estratégia adequada à aplicação em desenvolvimento. Apresentam-se a arquitetura e a funcionalidade da ferramenta, seu modelo de programação e a descrição dos algortimos que compõem sua biblioteca de balanceamento de carga.

Palavras-chave: SPMD, Balanceamento de Carga, Aplicações Paralelas.

1 Introdução

Algoritmos paralelos são projetados segundo dois modelos de programação: paralelismo de função (ou de controle) e paralelismo de dados. No primeiro modelo, um programa paralelo é composto por um conjunto de tarefas diferentes, que implementam funções distintas, executadas concorrentemente nos diversos processadores. No modelo de paralelismo de dados, um programa paralelo é caracterizado por um mesmo processo executado em diversos processadores sobre diferentes conjuntos de dados. Desta forma, o que define uma tarefa neste modelo é o conjunto de dados sobre o qual o código comum deve ser executado.

O modelo de arquitetura MIMD (Multiple Instruction, Multiple Data), amplamente adotado no projeto de máquinas paralelas [Duc96, Mat96, Zom96], caracteriza-se por um conjunto de processadores que executam, de forma independente e assíncrona, seqüências de instruções (processos) diferentes sobre conjuntos de dados distintos. O modelo de paralelismo de dados, quando associado a arquiteturas MIMD, é conhecido como SPMD (Single Program, Multiple Data), no qual o mesmo programa é executado, sem sincronismo no nível de instrução, nos diferentes processadores, manipulando conjuntos distintos de dados (tarefas) [Qui94]. O modelo SPMD tem sido amplamente utilizado devido à facilidade de desenvolvimento e controle de um único código executado nos diversos processadores. Além disso, muitos problemas podem ser resolvidos por algoritmos paralelos baseados neste modelo [Mat96].

A redução do tempo de resposta de programas paralelos tem sido alvo de um grande esforço de pesquisa. O tempo de execução de um programa paralelo é definido pelo intervalo entre o instante em que o primeiro processador inicia a sua parte do processamento e o momento em que o último encerra a sua parte. A adequada distribuição de tarefas nos diversos processadores é fundamental para a minimização do tempo de execução do programa paralelo e a maximização da utilização dos recursos computacionais disponíveis. A identificação desta distribuição ideal de tarefas caracteriza o problema de **balanceamento de carga**.

Um algoritmo de balanceamento de carga de uma aplicação paralela tem como objetivo, então, distribuir entre os processadores as tarefas que compõem a aplicação, de forma a minimizar o seu tempo de execução. Para tanto, o algoritmo de balanceamento de carga deve controlar o efeito de diferentes fatores de desequilíbrio que ocorrem em diversas aplicações e arquiteturas, tais como: a heterogeneidade da arquitetura paralela adotada, o desconhecimento da quantidade de processamento envolvida em cada tarefa, a criação dinâmica de tarefas, a migração de tarefas, além da heterogeneidade e da variação da carga externa à aplicação nos diversos processadores de um ambiente paralelo não dedicado ([ZLP97]), adaptativo e não uniforme ([KR97]) ou instável ([AM96]).

No caso do modelo SPMD, a estratégia de balanceamento de carga, freqüentemente, aparece inserida na própria aplicação. Isto exige do projetista uma preocupação com a escolha da estratégia de balanceamento adequada. Em [AM96, dBS97, Eri96, FG96, FTI90, GZ97, KGGK94, KR97, LPR97, WLR93, ZLP97], apresentam-se algoritmos de balanceamento de carga utilizados em algumas aplicações. Cabe observar que, no caso de paralelismo de função, o escalonador de tarefas é o responsável por distribuir, de forma equilibrada, as várias tarefas que compõem a aplicação paralela entre os processadores, obedecendo às restrições de precedência existentes entre estas tarefas [CK88].

Se, por um lado, o modelo SPMD tem atraído a atenção de muitos projetistas e pesquisadores por oferecer uma estrutura mais simples e clara de programação paralela, por outro, exige um esforço na identificação do algoritmo de balanceamento de carga ideal à aplicação sendo desenvolvida.

Em [PRR98], propõe-se uma taxonomia de algoritmos de balanceamento de carga para aplicações SPMD. Esta taxonomia oferece uma terminologia padronizada e viabiliza uma classificação das muitas estratégias existentes, facilitando uma comparação mais criteriosa entre estas.

As contribuições apresentadas neste trabalho visam auxiliar o desenvolvimento de aplicações SPMD, com ênfase na tarefa de identificação de uma estratégia adequada de balanceamento de carga. A proposta central é a ferramenta SAMBA (Single Application, Multiple Load Balancing) de apoio ao desenvolvimento de aplicações SPMD, cujas principais funções são:

- Geração automática de aplicações SPMD com balanceamento de carga, a partir, basicamente, da codificação das rotinas responsáveis pela geração e execução das tarefas.
- Suporte ao projetista da aplicação SPMD na identificação de uma estratégia de balanceamento de carga adequada à aplicação em desenvolvimento. Diferentes versões da aplicação podem ser geradas, uma para cada estratégia existente na biblioteca de algoritmos de balanceamento de carga da ferramenta. Desta forma, o projetista pode testar e comparar diversas estratégias, para a mesma aplicação, sem custo algum de reprogramação.
- Facilitar o projeto e a avaliação de novos algoritmos de balanceamento de carga. Para tanto, a ferramenta dispõe de um conjunto de funções tipicamente utilizadas em algoritmos de balanceamento de carga. Além disso, permite a inclusão de novos algoritmos na biblioteca, o que viabiliza a comparação com os já existentes, utilizando uma base de aplicações anteriormente desenvolvida. O projeto da versão atual da biblioteca de balanceamento de carga foi guiado pela taxonomia proposta em [PRR98],

procurando-se variar as características e classes abordadas.

A proposta da arquitetura da ferramenta SAMBA foi validada por uma implementação utilizando-se a linguagem C e uma biblioteca MPI (Message Passing Interface - proposta de padronização das bibliotecas que implementam o modelo de troca de mensagens em ambientes com memória distribuída [MPI93, BS96, Fos94]). Esta implementação foi testada em dois ambientes: em um sistema paralelo IBM SP2, composto por 40 estações de trabalho RS/6000 ligadas por um *switch* de alta velocidade e em um *cluster* composto por 32 IBM PCs de 400MHz, executando Linux.

Este texto encontra-se organizado da seguinte forma. A Seção 2 apresenta um resumo da taxonomia proposta em [PRR98], cuja terminologia será adotada neste trabalho. A funcionalidade da ferramenta SAMBA é descrita na Seção 3. A Seção 4 define o modelo SAMBA de programação de aplicações SPMD. A biblioteca de algoritmos de balanceamento de carga da ferramenta é apresentada na Seção 5. O Apêndice A encerra este trabalho com detalhes operacionais sobre a ferramenta.

2 Resumo da Taxonomia de Algoritmos de Balanceamento de Carga para Aplicações SPMD

Nesta seção, apresenta-se um resumo da taxonomia proposta em [PRR98], que permite a classificação dos algoritmos de balanceamento de carga projetados para aplicações SPMD. Esta organização facilita a comparação mais criteriosa de estratégias, auxiliando projetistas na tarefa de escolher uma adequada à aplicação SPMD em desenvolvimento. A hierarquia da Figura 1 representa a taxonomia proposta.

Todo algoritmo de balanceamento de carga para aplicações SPMD possui uma **política de distribuição**. Esta se refere aos critérios que definem o particionamento dos dados entre os processadores, ou seja, critérios que definem de que forma e em que momento deve ser executada a distribuição das tarefas entre os processadores.

Estático - Um algoritmo estático define e realiza, antes da execução das tarefas da aplicação, a distribuição equilibrada destas tarefas.

Dinâmico - Em uma estratégia dinâmica, a alocação das tarefas aos processadores ocorre no início da sua execução, podendo ser dinamicamente alterada através da transferência de tarefas, de forma a controlar os efeitos dos fatores dinâmicos de desequilíbrio existentes. Em alguns casos, a própria distribuição das tarefas é feita dinamicamente ao longo da execução da aplicação.

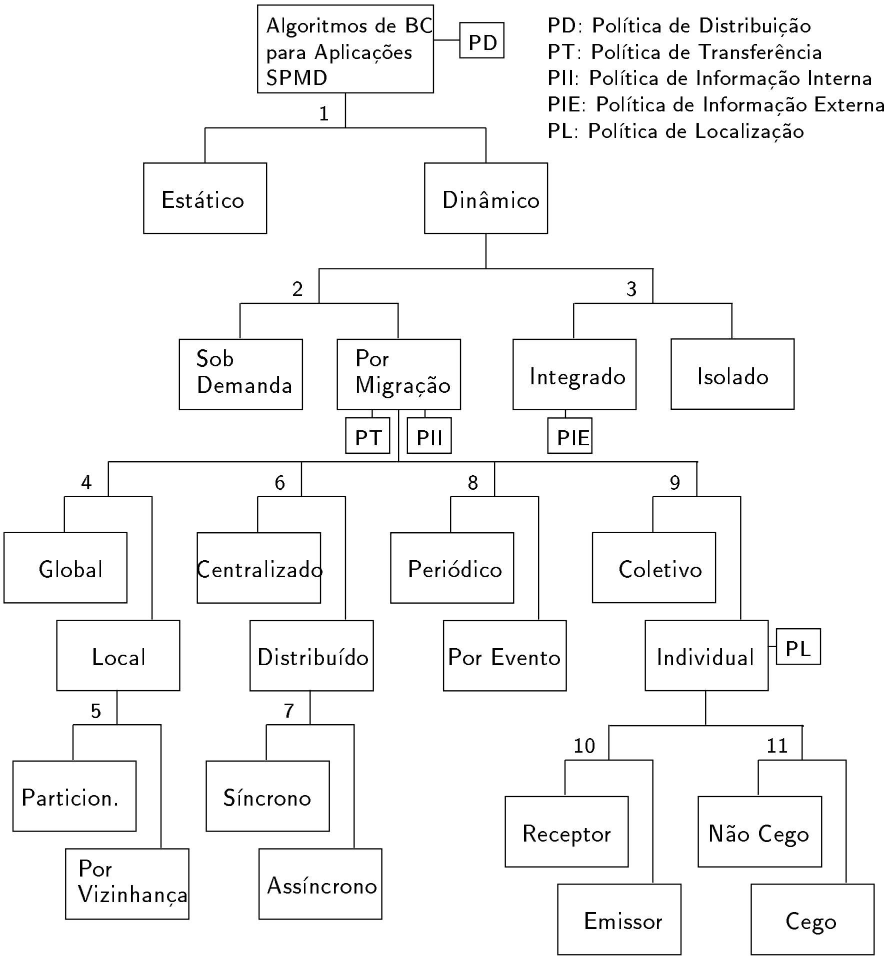

Figura 1: Taxonomia de algoritmos de balanceamento de carga para aplicações SPMD.

Sob Demanda - Nas estratégias dinâmicas sob demanda, a distribuição é feita ao longo da execução da aplicação, de acordo com o comportamento dos processadores. Inicialmente, um bloco de tarefas é distribuído, por um coordenador, para cada processador. Em seguida, cada processador recebe um novo bloco de tarefas do coordenador quando encerra a execução da parte anterior.

Por Migração - Em uma estratégia dinâmica por migração, realiza-se no início da execução da aplicação a distribuição equilibrada de todas as tarefas entre os processadores. Mais adiante, durante a aplicação, o balanceamento de carga é executado através da transferência de tarefas de processadores sobrecarregados para os subcarregados.

As estratégias por migração se caracterizam pela **política de transferência** adotada. Esta se refere ao critério utilizado na definição das transferências de carga necessárias ao balanceamento, ou seja, na definição do processador origem, do processador destino e da respectiva carga de cada transferência. Além desta, os algoritmos por migração são caracterizados por uma **política de informação interna**. Nesta, define-se o índice de carga interna a ser utilizado pela estratégia de balanceamento de carga. Este índice representa a carga de trabalho ainda a ser executada, relativa à aplicação, em cada processador. Em geral, este índice é definido pela quantidade de tarefas ainda não executadas em cada processador. A política de informação interna determina também como e com que freqüência são obtidos e atualizados os índices de carga interna nos diversos processadores.

Integrado e Isolado - A heterogeneidade e a variação da carga externa à aplicação nos diversos processadores, em ambientes não dedicados, são fatores dinâmicos de desequilíbrio que devem ser tratados pelos algoritmos de balanceamento de carga. As estratégias dinâmicas são também classificadas como integradas ou isoladas, quando, respectivamente, utilizam ou não índices que representam a carga externa à aplicação na definição das decisões que visam o equilíbrio da carga.

As estratégias integradas são caracterizadas por uma **política de carga externa**. Esta define o índice de carga externa a ser utilizado, além de estabelecer a forma e a freqüência de obtenção e atualização destes índices nos diversos processadores.

Global e Local - Em alguns algoritmos por migração, as transferências de carga podem ocorrer entre quaisquer processadores; em outras, apenas entre os que pertencem ao mesmo grupo, de acordo com algum critério de agrupamento previamente estabelecido. Definem-se, desta forma, respectivamente, os algoritmos globais e locais. As decisões de um algoritmo local são tomadas, separadamente, por grupo de processadores, considerando-se apenas as informações locais a cada grupo.

Particionado - Em um algoritmo local particionado, os grupos são definidos de forma que todo processador pertença a um único grupo.

Por Vizinhança - Em um algoritmo com localidade definida por vizinhança, os grupos são formados considerando-se a vizinhança entre os processadores na topologia física da rede de interconexão do ambiente paralelo ou na topologia lógica definida pela aplicação. Neste caso, um processador pode pertencer a mais de um grupo.

Centralizado - Em um algoritmo centralizado, um único processador define as transferências necessárias ao equilíbrio de carga e as informa aos processadores envolvidos para que estes as realizem.

Distribuído - Um algoritmo distribuído é executado localmente em cada processador. Para tanto, o seu código deve estar replicado em todos os processadores. Cada processador, isoladamente, define as transferências necessárias e providencia a realização daquelas nas quais está envolvido.

Síncrono - A execução de um algoritmo distribuído síncrono é caracterizada pela participação, ao mesmo tempo, de todos os processadores envolvidos no balanceamento de carga.

Assíncrono - A execução de um algoritmo assíncrono ocorre de maneira independente nos diversos processadores. Cada processador pode disparar a execução do algoritmo no momento em que for necessário, sem que haja sincronismo com a execução em outros processadores.

Periódica - Em uma estratégia periódica, o balanceamento de carga é ativado entre intervalos especificados. A periodicidade das ativações pode ser definida, por exemplo, pelo tempo decorrido ou pela quantidade de operações realizadas.

Por evento - Em uma estratégia por evento, o balanceamento é ativado pela ocorrência de alguma condição, como o fim das tarefas de algum processador ou a redução da carga de um processador a um nível inferior a um limite mínimo.

Coletivo - Em um algoritmo coletivo, o objetivo é balancear a carga de um conjunto não unitário de processadores do sistema (muitas vezes, quando se trata de um algoritmo global, o conjunto de todos os processadores do sistema).

Individual - Um algoritmo individual procura resolver, isoladamente, o problema de um processador que se apresenta subcarregado ou com carga em excesso.

A política de transferência dos algoritmos individuais se particulariza por conter uma **política de localização**, que se refere aos critérios utilizados na definição de quais processadores devem enviar ou receber tarefas do processador com problemas de carga.

Receptor - Nas estratégias individuais receptoras, identificam-se os processadores que devem enviar carga aos processadores subcarregados.

Emissor - Nas estratégias individuais emissoras, localizam-se os processadores para os quais se deve distribuir o excesso de carga de um processador sobrecarregado.

Cego e Não Cego - Em um algoritmo individual, a identificação dos processadores que devem enviar ou receber tarefas do processador com problemas de carga pode ser feita tomando-se por base informações sobre a carga dos processadores candidatos. Os algoritmos individuais cegos não utilizam estas informações, enquanto os não cegos as consideram. Em vários algoritmos cegos, os processadores envolvidos nas transferências são escolhidos aleatoriamente ou por rodízio (*round robin*).

3 A Ferramenta SAMBA

As aplicações SPMD podem ser caracterizadas por três componentes principais: o código de geração e execução das tarefas que compõem a aplicação, o algoritmo de balanceamento de carga e uma estrutura central que inicia e conclui a aplicação, além de ativar e controlar a execução das demais partes.

Considerando este princípio, a ferramenta SAMBA, ilustrada na Figura 2, foi projetada para gerar automaticamente aplicações paralelas SPMD com balanceamento de carga, bastando que o usuário escreva, basicamente, o código de geração de tarefas e o código comum que será executado sobre as diferentes partições de dados (tarefas SPMD). Para oferecer esta funcionalidade, a ferramenta dispõe da estrutura central citada e contém uma biblioteca de algoritmos de balanceamento de carga.

Além de facilitar o desenvolvimento de aplicações SPMD, a ferramenta permite que o usuário obtenha automaticamente, sem custo algum de reprogramação, versões diferentes da aplicação, uma para cada estratégia de balanceamento de carga disponível na respectiva biblioteca. O usuário pode assim testar e comparar diferentes estratégias e identificar as mais adequadas à sua aplicação.

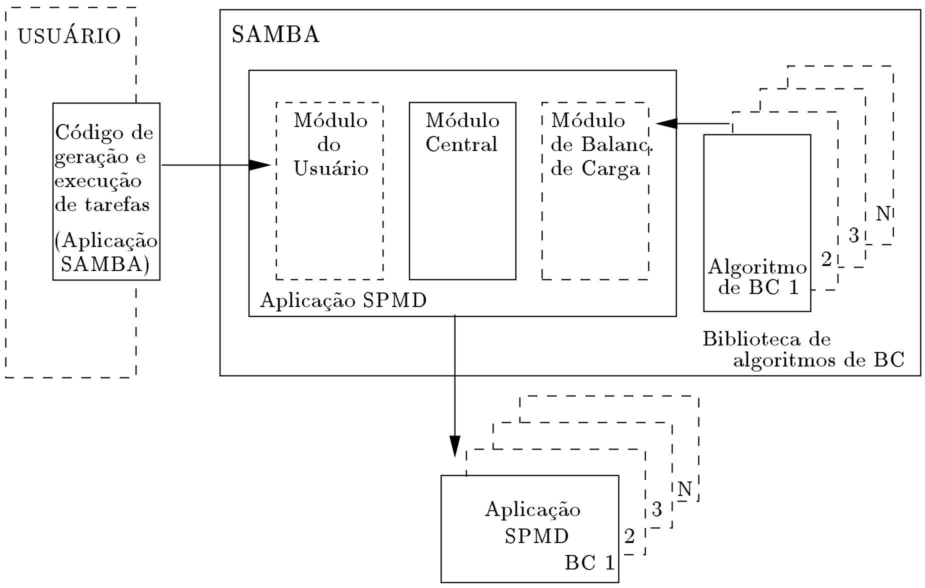

Figura 2: Funcionalidade da ferramenta SAMBA.

Toda aplicação projetada a partir desta ferramenta é composta por três módulos: o módulo do usuário (aplicação SAMBA), o módulo central e o módulo de balanceamento de carga.

O projetista de aplicações SPMD deve desenvolver a aplicação SAMBA segundo o modelo de programação SPMD próprio da ferramenta, definido na Seção 4, para garantir a correta integração entre o módulo do usuário e a ferramenta. As rotinas deste módulo são responsáveis por gerar as tarefas, executá-las e, dependendo da aplicação, tratar os seus resultados.

O módulo de balanceamento de carga pode ser preenchido por qualquer algoritmo da biblioteca. Estes algoritmos são responsáveis por controlar o efeito de diversos fatores de desequilíbrio existentes. Novos algoritmos podem ser facilmente desenvolvidos e inseridos na biblioteca, bastando para isso que o projetista deste tipo de estratégia conheça a estrutura deste módulo, apresentada na Seção 5. Esta flexibilidade facilita a avaliação de novas estratégias, pois viabiliza a comparação com as já existentes na biblioteca, utilizando-se uma base de aplicações SPMD anteriormente desenvolvidas.

O projeto da versão atual da biblioteca de balanceamento de carga, descrito na Seção 5, foi guiado pela taxonomia proposta em [PRR98], procurando-se variar as características e classes abordadas.

O módulo central, interno à ferramenta, é composto de rotinas responsáveis por iniciar a aplicação, ativar os demais módulos e concluir a aplicação. Além das rotinas que cumprem estes objetivos, possui outras projetadas para: gerenciar o conjunto de tarefas em cada processador, atender a pedidos de transferências de tarefas entre processadores, atender a pedidos de execução de tarefas ativando o módulo do usuário e, ainda, para reportar ao usuário erros que eventualmente ocorram.

O módulo central gerencia o reposiório de tarefas existente em cada processador. As rotinas que possuem esta função estão separadas em um submódulo, chamado gerente de tarefas. Desta forma, se uma aplicação necessitar de uma estrutura de dados diferente da oferecida pela ferramenta - uma lista -, este submódulo poderá ser reprogramado pelo usuário.

Cada tarefa é definida por uma área de memória na qual fica armazenado o conjunto de dados que a representa e pelo tamanho desta área. Além destes dois atributos, um adicional, do tipo inteiro, é reservado para alguma necessidade específica da aplicação.

A ferramenta, então, percebe uma tarefa apenas como uma área de memória que contém a partição de dados sobre o qual uma rotina, codificada pelo usuário, será executada. Esta mesma rotina será executada sobre todos os dados (*single program, multiple data*). O usuário é o responsável por organizar os dados que representam uma tarefa em uma área de memória contígua, ou seja, por gerar as tarefas. Estas áreas serão organizadas pela ferramenta no repositório de tarefas e entregues de volta ao usuário no momento da execução da tarefa.

Para os módulos central e de balanceamento de carga, não interessa o conteúdo destas áreas de memória que representam as tarefas. Foi exatamente este encapsulamento que tornou viável o projeto da ferramenta, cuja funcionalidade permite a troca do algoritmo de balanceamento de carga sem que a aplicação sofra uma só alteração de código. O usuário, basicamente, escreve o código de geração e execução das tarefas e obtém a aplicação SPMD com o algoritmo de balanceamento de carga escolhido.

A Figura 3 ilustra a integração entre os três módulos que compõem a ferramenta, apresentando o modelo de execução de uma aplicação SPMD gerada. A execução inicia-se no módulo central, com críticas e inicializações (Fase 1). Em seguida, é desviada para o módulo do usuário, no qual as tarefas são geradas e inseridas no repositório de tarefas (Fase 2). Logo após, o controle passa para o módulo de balanceamento de carga, que é responsável pela distribuição, controle da execução e eventual transferência das tarefas (Fase 3). Ao final da execução das tarefas, o controle retorna ao módulo do usuário para um possível tratamento dos resultados (Fase 4). Em seguida, o controle da execução volta ao módulo central que, após algumas finalizações, encerra a execução da aplicação SPMD gerada (Fase 5).

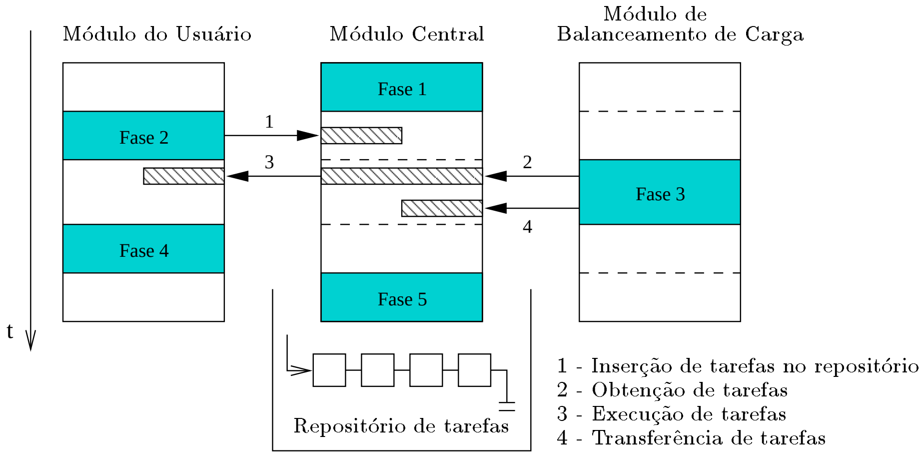

Figura 3: Modelo de execução de uma aplicação gerada pela ferramenta.

O módulo central, durante a Fase 2, é ativado pelo módulo do usuário para a inserção das tarefas no repositório. Na Fase 3, o módulo de balanceamento de carga utiliza rotinas oferecidas pelo módulo central para transferência de tarefas entre os processadores e para recuperação de tarefas do repositório para execução. A execução destas ocorre através da ativação do módulo do usuário.

O usuário deve codificar as rotinas da aplicação SAMBA (módulo do usuário) em C. Este código é compilado e integrado ao módulo central e ao algoritmo de balanceamento de carga selecionado da biblioteca por meio de um *makefile*. O resultado deste processo é um executável que representa a aplicação SPMD especificada pelo usuário com o algoritmo de balanceamento de carga escolhido. A utilização deste *makefile* é explicada no Apêndice A.

A seguir, o módulo central e o submódulo gerente de tarefas serão apresentados com mais detalhes. O módulo do usuário será detalhado na Seção 4, na qual o modelo SAMBA de programação de aplicações SPMD é definido. O módulo de balanceamento de carga e a respectiva biblioteca serão apresentados na Seção 5.

O Módulo Central

A seguir, com o objetivo de ilustrar a funcionalidade do módulo central, serão apresentadas as suas principais rotinas. Algumas são ativadas localmente, apenas por rotinas do próprio módulo, outras são projetadas para atender às necessidades tanto do projetista de aplicações SAMBA quanto do projetista de algoritmos de balanceamento de carga. Todas as rotinas do módulo central possuem o prefixo `C_`. A implementação deste módulo encontra-se nos arquivos `central.c` e `central.h`.

<u>`C_Inicial`</u>

Ativa o MPI em todos os processadores, obtém o identificador do processador em questão e o número total de processadores em execução. Além disso, separa dos parâmetros de ativação que são passados à aplicação no início de sua execução os parâmetros que devem ser enviados ao algoritmo de balanceamento de carga.

Na ativação de uma aplicação gerada pela ferramenta, são passados os parâmetros de entrada que serão utilizados pelo módulo do usuário e os que serão utilizados pelo módulo de balanceamento de carga. São exemplos dos dois tipos de parâmetros, respectivamente: o nome do arquivo no qual estão armazenados os dados que compõem as tarefas e o nome do arquivo que contém a definição dos grupos de processadores a serem considerados no balanceamento de carga.

Propõe-se que, na ativação da aplicação, após o nome do arquivo executável estejam os parâmetros do usuário e os parâmetros de balanceamento de carga separados por uma barra inclinada (/). Exemplo: `samba arquivo-dados / arquivo-grupos`

▷ Parâmetros de entrada:
- número total de parâmetros de ativação (`int*`),
- vetor contendo os parâmetros de ativação (`char* []`).

▷ Parâmetros de saída:

- identificador do processador em questão (`int*`),
- número de processadores em execução (`int*`),
- vetor contendo os parâmetros de balanceamento de carga (`char* []`),
- número de parâmetros de balanceamento de carga (`int*`).

▷ Rotinas ativadas:

- `MPI_Init`,
- `MPI_Comm_rank`,
- `MPI_Comm_size`,
- `C_Reporta_Erro`.

▷ Rotina que a ativa:

- esta rotina é ativada em todos os processadores pelo programa principal.

<u>`C_Final`</u>

Desativa o MPI em todos os processadores.

▷ Rotinas ativadas:

- `MPI_Finalize`,
- `C_Reporta_Erro`.

▷ Rotina que a ativa:

- esta rotina é ativada em todos os processadores pelo programa principal.

<u>`C_Envia_Protocolo`</u>

Envia uma mensagem a um processador (A) avisando que este deverá receber ou enviar um conjunto de tarefas a outro processador (B). A quantidade de tarefas em questão será informada por um parâmetro do tipo inteiro. Se o valor deste parâmetro for positivo (n), indica que A deverá receber n tarefas de B, se for negativo (-n), indica que A deverá enviar n tarefas a B.

▷ Parâmetros de entrada:

- identificador do processador A (`int`),
- identificador do processador B (`int`),
- quantidade de tarefas a serem transferidas (`int`),
- identificador do grupo de comunicação (`MPI_Comm`): este parâmetro faz-se necessário, pois, em alguns algoritmos de balanceamento de carga, vários grupos de comunicação podem ser criados e, portanto, a identificação de um processador passa a ser feita pelo seu número e pelo identificador do grupo de comunicação.

▷ Rotinas ativadas:
- `MPI_Send`,
- `C_Reporta_Erro`.

▷ Rotinas que a ativam:
- esta rotina foi projetada para ser utilizada por rotinas dos algoritmos de balanceamento de carga.

<u>`C_Recebe_Protocolo`</u>

Recebe uma mensagem de um processador (A) informando que outro processador (B) deverá receber ou enviar um conjunto de tarefas ao processador executante (esta expressão será utilizada para designar o processador que executa a rotina em questão). Mantém-se a convenção adotada para o sinal do parâmetro que indica a quantidade de tarefas a ser transferida. Portanto, se o valor deste parâmetro for positivo (n), indica que o processador corrente deverá receber n tarefas de B, se for negativo (-n), indica que o corrente deverá enviar n tarefas a B.

▷ Parâmetros de entrada:
- identificador do processador A (`int`),
- identificador do processador B (`int*`),
- quantidade de tarefas a serem transferidas (`int*`),
- identificador do grupo de comunicação (`MPI_Comm`).

▷ Rotinas ativadas:
- `MPI_Recv`,
- `C_Reporta_Erro`.

▷ Rotinas que a ativam:
- esta rotina foi projetada para ser utilizada por rotinas dos algoritmos de balanceamento de carga.

<u>`C_Envia_Tarefas`</u>

Retira um conjunto de tarefas do repositório do processador executante e as envia a um outro determinado processador.

▷ Parâmetros de entrada:
- quantidade de tarefas a serem enviadas (`int`),
- identificador do processador destino (`int`),
- identificador do grupo de comunicação (`MPI_Comm`).

▷ Rotinas ativadas:
- `GR_Obtem_Tarefa`,
- `MPI_Send`,
- `GR_Libera_Area`,
- `C_Reporta_Erro`.

▷ Rotinas que a ativam:
- esta rotina foi projetada para ser utilizada por rotinas dos algoritmos de balanceamento de carga.

<u>`C_Recebe_Tarefas`</u>

Recebe um conjunto de tarefas de um determinado processador e as insere no repositório de tarefas do processador executante.

▷ Parâmetros de entrada:
- quantidade de tarefas a serem recebidas (`int`),
- identificador do processador origem (`int`),
- identificador do grupo de comunicação (`MPI_Comm`).

▷ Rotinas ativadas:
- `MPI_Probe`,
- `MPI_Get_count`,
- `MPI_Recv`,
- `GR_Insere_Tarefa`,
- `C_Reporta_Erro`.

▷ Rotinas que a ativam:
- esta rotina foi projetada para ser utilizada por rotinas dos algoritmos de balanceamento de carga.

<u>`C_Executa_Tarefas`</u>

Retira um conjunto de tarefas do repositório do processador executante e providencia a execução destas, ativando a rotina do módulo do usuário responsável pela execução de tarefas.

▷ Parâmetros de entrada:
- quantidade de tarefas a serem executadas (`int`).

▷ Rotinas ativadas:
- `GR_Obtem_Tarefa`,
- `U_Executa_Tarefa`,
- `GR_Libera_Area`.

▷ Rotinas que a ativam:
- esta rotina foi projetada para ser utilizada por rotinas dos algoritmos de balanceamento de carga.

<u>`C_Reporta_Erro`</u>

Reporta ao usuário algum erro ocorrido durante a execução da aplicação, encerrando-a em seguida.

▷ Parâmetros de entrada:
- código do erro ocorrido (`int`),
- descrição do erro ocorrido (`char*`).

▷ Rotinas ativadas:
- `MPI_Abort`.

▷ Rotinas que a ativam:
- `C_Inicial`,
- `C_Final`,
- `C_Envia_Protocolo`,
- `C_Recebe_Protocolo`,
- `C_Envia_Tarefas`,
- `C_Recebe_Tarefas`,
- esta rotina foi projetada também para ser utilizada por rotinas do módulo do usuário e por rotinas dos algoritmos de balanceamento de carga. Neste caso, a constante `C_ERRO` deve ser informada como código do erro e uma cadeia de caracteres deve ser informada para descrever o erro ocorrido.

<u>`Programa Principal`</u>

O programa principal é ativado no início da execução da aplicação em todos os processadores. Controla a execução da aplicação. Primeiramente, ativa as rotinas que fazem as críticas iniciais em cada módulo: `C_Inicial`, `U_Inicial` e `BC_Inicial`. Em seguida, ativa, simultaneamente, as rotinas do usuário projetadas para a geração de tarefas: `U_Inicial_Mestre` - ativada em um processador eleito mestre - e `U_Inicial_Escravo` - ativada nos demais processadores. Esta distinção entre mestre e escravos permite que a aplicação tenha algum processamento diferenciado em um processador central, como, por exemplo, a leitura de um arquivo centralizado que contém os dados que representam as tarefas. Caso não haja nenhum processamento centralizado, estas rotinas poderão ter códigos idênticos. Estas rotinas devem utilizar o módulo gerente de tarefas para a inserção de tarefas no repositório. O módulo de balanceamento de carga é então ativado, em todos os processadores, por meio das rotinas `BC_Polit_Distrib` e `BC_Polit_Transf`, para controlar a execução das tarefas de forma balanceada. Após a execução das tarefas, o controle volta ao módulo do usuário para eventual tratamento de resultados. Para esta função, também rotinas distintas são ativadas no mestre e nos escravos: `U_Final_Mestre` e `U_Final_Escravo`, prevendo uma possível necessidade de um tratamento centralizado dos resultados. A execução da aplicação se encerra após a ativação das rotinas de finalização de cada módulo: `BC_Final`, `U_Final` e `C_Final`.

O programa principal da aplicação reside, portanto, no módulo central. Os módulos do usuário e de balanceamento de carga possuem, então, rotinas que serão direta ou indiretamente ativadas por este programa principal. No Apêndice A, apresenta-se uma alternativa para esta estrutura, na qual o programa principal encontra-se no módulo do usuário. Esta outra forma facilita a integração da ferramenta com aplicações SPMD já desenvolvidas, visando a utilização dos algoritmos da biblioteca de balanceamento de carga. Além disso, auxilia o desenvolvimento de aplicações SPMD iterativas e de aplicações paralelas do qual apenas parte do código é SPMD.

▷ Parâmetros de entrada:
- número de parâmetros de ativação (`int`),
- vetor contendo os parâmetros de ativação (`char* []`).

▷ Rotinas ativadas:
- `C_Inicial`,
- `U_Inicial`,
- `BC_Inicial`,
- `U_Inicial_Mestre`,
- `U_Inicial_Escravo`,
- `BC_Polit_Distrib`,
- `BC_Polit_Transf`,
- `U_Final_Mestre`,
- `U_Final_Escravo`,
- `BC_Final`,
- `U_Final`,
- `C_Final`.

O Submódulo Gerente de Tarefas

As rotinas deste submódulo são responsáveis por gerenciar o repositório de tarefas. Estas foram projetadas separadas das rotinas do módulo central para permitir a sua substituição, caso outra estrutura de dados deva ser utilizada para armazenar as tarefas. A versão disponível do repositório é implementada como uma lista. Todas as rotinas do módulo gerente de tarefas possuem o prefixo `GR_`. A implementação disponível deste submódulo encontra-se no arquivo `gr1.c`.

<u>`GR_Insere_Tarefa`</u>

Copia os dados que definem a tarefa em questão para uma nova área e insere a tarefa no repositório do processador executante. O módulo do usuário deverá utilizar esta rotina para o armazenamento inicial das tarefas no repositório. No caso de aplicações que gerem tarefas dinamicamente, esta rotina também será ativada ao longo da execução da aplicação.

▷ Parâmetros de entrada:
- endereço da área que contém os dados que definem a tarefa (`void*`),
- tamanho desta área (`int`),
- atributo reservado (`int`).

▷ Rotinas que a ativam:
- `C_Recebe_Tarefas`,
- esta rotina foi projetada também para ser utilizada por rotinas do módulo do usuário.

<u>`GR_Obtem_Tarefa`</u>

Recupera uma tarefa do repositório do processador executante. Esta rotina não libera a área de memória associada a tarefa retirada do repositório. Esta área só deverá ser liberada após a utilização dos dados que representam a tarefa, através da rotina `GR_Libera_Area`, definida a seguir. Por este motivo, a rotina `GR_Obtem_Tarefa` deve retornar o endereço do registro da tarefa no repositório que permitirá a referida liberação de memória. Cabe ressaltar que uma tarefa é retirada do repositório em duas situações: quando vai ser executada ou quando vai ser transferida para outro processador.

▷ Parâmetros de saída:
- endereço da área que contém os dados que definem a tarefa (`void**`),
- tamanho desta área (`int*`),
- atributo reservado (`int*`).

▷ Valor retornado:

- endereco do registro da tarefa que será, posteriormente, utilizado na liberação da área ocupada pela tarefa (`pt_reg_tar`).

▷ Rotinas que a ativam:

- `C_Envia_Tarefas`,
- `C_Executa_Tarefas`.

<u>`GR_Libera_Area`</u>

Libera área referente a uma tarefa já removida do repositório do processador executante. Será sempre ativada após a utilização dos dados que representam a tarefa recuperada pela rotina `GR_Obtem_Tarefa`.

▷ Parâmetros de entrada:

- endereço da área a ser liberada (`pt_reg_tar`).

▷ Rotinas que a ativam:

- `C_Envia_Tarefas`,
- `C_Executa_Tarefas`.

4 O Modelo SAMBA de Programação de Aplicações SPMD

Apresenta-se, nesta seção, o modelo de programação, a partir do qual, o usuário da ferramenta define uma aplicação SAMBA - o módulo do usuário.

Este modelo especifica o conjunto de rotinas que devem ser codificadas pelo usuário, em linguagem C, de forma a definir uma aplicação SAMBA. Duas rotinas são obrigatorias, independentemente da aplicação, no módulo do usuário: a que define as tarefas e a que as executa. A especificação das demais tenta capturar as possíveis necessidades de processamento de uma aplicação SPMD genérica. Dependendo da aplicação, algumas não são necessárias. Neste caso, devem ser definidas com corpo vazio.

Uma vez codificada a aplicação SAMBA (em um arquivo com extensão `.c`), o usuário, por meio de um *makefile*, poderá compilá-la e integrá-la à ferramenta. Além do nome do arquivo que contém a aplicação SAMBA, o usuário deverá identificar algum algoritmo de balanceamento de carga pertencente à respectiva biblioteca. Gera-se então, automaticamente, uma aplicação SPMD com balanceamento de carga. Este processo é ilustrado no Apêndice A.

Apresentam-se a seguir todas as rotinas, com seus respectivos objetivos, previstas pelo modelo SAMBA. Detalhes como o tipo dos parâmetros de entrada e de saída, assim como o tipo do valor retornado pela rotina, serão descritos. Cada rotina será ilustrada com um exemplo. O conjunto utilizado de rotinas-exemplo forma uma aplicação SAMBA que realiza uma multiplicação paralela de matrizes. Todas as rotinas de uma aplicação SAMBA (módulo do usuário) possuem o prefixo `U_`.

<u>`U_Inicial`</u>

Esta rotina é reservada para os procedimentos iniciais que o usuário necessitar codificar, tais como: inicialização de variáveis globais ou uma possível crítica dos parâmetros de ativação do programa. Para codificar esta possível crítica, o usuário deve lembrar que a ativação de uma aplicação gerada pela ferramenta ocorre da seguinte forma: nome do executável seguido dos parâmetros do usuário e dos parâmetros de balanceamento de carga separados por uma barra inclinada (`/`).

▷ Parâmetros de entrada:
- número total de parâmetros de ativação (`int`),
- vetor contendo os parâmetros de ativação (`char* []`).

▷ Prováveis rotinas ativadas:
- possíveis sub-rotinas auxiliares definidas pelo usuário. Cabe ressaltar que, sempre que o usuário julgar necessário, poderá modularizar, em um conjunto próprio de sub-rotinas, qualquer uma das rotinas definidas no modelo SAMBA.

▷ Rotina que a ativa:
- esta rotina é ativada em todos os processadores pelo programa principal.

Exemplo:

```
/* ------------------------------------------------------------------ */
void U_Inicial(argc,argv)
int argc;
char *argv[];
{
   if (argc<4)
      C_reporta_erro(C_ERRO,"\n\n Erro na Ativação do Programa.\n\n");
   Inicializa_Globais();
}
/* ------------------------------------------------------------------ */
```

Cabe lembrar que qualquer rotina do módulo do usuário, obrigatória ou não, poderá utilizar a rotina `C_Reporta_Erro`, do módulo central, para encerrar a execução da aplicação após a ocorrência de algum erro, reportando-o ao usuário.

Conforme visto na seção anterior, o módulo central, em seu programa principal, após ativar as rotinas iniciais de cada módulo, ativa as rotinas do usuário para a geração das tarefas. Há duas possibilidades previstas pela ferramenta para a geração das tarefas. Na primeira, no início do processamento, apenas um dos processadores possui os dados que definem as tarefas. Este processador, eleito mestre, deve ser, portanto, responsável pela geração destas. Na segunda, os dados que definem as tarefas encontram-se distribuídos no início do processamento. Neste caso, todos os processadores devem participar da geração das tarefas. Prevendo as duas possibilidades, o módulo central foi projetado para, no momento da geração das tarefas, ativar a rotina `U_Inicial_Mestre` no processador eleito mestre e a rotina `U_Inicial_Escravo` nos demais. Não havendo necessidade de um processamento inicial centralizado, estas duas rotinas poderão ter o mesmo código.

Alguns algoritmos de balanceamento de carga executam uma distribuição inicial de tarefas entre os processadores. Portanto, se estas tarefas já estiverem distribuídas, esta fase deverá não ser executada. Na Seção 5, esta questão será discutida com mais detalhes.

<u>`U_Inicial_Mestre`</u>

Esta rotina deve ser codificada, entre outros objetivos, para a geração de tarefas no processador eleito mestre. As partições de dados que definem as tarefas devem ser empacotadas e entregues ao gerente de tarefas para serem armazenadas no repositório do processador mestre. O usuário deve empacotar os dados que definem cada tarefa em uma área contígua de memória. Estas tarefas serão, posteriormente, devolvidas ao módulo do usuário para sua execução.

Além deste objetivo principal, esta rotina pode ser utilizada para a execução de outros procedimentos, cuja necessidade depende da aplicação, como: leitura e crítica de dados, envio de réplica de dados ou, genericamente, qualquer processamento que deva ser executado somente pelo mestre antes de o controle ser passado ao módulo de balanceamento de carga que distribuirá as tarefas e controlará sua execução.

▷ Parâmetros de entrada:

- vetor contendo os parâmetros de ativação (`char* []`),
- número total de processadores em execução (`int`), necessário, em alguns casos, na geração de tarefas.

▷ Prováveis rotinas ativadas:

- possíveis sub-rotinas auxiliares definidas pelo usuário,
- `GR_Insere_Tarefa`, direta ou indiretamente, pois o usuário poderá modularizar o código e ativá-la por meio de uma sub-rotina.

▷ Rotina que a ativa:

- esta rotina é ativada no processador mestre pelo programa principal.

Exemplo:

```
/* ------------------------------------------------------------------------ */
void U_Inicial_Mestre(argv,size)
char *argv[];
int size;
{
   int i;
   Le_Matriz(MA,argv[1],&nlina,&ncola);
   Le_Matriz(MB,argv[2],&nlinb,&ncolb);
   if (ncola!=nlinb)
      C_reporta_erro(C_ERRO,
      "\n\n Estas matrizes não podem ser multiplicadas.\n\n");
   Envia_Replica_Matriz(MB,nlinb,ncolb);
   /* Geração das tarefas. */
   for (i=0;i<nlina;i++)
      GR_Insere_Tarefa(MA[i],sizeof(int)*ncola,i);
}
/* ------------------------------------------------------------------------ */
```

U_Inicial_Escravo

Esta rotina é reservada para os procedimentos iniciais a serem executados nos processadores escravos, tais como: leitura e crítica de dados, recebimento de réplica de dados enviada pelo mestre ou, genericamente, qualquer processamento que deva ser executado somente pelos escravos antes de o controle ser passado ao módulo de balanceamento de carga.

Em algumas aplicações, os dados que definem as tarefas encontram-se distribuídos entre os processadores no início do processamento. Neste caso, esta rotina deve ser utilizada também para a geração de tarefas em todos os escravos. Eventualmente, terá o mesmo código da rotina `U_Inicial_Mestre`.

▷ Prováveis rotinas ativadas:
- possíveis sub-rotinas auxiliares definidas pelo usuário,
- `GR_Insere_Tarefa`, caso a aplicação gere as tarefas de forma distribuída.

▷ Rotina que a ativa:
- esta rotina é ativada nos processadores escravos pelo programa principal.

Exemplo:

```
/* ----------------------------------------------------------------------- */
void U_Inicial_Escravo()
{
   Recebe_Replica_Matriz(MB,&nlinb,&ncolb);
}
/* ----------------------------------------------------------------------- */
```

U_Final_Mestre

Esta rotina é reservada para os procedimentos finais a serem executados no mestre. Por exemplo: o recebimento dos resultados enviados pelos escravos e sua análise. Genericamente, deve ser codificado nesta rotina qualquer processamento a ser executado somente pelo mestre após a execução das tarefas e antes do encerramento da aplicação.

▷ Parâmetros de entrada:

- número total de processadores em execução (`int`), possivelmente necessário em um provável recebimento de resultados,
- vetor contendo os parâmetros de ativação (`char* []`).

▷ Prováveis rotinas ativadas:

- possíveis sub-rotinas auxiliares definidas pelo usuário.

▷ Rotina que a ativa:

- esta rotina é ativada no processador mestre pelo programa principal.

Exemplo:

```
/* ----------------------------------------------------------------------- */
void U_Final_Mestre(size,argv)
int size;
char *argv[];
{
   Recebe_Resultados(MR,ncolb,size);
   Grava_Matriz(MR,nlina,ncolb,argv[3]);
}
/* ----------------------------------------------------------------------- */
```

U_Final_Escravo

Esta rotina é reservada para os procedimentos finais a serem executados nos escravos. Por exemplo: o envio dos resultados ao mestre. Genericamente, deve ser codificado nesta rotina qualquer processamento a ser executado somente pelos escravos após a execução das tarefas e antes do encerramento da aplicação.

▷ Prováveis rotinas ativadas:
- possíveis sub-rotinas auxiliares definidas pelo usuário.

▷ Rotina que a ativa:
- esta rotina é ativada nos processadores escravos pelo programa principal.

Exemplo:

```
/* ----------------------------------------------------------------------- */
void U_Final_Escravo()
{
   Envia_Resultado(MR,ncolb);
}
/* ----------------------------------------------------------------------- */
```

<u>U_Final</u>

Esta rotina é reservada para os procedimentos finais a serem executados em todos os processadores. Por exemplo: a gravação de informações sobre a execução da aplicação, como o tempo de execução ou o número de tarefas executadas em cada procesador.

▷ Parâmetro de entrada:
- identificador do processador em questão (`int`).

▷ Prováveis rotinas ativadas:
- possíveis sub-rotinas auxiliares definidas pelo usuário.

▷ Rotina que a ativa:
- esta rotina é ativada em todos os processadores pelo programa principal.

Exemplo:

```
/* ----------------------------------------------------------------------- */
void U_Final(rank)
int rank;
{
   Informa_Tempos(rank);
}
/* ----------------------------------------------------------------------- */
```

<u>U_Executa_Tarefa</u>

Esta rotina deve conter a parte central de uma aplicação SPMD: o código que será executado sobre todas as partições de dados (tarefas) que compõem a aplicação. Inicialmente, deve desempacotar a tarefa enviada pelo módulo central, retirada do repositório do processador executante. Em seguida, deve executar o código SPMD sobre a partição de dados que define a tarefa em questão.

▷ Parâmetros de entrada:
- endereço da área que contém os dados que definem a tarefa (`void*`),
- tamanho desta área (`int`),
- atributo reservado (`int`).

▷ Prováveis rotinas ativadas:
- possíveis sub-rotinas auxiliares definidas pelo usuário.

▷ Rotina que a ativa:
- `C_Executa_Tarefas`.

Exemplo:

```
/* ---------------------------------------------------------------- */
void U_Executa_Tarefa(pt_tar_usu,tam_tar_usu,id_tar_usu)
void *pt_tar_usu;
int  tam_tar_usu;
int  id_tar_usu;
{
   int M[MAXNCOL];
   int i,j,k;
   /* Desempacotamento da tarefa. */
   ncola=tam_tar_usu/sizeof(int);
   for(i=0;i<ncola;i++)
   {
      M[i]=*((int *)pt_tar_usu);
      pt_tar_usu = (void *)((long)pt_tar_usu + sizeof(int));
   }
   /* Execução da tarefa: multiplicação de uma linha por uma coluna. */
   for (j=0; j<ncolb; j++)
      for (k=0; k<ncola; k++)
         MR[id_tar_usu][j] = MR[id_tar_usu][j] + (M[k] * MB[k][j]);
   MR[id_tar_usu][MAXNCOL] = 1;
}
/* ---------------------------------------------------------------- */
```

Todas as rotinas de uma aplicação SAMBA devem ser codificadas em um arquivo com extensão `.c`. No início deste arquivo, o usuário deverá incluir o arquivo `"central.h"` e poderá, se necessário, declarar suas constantes, tipos e variáveis globais. Cabe ressaltar que as variáveis globais definidas são criadas em todos os processadores.

As aplicações SPMD geradas pela ferramenta executam troca de mensagens através de chamadas ao MPI. Rotinas dos módulos central e de balanceamento de carga utilizam um conjunto de constantes (*tags*) para definir o protocolo de comunicação adotado. Caso o usuário utilize o MPI para trocas de mensagens, deve definir suas *tags* com valores maiores do que 20 para não conflitarem com as constantes dos outros módulos.

Exemplo:

```
/* ---------------------------------------------------------------------- */
#include "central.h"
/* Constantes da aplicação. */
#define MAXNLIN 1000
#define MAXNCOL 1000
/* TAGS MPI. */
#define TAG     21
#define FIM     22
#define ULTTAG  23
/* Variáveis globais. */
int MA[MAXNLIN][MAXNCOL];
int MB[MAXNLIN][MAXNCOL];
int MR[MAXNLIN][MAXNCOL+1];
int nlina,ncola;
int nlinb,ncolb;
/* ---------------------------------------------------------------------- */
```

As variáveis globais definidas serão criadas em todos os processadores e poderão, em cada um, ter valores diferentes. A matriz `MR`, por exemplo, terá valores distintos, nos diversos processadores, pois cada um fará o cálculo de um subconjunto diferente de linhas da matriz resultante.

Junto ao conjunto de arquivos que define a ferramenta (apresentado no Apêndice A) estão alguns arquivos que podem auxiliar o desenvolvimento de uma aplicação SAMBA (módulo do usuário). O arquivo `modelo.c` contém o esqueleto de uma aplicação SAMBA a partir do qual o usuário poderá desenvolver a sua própria aplicação. O arquivo `exemplo1.c` contém uma aplicação que implementa uma multiplicação paralela de matrizes, que também poderá auxiliar o desenvolvimento do módulo do usuário. O `exemplo2.c`, que implementa a mesma multiplicação paralela de matrizes, ilustra uma outra forma de utilização da ferramenta, chamada versão biblioteca, também apresentada no Apêndice A.

5 A Biblioteca de Algoritmos de Balanceamento de Carga

Uma das funções da ferramenta SAMBA é auxiliar o projetista de aplicações SPMD na identificação de uma estratégia de balanceamento de carga adequada à aplicação em desenvolvimento. Para tanto, a ferramenta dispõe de uma biblioteca de balanceamento de carga cujos algoritmos podem ser facilmente selecionados e incorporados à aplicação sendo projetada. O usuário poderá testar as diferentes estratégias disponíveis, sem custo algum de reprogramação, e selecionar a que apresentar o melhor desempenho.

O módulo de balanceamento de carga (Figura 2, Seção 3) pode ser preenchido por qualquer algoritmo da biblioteca. A escolha e a conseqüente integração de um algoritmo da biblioteca à aplicação em desenvolvimento ocorre por meio de um *makefile* da ferramenta. Este procedimento é ilustrado no Apêndice A.

O projeto desta biblioteca foi guiado pela taxonomia de algoritmos de balanceamento de carga para aplicações SPMD, proposta em [PRR98] e resumida na Seção 2. Os algoritmos da biblioteca implementam diferentes classes e características tratadas nesta taxonomia.

A versão atual da biblioteca possui dez algoritmos, apresentados a seguir. Novas estratégias podem ser inseridas, bastando que o projetista deste tipo de algoritmos conheça a estrutura do módulo de balanceamento de carga, apresentado no fim desta seção.

Algoritmos da Biblioteca

De acordo com a taxonomia apresentada em [PRR98], em relação à política de distribuição, uma estratégia de balanceamento de carga pode apresentar três comportamentos. Em uma estratégia estática, há uma distribuição inicial das tarefas seguida de sua execução. Em uma estratégia dinâmica sob demanda, a distribuição das tarefas ocorre ao longo da execução da aplicação de acordo com o comportamento dos processadores, tentando contornar os fatores dinâmicos de desequilíbrio. Em estratégias dinâmicas por migração, há uma distribuição inicial das tarefas com possíveis transferências destas ao longo da execução que visam desfazer os efeitos dos fatores dinâmicos de desequilíbrio. Estas últimas são caracterizadas por sua política de transferência.

Os algoritmos da biblioteca de balanceamento de carga são implementados em duas partes: a que se refere à política de distribuição e aquela que implementa a política de transferência (no caso de estratégias dinâmicas por migração).

A implementação separada destas duas políticas possibilita combinar diferentes políticas de distribuição com diferentes políticas de transferência sem custo de reprogramação. Além disso, permite que a política de distribuição de estratégias estáticas e de estratégias dinâmicas por migração seja facilmente retirada para que a ferramenta suporte o desenvolvimento de um conjunto específico de aplicações. Trata-se de aplicações em que os dados que definem as tarefas já se encontram distribuídos entre os processadores e as tarefas são diretamente armazenadas no respositório de cada processador.

Apresentam-se, a seguir, os dez algoritmos integrantes da versão atual da biblioteca. Após o identificador do algoritmo, acrescenta-se a sua classificação segundo a taxonomia de [PRR98]. Serão informados também os aquivos que contêm a implementação das políticas de distribuição e de transferência dos respectivos algoritmos.

Algoritmo 1 - Estático

Política de Distribuição

O processador mestre, que contém o conjunto inicial de tarefas, as distribui igualmente entre todos os processadores, incluindo o próprio.

A implementação desta política de distribuição encontra-se no arquivo `pd1.c`.

Política de Transferência

Não há política de transferência em estratégias estáticas. Após a distribuição das tarefas, estas são executadas sem que nenhuma ação visando o balanceamento de carga seja considerada.

O arquivo `pt1.c` contém a definição do cabeçalho das rotinas ativadas pelo módulo central previstas para implementar a política de transferência. Apenas uma ação é ativada: a execução das rotinas previamente distribuídas.

Este algoritmo não possui parâmetros de ativação.

Algoritmo 2 - Sob Demanda e Isolado

Política de Distribuição

Neste algoritmo, a distribuição das tarefas é feita ao longo da execução da aplicação de acordo com o comportamento dos processadores. Inicialmente, o processador mestre distribui um bloco de tarefas para cada processador escravo. O número de blocos é definido por um parâmetro de ativação. O número de tarefas em cada bloco é definido pela divisão da quantidade total de tarefas pelo número de blocos.

Após o recebimento do bloco inicial, cada processador escravo pede um novo bloco ao mestre quando encerra a execução das tarefas do bloco anterior. Enquanto houver blocos de tarefas, o processador mestre atende aos pedidos de novos blocos, enviando-os aos processadores solicitantes. O processador mestre não executa tarefas.

A implementação desta política de distribuição encontra-se no arquivo `pd2.c`.

Política de Transferência

Não há política de transferência em estratégias sob demanda. As tarefas são executadas nos processadores escravos imediatamente após o recebimento de cada bloco, não havendo transferência de tarefas entre estes processadores.

O arquivo `pt2.c` contém apenas a definição do cabeçalho das rotinas ativadas pelo módulo central previstas para implementar a política de transferência.

▷ Parâmetro de ativação do algoritmo: o número de blocos.

Algoritmo 3 - Distribuído, Síncrono, Global, Por Evento, Coletivo e Isolado

Política de Distribuição

O processador mestre, que contém o conjunto inicial de tarefas, as distribui igualmente entre todos os processadores, incluindo o próprio.

A implementação desta política de distribuição encontra-se no arquivo `pd1.c`.

Política de Transferência

Após a distribuição das tarefas, inicia-se a sua execução. Sempre que um processador encerra as suas tarefas, envia uma mensagem aos outros processadores solicitando a execução síncrona do balanceamento de carga. Cada processador verifica se algum outro solicitou o balanceamento de carga após toda execução de um conjunto de `T` tarefas. `T` é um parâmetro de ativação. Após a sincronização de todos os processadores, cada um - inclusive o que solicitou o balanceamento de carga - envia a todos os outros o seu índice de carga interna, no caso, a quantidade de tarefas ainda não executadas.

Em seguida, todos os processadores executam, ao mesmo tempo, o algoritmo que define as transferências necessárias ao equilíbrio de carga. Este algoritmo é efetivamente executado somente se a carga de algum processador for maior do que um certo limite (`lim`). Caso contrário, considera-se o balanceamento não mais necessário, pois a carga em todos os processadores encontra-se próxima ao fim. O valor `lim` é definido a partir de um parâmetro de ativação da aplicação. No início desta, define-se que `lim` equivale a `P%` do número de tarefas no processador mestre após a distribuição inicial. `P` é o parâmetro de ativação.

Todos os processadores definem então, diante da necessidade do balanceamento de carga, as transferências necessárias ao equilíbrio exato - mesma carga interna em todos os processadores. Finalmente, cada processador executa as transferências nas quais está envolvido.

Em aplicações SPMD nas quais tarefas são geradas dinamicamente, o parâmetro `P` deve ser definido com o valor 0, pois a proximidade do fim da aplicação não pode ser prevista pela carga aparente nos processadores.

A implementação desta política de transferência encontra-se no arquivo `pt3.c`.

▷ Parâmetro de ativação do algoritmo: o valor de `P` e o valor de `T`.

Algoritmo 4 - Centralizado, Global, Por Evento, Coletivo e Isolado

Política de Distribuição

O processador mestre, que contém o conjunto inicial de tarefas, as distribui igualmente entre todos os processadores, incluindo o próprio.

A implementação desta política de distribuição encontra-se no arquivo `pd1.c`.

Política de Transferência

Após a distribuição das tarefas, inicia-se a sua execução. Sempre que um processador encerra suas tarefas, envia uma mensagem aos outros processadores solicitando que estes enviem ao processador mestre o índice de carga interna; no caso, a quantidade de tarefas ainda não executadas. O processador mestre executará o algoritmo que define as transferências necessárias ao equilíbrio de carga. Todo processador, inclusive o mestre, verifica se algum outro solicitou o índice de carga após toda execução de um conjunto de `T` tarefas. `T` é um parâmetro de ativação.

Após receber o índice de carga de todos os processadores, inclusive do que solicitou o balanceamento de carga, o processador mestre inicia o algoritmo que define as transferências necessárias ao equilíbrio de carga. Este algoritmo é efetivamente executado somente se a carga de algum processador for maior do que um certo limite (`lim`). Caso contrário, considera-se o balanceamento não mais necessário, pois a carga em todos os processadores encontra-se próxima ao fim. Neste caso, o mestre avisa aos escravos que não haverá transferência de tarefas. O valor `lim` é definido a partir de um parâmetro de ativação da aplicação. No início desta, define-se que `lim` equivale a `P%` do número de tarefas no processador mestre após a distribuição inicial. `P` é o parâmetro de ativação.

Diante da necessidade do balanceamento de carga, o mestre então define as transferências de carga necessárias ao equilíbrio exato da carga em todos os processadores e informa aos escravos as transferências nas quais estes devem participar. Em seguida, cada processador executa as transferências indicadas.

Em aplicações SPMD nas quais tarefas são geradas dinamicamente, o parâmetro `P` deve ser definido com o valor 0, pois a proximidade do fim da aplicação não pode ser prevista pela carga aparente nos processadores.

A implementação desta política de transferência encontra-se no arquivo `pt4.c`.

▷ Parâmetro de ativação do algoritmo: o valor de `P` e o valor de `T`.

<u>Algoritmo 5</u> - Distribuído, Síncrono, Global, Periódico, Coletivo e Isolado

Política de Distribuição

O processador mestre, que contém o conjunto inicial de tarefas, as distribui igualmente entre todos os processadores, incluindo o próprio.

A implementação desta política de distribuição encontra-se no arquivo `pd1.c`.

Política de Transferência

Após a distribuição das tarefas, inicia-se a sua execução. O balanceamento de carga é ativado, periodicamente, em todos os processadores de forma síncrona. O período pode ser definido de duas formas: pela quantidade de tarefas executadas em um dos processadores, no caso, o mestre, ou por tempo corrido. O período e o seu tipo são definidos por meio de dois parâmetros de ativação da aplicação.

O mestre inicia o balanceamento de carga após executar a quantidade de tarefas definida como período ou após decorrido o tempo que define o período, enviando uma mensagem aos outros processadores solicitando a execução síncrona da estratégia. Cada processador verifica se o mestre solicitou o balanceamento de carga após toda execução de um conjunto de `T` tarefas. `T` é um parâmetro de ativação. Após a sincronização de todos os processadores, cada um - inclusive o mestre - envia a todos os outros o seu índice de carga interna, no caso, a quantidade de tarefas ainda não executadas.

Em seguida, todos os processadores executam, ao mesmo tempo, o algoritmo que define as transferências necessárias ao equilíbrio de carga. Calcula-se a média das cargas (`md`) de todos os processadores. O algoritmo continua somente se a carga de algum dos processadores encontra-se fora da faixa definida pelos valores `md-delta` e `md+delta`. Caso contrário, considera-se o balanceamento de carga desnecessário nesta ativação, pois a carga encontra-se relativamente equilibrada. O valor `delta` é definido a partir de um parâmetro de ativação da aplicação. No início da aplicação, define-se que `delta` equivale a `P%` do número de tarefas no processador mestre após a distribuição inicial. `P` é o parâmetro de ativação.

Todos os processadores definem então, diante da necessidade do balanceamento de carga, as transferências necessárias ao equilíbrio exato - a mesma carga interna em todos os processadores. Em seguida, cada processador executa as transferências nas quais está envolvido.

Cabe informar que as tarefas não são interrompidas para a execução do balanceamento de carga. Portanto, se o período definido em segundos se encerrar durante a execução de uma tarefa, o balanceamento só será ativado após a execução desta. Um novo período é iniciado após cada ativação do balanceamento de carga.

Nesta estratégia periódica, o balanceamento de carga não mais é ativado após encerradas as tarefas no processador que controla o período de ativação. Considera-se que, devido ao balanceamento freqüente, a carga nos outros processadores esteja também próxima ao fim.

A implementação desta política de transferência encontra-se no arquivo `pt5.c`.

▷ Parâmetros de ativação do algoritmo: o valor de `P`, a quantidade de tarefas ou o tempo em segundos que define o período, o tipo do período (`t` indica período definido por quantidade de tarefas e `s` indica período definido por tempo em segundos), e por último o valor de `T`.

Algoritmo 6 - Distribuído, Assíncrono, Global, Por Evento, Individual, Receptor, Não-Cego e Isolado

Política de Distribuição

O processador mestre, que contém o conjunto inicial de tarefas, as distribui igualmente entre todos os processadores, incluindo o próprio.

A implementação desta política de distribuição encontra-se no arquivo `pd1.c`.

Política de Transferência

Após a distribuição das tarefas, inicia-se a sua execução. Sempre que um processador encerra suas tarefas, envia uma mensagem aos demais solicitando o índice de carga interna de cada um, no caso, a quantidade de tarefas ainda não executadas. Com base nestes índices, o processador que ativou o algoritmo decidirá a que processador enviará um pedido de carga. Cada processador verifica se algum outro solicitou o índice de carga após toda execução de um conjunto de `T` tarefas. `T` é um parâmetro de ativação.

Após receber os índices, o processador responsável pelo balanceamento identifica que processador possui a maior carga e verifica se este valor é maior do que certo limite (`lim`). Este limite é definido a partir de um parâmetro de ativação da aplicação. No início da aplicação, define-se que `lim` equivale a `P%` do número de tarefas no processador mestre após a distribuição inicial. `P` é o parâmetro de ativação. Se a maior carga for maior do que o limite, uma mensagem é enviada ao processador que a possui para que este transfira metade de sua carga ao processador subcarregado. Esta transferência é então realizada. Caso contrário, considera-se o balanceamento de carga não mais necessário, pois a carga em todos os processadores encontra-se próxima ao fim.

Em aplicações SPMD nas quais tarefas são geradas dinamicamente, o parâmetro `P` deve ser definido com o valor 0, pois a proximidade do fim da aplicação não pode ser prevista pela carga aparente nos processadores.

A implementação desta política de transferência encontra-se no arquivo `pt6.c`.

▷ Parâmetro de ativação do algoritmo: o valor de `P` e o valor de `T`.

<u>Algoritmo 7</u> - Distribuído, Síncrono, Local, Particionado, Por Evento, Coletivo e Isolado

Neste algoritmo, as transferências de tarefas ocorrem somente entre processadores de um mesmo grupo. Cada processador pertence somente a um grupo, sendo então todos disjuntos. Os grupos são definidos pelo usuário em um arquivo, cujo nome é informado como parâmetro de ativação da aplicação.

Política de Distribuição

O processador mestre, que contém o conjunto inicial de tarefas, as distribui igualmente entre todos os processadores, incluindo o próprio.

A implementação desta política de distribuição encontra-se no arquivo `pd1.c`.

Política de Transferência

Após a distribuição das tarefas, inicia-se a sua execução. Sempre que um processador conclui as suas tarefas, envia uma mensagem aos outros processadores do mesmo grupo, solicitando a execução síncrona do balanceamento de carga. Cada processador verifica se algum outro do grupo solicitou o balanceamento de carga após toda execução de um conjunto de `T` tarefas. `T` é um parâmetro de ativação. Após a sincronização de todos os processadores do grupo, cada um - inclusive o que solicitou o balanceamento de carga - envia a todos os outros o seu índice de carga interna, no caso, a quantidade de tarefas ainda não executadas.

Em seguida, todos os processadores do grupo executam, ao mesmo tempo, o algoritmo que define as transferências necessárias ao equilíbrio de carga. Este algoritmo é efetivamente executado somente se a carga de algum processador do grupo for maior do que um certo limite (`lim`). Caso contrário, considera-se o balanceamento deste grupo não mais necessário, pois a carga nos seus processadores encontra-se próxima ao fim. O valor `lim` é definido a partir de um parâmetro de ativação da aplicação. No início desta, define-se que `lim` equivale a `P%` do número de tarefas no processador mestre após a distribuição inicial. `P` é o parâmetro de ativação.

Os processadores do grupo definem então, diante da necessidade do balanceamento de carga, as transferências necessárias ao equilíbrio exato - a mesma carga interna em todos os processadores do grupo. Em seguida, cada processador executa as transferências nas quais está envolvido.

Em aplicações SPMD nas quais tarefas são geradas dinamicamente, o parâmetro `P` deve ser definido com o valor 0, pois a proximidade do fim da aplicação não pode ser prevista pela carga aparente nos processadores.

A implementação desta política de transferência encontra-se no arquivo `pt7.c`.

▷ Parâmetros de ativação do algoritmo: o nome do arquivo que contém a definição do agrupamento, o valor de `P` e o valor de `T`.

▷ Formato do arquivo que contém a definição dos grupos: a primeira linha do arquivo contém o número de grupos, `n`. Cada grupo possui um identificador inteiro, `0..n-1`. Considere `p` processadores, com identificadores `0..p-1`. A segunda linha do arquivo contém o identificador do grupo do processador `0`. A terceira linha, o identificador do grupo do processador `1`. A `i`-ésima linha identifica então o grupo do processador `i-2`.

O conteúdo do arquivo exemplificado a seguir especifica que os oito processadores estão organizados em dois grupos. Os quatro primeiros, no grupo `0`, e os demais, no grupo `1`.

```
2
0
0
0
0
1
1
1
1
```

Algoritmo 8 - Distribuído, Assíncrono, Local, Por Vizinhança, Por Evento, Individual, Receptor, Não Cego e Isolado

Neste algoritmo, as transferências de tarefas ocorrem somente entre processadores de um mesmo grupo. Os grupos são formados considerando-se a vizinhança lógica entre os processadores. Neste caso, cada processador pode pertencer a mais de um grupo. A vizinhança lógica é definida pelo usuário em um arquivo, cujo nome é informado como parâmetro de ativação da aplicação.

Política de Distribuição

O processador mestre, que contém o conjunto inicial de tarefas, as distribui igualmente entre todos os processadores, incluindo o próprio.

A implementação desta política de distribuição encontra-se no arquivo `pd1.c`.

Política de Transferência

Após a distribuição das tarefas, inicia-se a sua execução. Sempre que um processador conclui as suas tarefas, envia uma mensagem aos seus processadores vizinhos, solicitando o índice de carga interna de cada um, no caso, a quantidade de tarefas ainda não executadas. Com base nestes índices, o processador que ativou o algoritmo decidirá a que processador enviará um pedido de carga. Cada processador verifica se algum vizinho solicitou o índice de carga após toda execução de um conjunto de `T` tarefas. `T` é um parâmetro de ativação.

Após receber os índices, o processador responsável pelo balanceamento identifica que vizinho possui a maior carga e verifica se este valor é maior do que um limite (`lim`). Este limite é definido a partir de um parâmetro de ativação da aplicação. No início da aplicação, define-se que `lim` equivale a `P%` do número de tarefas no processador mestre após a distribuição inicial. `P` é o parâmetro de ativação. Se a maior carga for maior do que o limite, uma mensagem é enviada ao processador que a possui para que este transfira metade de sua carga ao processador subcarregado. Esta transferência é então realizada. Caso contrário, considera-se a transferência desnecessária, pois a reduzida carga dos vizinhos sugere a proximidade do fim da aplicação.

Nesta estratégia de balanceamento de carga, há a necessidade de se adotar um algoritmo de detecção de término. A ausência de carga nos processadores vizinhos não é uma garantia, para o processador que encerrou as suas tarefas e ativou a estratégia, de fim da aplicação. A carga que, eventualmente, existir em uma região da rede de processadores pode migrar até atingir o processador subcarregado. Para se ter a certeza do fim da carga em todo o sistema (ou seja, de que não há carga migrando pela rede), adotou-se o algoritmo de detecção de término de Dijkstra [DSG83, KGGK94].

Em aplicações SPMD nas quais tarefas são geradas dinamicamente, o parâmetro `P` deve ser definido com o valor 0, pois a aparente carga reduzida nos vizinhos não indica sequer a proximidade do fim da aplicação.

A implementação desta política de transferência encontra-se no arquivo `pt8.c`.

▷ Parâmetros de ativação do algoritmo: o nome do arquivo que contém a definição da vizinhança lógica, o valor de `P` e o valor de `T`.

▷ Formato do arquivo que contém a definição dos grupos: considere `p` processadores, com identificadores `0..p-1`. A primeira linha do arquivo contém a lista de identificadores dos processadores vizinhos do processador `0`. O separador desta lista é o caracter branco e o valor `-1` indica o fim da lista. A segunda linha contém a lista de identificadores dos processadores vizinhos do processador `1`. A `i`-ésima linha contém então a lista de vizinhos do processador `i-1`.

O conteúdo do arquivo exemplificado a seguir define uma malha de nove processadores. A primeira linha indica que os processadores `1` e `3` são vizinhos do processador `0`. A última linha indica que os processadores `5` e `7` são vizinhos do processador `8`.

```
1 3 -1
0 2 4 -1
1 5 -1
0 4 6 -1
1 3 5 7 -1
2 4 8 -1
3 7 -1
4 6 8 -1
5 7 -1
```

Algoritmo 9 - Distribuído, Síncrono, Global, Por Evento, Coletivo e Integrado

Política de Distribuição

O processador mestre, que contém o conjunto inicial de tarefas, as distribui igualmente entre todos os processadores, incluindo o próprio.

A implementação desta política de distribuição encontra-se no arquivo `pd1.c`.

Política de Transferência

Após a distribuição das tarefas, inicia-se a sua execução. Sempre que um processador encerra as suas tarefas, envia uma mensagem aos outros processadores solicitando a execução síncrona do balanceamento de carga. Cada processador verifica se algum outro solicitou o balanceamento de carga após toda execução de um conjunto de `T` tarefas. `T` é um parâmetro de ativação. Após a sincronização de todos os processadores, cada um - inclusive o que solicitou o balanceamento de carga - envia a todos os outros os seus índices de carga interna e externa. O índice de carga interna é representado pela quantidade de tarefas ainda não executadas e o de carga externa, pela quantidade média de tarefas executadas por unidade de tempo. Esta quantidade média é calculada, em cada processador, a partir do número total de tarefas executadas dividido pelo tempo decorrido desde o início da aplicação. Este índice de carga externa reflete a capacidade média apresentada pelo processador no intervalo de tempo considerado.

Em seguida, todos os processadores executam, ao mesmo tempo, o algoritmo que define as transferências necessárias ao equilíbrio de carga. Este algoritmo é efetivamente executado somente se a carga de algum processador for maior do que um certo limite (`lim`). Caso contrário, considera-se o balanceamento não mais necessário, pois a carga em todos os processadores encontra-se próxima ao fim. O valor `lim` é definido a partir de um parâmetro de ativação da aplicação. No início desta, define-se que `lim` equivale a `P%` do número de tarefas no processador mestre após a distribuição inicial. `P` é o parâmetro de ativação.

Todos os processadores definem então, diante da necessidade do balanceamento de carga, as transferências necessárias ao equilíbrio da carga do sistema. Este equilíbrio de carga é relativo à capacidade média apresentada por cada processador. Portanto, cada processador, após a execução do balanceamento de carga, deverá possuir uma quantidade de tarefas proporcional à sua capacidade de processamento. Por fim, cada processador executa as transferências nas quais está envolvido.

Em aplicações SPMD nas quais tarefas são geradas dinamicamente, o parâmetro `P` deve ser definido com o valor 0, pois a proximidade do fim da aplicação não pode ser prevista pela carga aparente nos processadores.

A implementação desta política de transferência encontra-se no arquivo `pt9.c`.

▷ Parâmetro de ativação do algoritmo: o valor de `P` e o valor de `T`.

Algoritmo 10 - Distribuído, Síncrono, Local, Particionado, Por Evento, Coletivo e Integrado

Neste algoritmo, as transferências de tarefas ocorrem somente entre processadores de um mesmo grupo. Cada processador pertence somente a um grupo, sendo então todos disjuntos. Os grupos são definidos pelo usuário em um arquivo, exatamente como especificado no Algoritmo 7. O nome deste arquivo é informado como parâmetro de ativação da aplicação.

Política de Distribuição

O processador mestre, que contém o conjunto inicial de tarefas, as distribui igualmente entre todos os processadores, incluindo o próprio.

A implementação desta política de distribuição encontra-se no arquivo `pd1.c`.

Política de Transferência

Após a distribuição das tarefas, inicia-se a sua execução. Sempre que um processador conclui as suas tarefas, envia uma mensagem aos outros processadores do mesmo grupo, solicitando a execução síncrona do balanceamento de carga. Cada processador verifica se algum outro do grupo solicitou o balanceamento de carga após toda execução de um conjunto de `T` tarefas. `T` é um parâmetro de ativação. Após a sincronização de todos os processadores do grupo, cada um - inclusive o que solicitou o balanceamento de carga - envia a todos os outros os seus índices de carga interna e externa. O índice de carga interna é representado pela quantidade de tarefas ainda não executadas e o de carga externa, pela quantidade média de tarefas executadas por unidade de tempo. Esta quantidade média é calculada, em cada processador, a partir do número total de tarefas executadas dividido pelo tempo decorrido desde o início da aplicação. Este índice de carga externa reflete a capacidade média apresentada pelo processador no intervalo de tempo considerado.

Em seguida, todos os processadores do grupo executam, ao mesmo tempo, o algoritmo que define as transferências necessárias ao equilíbrio de carga. Este algoritmo é efetivamente executado somente se a carga de algum processador do grupo for maior do que um certo limite (`lim`). Caso contrário, considera-se o balanceamento deste grupo não mais necessário, pois a carga nos seus processadores encontra-se próxima ao fim. O valor `lim` é definido a partir de um parâmetro de ativação da aplicação. No início desta, define-se que `lim` equivale a `P%` do número de tarefas no processador mestre após a distribuição inicial. `P` é o parâmetro de ativação.

Os processadores do grupo definem então, diante da necessidade do balanceamento de carga, as transferências necessárias ao equilíbrio da carga do grupo. Este equilíbrio de carga é relativo à capacidade média apresentada por cada processador. Portanto, cada processador, após a execução do balanceamento de carga, deverá possuir uma quantidade de tarefas proporcional à sua capacidade de processamento. Por fim, cada processador do grupo executa as transferências nas quais está envolvido.

Em aplicações SPMD nas quais tarefas são geradas dinamicamente, o parâmetro `P` deve ser definido com o valor 0, pois a proximidade do fim da aplicação não pode ser prevista pela carga aparente nos processadores.

A implementação desta política de transferência encontra-se no arquivo `pt10.c`.

▷ Parâmetros de ativação do algoritmo: o nome do arquivo que contém a definição do agrupamento, o valor de `P` e o valor de `T`.

O Módulo de Balanceamento de Carga

Nesta subseção, detalha-se o módulo de balanceamento de carga. Definem-se as rotinas ativadas pelo módulo central que distribuem as tarefas e controlam a sua execução de forma balanceada, além daquelas que iniciam e encerram as atividades do módulo de balanceamento de carga. Esta subseção deverá guiar um projetista de estratégias de balanceamento de carga na implementação (em linguagem C) de uma nova estratégia, para fazer parte da biblioteca da ferramenta SAMBA.

Conforme visto anteriormente, os algoritmos de balanceamento de carga da biblioteca são implementados em duas partes: a que se refere à política de distribuição e aquela que implementa a política de transferência (no caso de estratégias dinâmicas por migração). Cada parte deve ser organizada em um arquivo com extensão `.c` separado.

No início de cada arquivo, o projetista deverá incluir o arquivo `"central.h"`. Neste, encontra-se declarada a variável externa que representa a quantidade total de tarefas ainda não executadas em cada processador (`extern int qtd_tar`), a qual é uma informação importante para um algoritmo de balanceamento de carga. Esta variável está definida no submódulo gerente de tarefas. O projetista poderá ainda, se necessário, declarar nestes arquivo suas constantes, tipos e variáveis globais. Cabe ressaltar que as variáveis globais definidas são criadas em todos os processadores.

As rotinas do módulo de balanceamento de carga devem utilizar funções MPI para implementar um provável protocolo de comunicação adotado. Deverão portanto utilizar um conjunto de constantes (*tags*) para definir este protocolo. Para não conflitarem com as *tags* dos outros módulos, as do balanceamento de carga devem ser definidas com valores entre 11 e 20, inclusive.

O novo algoritmo desenvolvido será compilado e integrado aos outros módulos da ferramenta através de um *makefile*. Este processo será ilustrado na próxima seção.

Apresentam-se a seguir as rotinas que compõem o módulo de balanceamento de carga. Detalhes como o tipo dos parametros de entrada e de saída, assim como o tipo do valor retornado pela rotina, serão descritos. As rotinas deste módulo possuem o prefixo `BC_`.

<u>`BC_Inicial`</u>

Esta rotina é reservada para os procedimentos executados no início da aplicação, necessários ao algoritmo de balanceamento de carga, tais como: crítica dos parâmetros de balanceamento de carga ou uma possível definição de grupos de comunicação.

▷ Parâmetros de entrada:
- vetor contendo os parâmetros de balanceamento de carga (`char* []`).
- número total de parâmetros de balanceamento de carga (`int`),
- número total de processadores em execução (`int`).

▷ Prováveis rotinas ativadas:
- possíveis sub-rotinas auxiliares definidas pelo projetista. Cabe ressaltar que, sempre que o projetista julgar necessário, poderá modularizar, em um conjunto próprio de sub-rotinas, qualquer uma das rotinas definidas no módulo de balanceamento de carga.

▷ Rotina que a ativa:
- esta rotina é ativada em todos os processadores pelo programa principal.

Cabe lembrar que qualquer rotina do módulo de balanceamento de carga poderá utilizar a rotina `C_Reporta_Erro`, do módulo central, para encerrar a execução da aplicação após a ocorrência de algum erro, relatando-o ao usuário.

<u>`BC_Final`</u>

Esta rotina é reservada para os procedimentos, relativos ao balanceamento de carga, executados no fim da aplicação. Por exemplo: liberação dos grupos de comunicação necessários em determinados algoritmos de balanceamento de carga.

▷ Prováveis rotinas ativadas:
- possíveis sub-rotinas auxiliares definidas pelo usuário.

▷ Rotina que a ativa:
- esta rotina é ativada em todos os processadores pelo programa principal.

<u>`BC_Polit_Distrib`</u>

Esta rotina deve implementar a política de distribuição do algoritmo de balanceamento de carga em questão.

▷ Parâmetros de entrada:
- identificador do processador em questão (`int`),
- número total de processadores em execução (`int`),
- vetor contendo os parâmetros de balanceamento de carga (`char* []`).

▷ Prováveis rotinas ativadas:
- possíveis sub-rotinas auxiliares definidas pelo usuário,
- `C_Envia_Protocolo`,
- `C_Recebe_Protocolo`,
- `C_Envia_Tarefas`,
- `C_Recebe_Tarefas`,
- `C_Executa_Tarefas`.

▷ Rotina que a ativa:
- esta rotina é ativada em todos os processadores pelo programa principal.

BC_Polit_Transf

Esta rotina deve implementar a política de transferência do algoritmo de balanceamento de carga em questão.

▷ Parâmetros de entrada:
- identificador do processador em questão (`int`),
- número total de processadores em execução (`int`),
- vetor contendo os parâmetros de balanceamento de carga (`char* []`).

▷ Prováveis rotinas ativadas:
- possíveis sub-rotinas auxiliares definidas pelo usuário,
- `C_Envia_Protocolo`,
- `C_Recebe_Protocolo`,
- `C_Envia_Tarefas`,
- `C_Recebe_Tarefas`,
- `C_Executa_Tarefas`.

▷ Rotina que a ativa:
- esta rotina é ativada em todos os processadores pelo programa principal.

Cabe ressaltar que a execução das tarefas pode ser ativada na rotina `BC_Polit_Distrib` ou na rotina `BC_Polit_Transf`, dependendo do tipo de algoritmo de balanceamento de carga. O primeiro caso pode ser exemplificado por estratégias sob demanda. O segundo, por estratégias por migração.

A rotina `BC_Polit_Distrib` deve ser implementada separadamente, em um arquivo diferente das outras rotinas. Isto permite combinar diferentes políticas de distribuição e diferentes políticas de trasferência sem custo de reprogramação. Além disso, facilita a eliminação da política de distribuição nos casos em que as tarefas já se encontram distribuídas nos diversos processadores no início da aplicação.

O arquivo `pd3.c` deve ser adotado quando se desejar não adotar política de distribuição. Este contém apenas a definição do cabeçalho da rotina ativada pelo módulo central prevista para implementar a política de distribuição.

Referências Bibliográficas

[AM96] J.N.C. Árabe e C.D. Murta, “Auto-balanceamento de carga em programas paralelos”, *Anais do VIII Simpósio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho*, 1996, 161–171.

[BS96] L. Baker e B.J. Smith, *Parallel Programming*, McGraw-Hill, 1996.

[CK88] T.L. Casavant e J.G. Kuhl, “A taxonomy of scheduling in general-purpose distributed computing systems”, *IEEE Transactions on Software Engineering* 14 (1998), 141–154.

[dBS97] R. de B. Seixas, *Visualização Volumétrica com Ray-Casting num Ambiente Distribuído*, Tese de doutorado, Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro, Brasil, 1997.

[DSG83] E. Dijkstra, W. Seijen e A. Gasteren, “Derivation of a termination detection algorithm for a distributed computation”, *Information Processing Letters* 16 (1983), 217–219.

[Duc96] R. Ducan, “MIMD architectures: Shared and distributed memory desings”, *Parallel and Distributed Computing Handbook* (A. Y. Zomaya, editor), Series on Computer Engineering, 680–698, McGraw-Hill, 1996.

[Eri96] H. Erika, “New results of the Salzburg NTN-method for the Radon transform”, *Proceedings of the Third International ACPC Conference with Special Emphasis on Parallel Databases and Parallel I/O*, Klagenfurt, Springer-Verlag, Lecture Notes in Computer Science 1127 (1996), 62–77.

[FG96] M.A. Franklin e V. Govindan, “A general matrix iterative model for dynamic load balancing”, *Parallel Computing* 22 (1996), 969–989.

[Fos94] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley, 1994.

[FTI90] M. Furuichi, K. Taki e N. Ichiyoshi, “A multi-level load balancing scheme for or-parallel exhaustive search programs on the multi-psi”, *Proceedings of the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1990, 50–59.

[GZ97] N. Guil e E.L. Zapata, “Fast Hough transform on multiprocessors: A branch and bound approach”, *Journal of Parallel and Distributed Computing* 45 (1997), 82–89.

[KGGK94] V. Kumar, A. Grama, A. Gupta e G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms*, Benjamin/Cummings, 1994.

[KR97] M. Kaddoura e S. Ranka, "Runtime support for parallelization of data-parallel applications on adaptive and nonuniform computational environment", *Journal of Parallel and Distributed Computing* 43 (1997), 163–168.

[LPR97] S. Lifschitz, A. Plastino e C.C. Ribeiro, "Exploring load balancing in parallel processing of recursive queries", *Proceedings of the Third International Euro-Par Parallel Processing Conference*, Passau, Springer-Verlag, Lecture Notes in Computer Science 1300 (1997), 1125–1129.

[Mat96] T.G. Mattson, "Scientific computation", *Parallel and Distributed Computing Handbook* (A. Y. Zomaya, editor), Series on Computer Engineering, 981–1002, McGraw-Hill, 1996.

[MPI93] Message Passing Interface Forum, "MPI: A message passing interface", *Proc. Supercomputing'93* (A. Y. Zomaya, editor), 878–883, IEEE Computer Society, 1993.

[PRR98] A. Plastino, C.C. Ribeiro e N. Rodriguez "Uma Taxonomia de Algoritmos de Balanceamento de Carga para Aplicações SPMD", Relatório Técnico PUC-RioInf.MCC49/98, Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro, Brasil, 1998.

[Qui94] M.J. Quinn, *Parallel Computing: Theory and Practice*, Series in Computer Science, McGraw-Hill, 1994.

[WLR93] M.A. Willebeek-LeMair e A.P. Reeves, "Strategies for dynamic load balancing on highly parallel computers", *IEEE Transactions on Parallel and Distributed Systems* 4 (1993), 979–993.

[ZLP97] M.J. Zaki, W. Li e S. Parthasarathy, "Customized dynamic load balancing for a network of workstations", *Journal of Parallel and Distributed Computing* 43 (1997), 156–162.

[Zom96] A.Y. Zomaya, "Parallel and distributed computing: The scene, the props, the players", *Parallel and Distributed Computing Handbook* (A. Y. Zomaya, editor), Series on Computer Engineering, 5–23, McGraw-Hill, 1996.

A Utilização da Ferramenta

Este apêndice trata de aspectos operacionais da ferramenta. Os arquivos que compõem a ferramenta são apresentados. A utilização do *makefile* da ferramenta é definida e ilustrada. Além disso, a utilização da ferramenta em um contexto particular - quando o programa principal reside no código do usuário e não no módulo central - é considerada.

Arquivos SAMBA

Os arquivos que compõem a ferramenta SAMBA estão organizados sob o diretório `samba`, ilustrado a seguir.

O subdiretório `central` contém os arquivos que implementam o módulo central. O arquivo `central.c` possui o núcleo da implementação deste módulo. O arquivo `central.h` contém definições de constantes e tipos utilizados por todos os módulos. O arquivo `centralbib.c` representa a versão do módulo central sem o programa principal. O arquivo `gr1.c` contém a implementação do submódulo gerente de tarefas disponível, o qual é definido a partir de uma lista.

O subdiretório `usuario` contém o *makefile*, utilizado pelo usuário para a geração de aplicações SPMD, e quatro subdiretórios: `fontes`, `obj`, `spmd` e `bib`. No primeiro, o usuário deverá armazenar os arquivos nos quais implementa suas aplicações SAMBA. Os arquivos objeto gerados pelo *makefile* da ferramenta serão armazenados no diretório `obj`. No `spmd`, estarão as aplicações SPMD (arquivos executáveis) geradas. No diretório `bib`, serão geradas as *libraries* para a versão biblioteca da ferramenta, descrita no fim deste apêndice. O diretório `fontes` posssui ainda os arquivos `modelo.c`, `exemplo1.c` e `exemplo2.c`. O primeiro contém o esqueleto de uma aplicação SAMBA a partir do qual o usuário poderá desenvolver a sua própria aplicação. O arquivo `exemplo1.c` contém uma aplicação que implementa uma multiplicação paralela de matrizes, que também poderá auxiliar o desenvolvimento do módulo do usuário. O `exemplo2.c`, que implementa a mesma multiplicação paralela de matrizes, ilustra uma outra forma de utilização da ferramenta, chamada versão biblioteca, apresentada na última subseção deste apêndice.

O subdiretório `bc` contém a biblioteca de balanceamento de carga, que por sua vez está organizado em outros dois subdiretórios: `pd` e `pt`. No primeiro, estão os arquivos que implementam diferentes políticas de distribuição. No segundo, os arquivos que implementam diferentes políticas de transferência.

```
/* ------------------------------------------------------------------------ */
samba/
    central/
        central.c
        central.h
        centrallib.c
        gr1.c
    usuario/
        makefile
        fontes/
           exemplo1.c
           exemplo2.c
           modelo.c
        obj/
        spmd/
        bib/
    bc/
        pd/
            pd1.c
            pd2.c
            pd3.c
        pt/
            pt1.c
            pt2.c
            pt3.c
            pt4.c
            pt5.c
            pt6.c
            pt7.c
            pt8.c
            pt9.c
            pt10.c
/* ------------------------------------------------------------------------ */
```

Makefile

A compilação dos fontes que implementam componentes de uma aplicação SPMD e a integração dos códigos gerados ocorrem por meio de um *makefile*. Os componentes em questão são: a aplicação SAMBA, as políticas de distribuição e de transferência que definem o algoritmo de balanceamento de carga, o módulo central e o gerente de tarefas.

A ativação do *makefile* da ferramenta deve ocorrer no diretório `samba/usuário`. A sintaxe geral de ativação é a seguinte:

`make SP=spx PD=pdx PT=ptx GR=grx`, onde:

- o arquivo `spx.c` contém a aplicação SAMBA desenvolvida pelo usuário e deve estar localizado no diretório `samba/usuario/fontes`;
- o arquivo `pdx.c` contém a política de distribuição do algoritmo de balanceamento de carga adotado e deve estar localizado no diretório `samba/bc/pd`; `pd1.c` é a opção *default*;
- o arquivo `ptx.c` contém a política de transeferência do algoritmo de balanceamento de carga adotado e deve estar localizado no diretório `samba/bc/pt`; `pt1.c` é a opção *default*;
- o arquivo `grx.c` contém a implementação do gerente de tarefas adotado e deve estar localizado no diretório `samba/central`; `gr1.c` é a opção *default*.

O resultado desta instrução é a geração da aplicação SPMD definida pela aplicação SAMBA `spx.c`, pelo algoritmo de balanceamento de carga formado por `pdx.c` e `ptx.c`, e pelo gerente de tarefas `grx.c`. Caso algum dos componentes envolvidos seja recém criado ou alterado, será compilado antes de integrado aos demais.

A aplicação é gerada em um arquivo executável, no diretório `samba/usuario/spmd`. O nome deste arquivo é formado pela concatenação dos nomes dos arquivos envolvidos, naquela ordem: `spxpdxptxgrx`.

Para incluir uma nova estratégia na biblioteca de balanceamento de carga ou disponibilizar um novo gerente de tarefas, o usuário deve apenas então inserir os arquivos correspondentes nos diretórios indicados.

Seguem exemplos de utilização do *makefile*.

A instrução `make SP=sp1` gera a aplicação SPMD `sp1pd1pt1gr1`, a partir da aplicação SAMBA `sp1.c`, do algoritmo de balanceamento de carga *default* (Algoritmo 1 - Seção 5), definido por `pd1.c` e `pt1.c`, e do gerente de tarefas *default* `gr1.c`.

A instrução `make SP=sp1 PD=pd1 PT=pt3` gera a aplicação SPMD `sp1pd1pt3gr1`, a partir da aplicação SAMBA `sp1.c`, do algoritmo de balanceamento de carga definido por `pd1.c` e `pt3.c` (Algoritmo 3 - Seção 5), e do gerente de tarefas *default* `gr1.c`.

A instrução `make SP=sp1 GR=gr2` gera a aplicação SPMD `sp1pd1pt1gr2`, a partir da aplicação SAMBA `sp1.c`, do algoritmo de balanceamento de carga *default*, definido por `pd1.c` e `pt1.c`, e de um novo gerente de tarefas `gr2.c`.

A instrução `make SP=sp1 PD=pd1 PT=pt9 GR=gr2` gera a aplicação SPMD `sp1pd1pt9-gr2`, a partir da aplicação SAMBA `sp1.c`, de um novo algoritmo de balanceamento de carga definido por `pd1.c` e `pt9.c`, e do gerente de tarefas `gr2.c`.

Versão Biblioteca

O programa principal da aplicação SPMD gerada pela versão apresentada da ferramenta reside no módulo central. O usuário, neste caso, deve codificar as rotinas que serão ativadas por este módulo e pelo módulo de balanceamento de carga. Entretanto, há uma versão de utilização da ferramenta, chamada versão biblioteca, a qual considera que o programa principal reside no módulo do usuário. Este fica responsável, então, pela ativação dos demais módulos.

Esta versão foi desenvolvida para permitir a ativação da ferramenta em um ponto qualquer do código de uma aplicação paralela. Cabe ressaltar que há aplicações paralelas em que apenas uma fase do código é caracterizada como SPMD.

A versão biblioteca permite também a ativação da ferramenta por uma aplicação SPMD já desenvolvida. Neste caso, o interesse maior na utilização da ferramenta estaria em avaliar e adotar um algoritmo da biblioteca de balanceamento de carga.

A versão biblioteca pode ser útil também no desenvolvimento de aplicações SPMD iterativas [PRR98]. Neste caso, a ferramenta seria ativada no início de cada iteração para controlar a execução, de forma balanceada, das tarefas da aplicação.

Para utilizar a versão biblioteca, o usuário deve, inicialmente, gerar uma *library* contendo o módulo central integrado ao algoritmo de balanceamento de carga e ao gerente de tarefas escolhidos. Para tanto, o usuário deve ativar o *makefile*, informando a palavra `bib` e os seguintes parâmetros:

`make bib PD=pdx PT=ptx GR=grx`, onde:

- o arquivo `pdx.c` contém a política de distribuição do algoritmo de balanceamento de carga adotado e deve estar localizado no diretório `samba/bc/pd`; `pd1.c` é a opção *default*;
- o arquivo `ptx.c` contém a política de transeferência do algoritmo de balanceamento de carga adotado e deve estar localizado no diretório `samba/bc/pt`; `pt1.c` é a opção *default*;
- o arquivo `grx.c` contém a implementação do gerente de tarefas adotado e deve estar localizado no diretório `samba/central`; `gr1.c` é a opção *default*.

O resultado desta instrução é a geração de uma *library* contendo o módulo central integrado ao algoritmo de balanceamento de carga formado por `pdx.c` e `ptx.c`, e ao gerente de tarefas `grx.c`. Caso algum dos componentes envolvidos seja recém criado ou alterado, será compilado antes de integrado aos demais.

A biblioteca é gerada em um arquivo com extensão `.a`, no diretório `samba/usuario/bib`. O nome deste arquivo é formado pela concatenação do nome `libsamba` e dos nomes dos arquivos envolvidos, naquela ordem: `libsambapdxptxgrx.a`.

Uma vez gerada a biblioteca, o usuário poderá compilar o seu módulo e integrá-lo a biblioteca, ativando o *makefile* da seguinte forma:

`make spmdbib SP=spx BB=bbx`, onde:

- o arquivo `spx.c` contém o código do usuário e deve estar localizado no diretório `samba/usuario/fontes`;

- o arquivo `libsambabbx.a` contém a biblioteca adotada e deve estar localizado no diretório `samba/usuario/bib`.

O resultado desta instrução é a geração da aplicação SPMD definida pelo código do usuário `spx.c` e pela biblioteca `libsambabbx.a`. Caso o código do usuário seja recém criado ou alterado, será compilado antes de integrado à biblioteca.

A aplicação é gerada em um arquivo executável, no diretório `samba/usuario/spmd`. O nome deste arquivo é formado pela concatenação dos nomes dos arquivos envolvidos, naquela ordem: `spxbbx`.

Para realizar esta última tarefa, o usuário poderá ter o seu próprio *makefile*. Isto provavelmente ocorrerá quando a ferramenta for utilizada em aplicações paralelas já desenvolvidas.

Devido à ausência do programa principal no módulo central, para utilizar a ferramenta na versão biblioteca, o usuário deve ficar responsável pela ativação das seguintes rotinas: `C_Inicial`, `U_Inicial`, `BC_Inicial`, `U_Inicial_Mestre`, `U_Inicial_Escravo`, `U_Final_Mestre`, `U_Final_Escravo`, `BC_Final`, `U_Final` e `C_Final`.

Cabe ressaltar que, na versão biblioteca, as rotinas do módulo do usuário serão não só escritas mas também ativadas pelo usuário (com exceção da rotina `U_Executa_Tarefa` que continuará sendo ativada pelo módulo de balanceamento de carga). Portanto, o usuário pode abandonar, se achar importante, os nomes e a modularização adotados pela ferramenta na versão original.

O usuário deve ativar o módulo central por meio da rotina `C_Main`. Esta, basicamente, ativa as seguintes rotinas do módulo de balanceamento de carga: `BC_Polit_Distrib` e `BC_Polit_Transf`.

A ativação da rotina `C_Main` deve ocorrer após a execução das funções previstas para as rotinas `U_Inicial_Mestre` e `U_Inicial_Escravo`, ou seja, após a geração das tarefas e antes das funções previstas para `U_Final_Mestre` e `U_Final_Escravo`. Segue a especificação da nova rotina do módulo central `C_Main` e do novo programa principal, agora no módulo do usuário.

<u>`C_Main`</u>

Esta rotina substitui o programa principal no módulo central.

▷ Parâmetros de entrada:
- vetor contendo os parâmetros de balanceamento de carga (`char* []`),
- identificador do processador em questão (`int`),
- número total de processadores em execução (`int`).

▷ Rotinas ativadas:
- `BC_Polit_Distrib`,
- `BC_Polit_Transf`.

▷ Rotina que a ativa:
- esta rotina é ativada em todos os processadores pelo programa principal, no módulo do usuário.

<u>`Programa Principal`</u>

O programa principal é ativado no início da execução da aplicação em todos os processadores. Na versão biblioteca, o programa principal reside no módulo do usuário, que deverá, direta ou indiretamente, ativar as rotinas dos demais módulos, conforme a especificação a seguir. Primeiramente, deve ativar as rotinas que fazem as críticas iniciais em cada módulo: `C_Inicial`, `U_Inicial` e `BC_Inicial`. A rotina `C_Inicial` deve ser a primeira a ser ativada, pois retorna valores utilizados pelas demais. Em seguida, o programa principal deve ativar, simultaneamente, as rotinas do usuário projetadas para a geração de tarefas: `U_Inicial_Mestre` - em um processador eleito mestre - e `U_Inicial_Escravo` - nos demais processadores. O módulo central deve ser então ativado, em todos os processadores, por meio da rotina `C_Main`. Esta ativa o módulo de balanceamento de carga para controlar a execução das tarefas. Em seguida, o controle volta ao programa principal (módulo do usuário) para eventual tratamento de resultados. Para esta função, também rotinas distintas devem ser ativadas no mestre e nos escravos: `U_Final_Mestre` e `U_Final_Escravo`. A execução da aplicação se encerra após a ativação das rotinas `BC_Final`, `U_Final` e `C_Final`.

▷ Parâmetros de entrada:
- número de parâmetros de ativação (`int`),
- vetor contendo os parâmetros de ativação (`char* []`).

▷ Rotinas ativadas:

- C_Inicial,
- U_Inicial,
- BC_Inicial,
- U_Inicial_Mestre,
- U_Inicial_Escravo,
- C_Main,
- U_Final_Mestre,
- U_Final_Escravo,
- BC_Final,
- U_Final,
- C_Final.

Exemplo:

```
/* ------------------------------------------------------------------------ */
main (argc,argv)
int argc;
char *argv[];
{
   char *parametros[C_NUMMAXPARAM];
   int npar, rank, size;
   C_Inicial(&argc,argv,&rank,&size,parametros,&npar);
   U_Inicial(argc,argv);
   BC_Inicial(parametros,npar,size);
   if (rank==C_ROOT)
      U_Inicial_Mestre(argv,size);
   else
      U_Inicial_Escravo();
   C_Main(parametros,rank,size);
   if (rank==C_ROOT)
      U_Final_Mestre(size,argv);
   else
      U_Final_Escravo();
   BC_Final();
   U_Final(rank);
   C_Final();
}
/* ------------------------------------------------------------------------ */
```