Funções MPI de Comunicação Coletiva Reconfiguráveis

Luiz Eduardo da Silva Ramos[1] e Carlos Augusto Paiva da Silva Martins[2]
Programa de Pós-Graduação em Engenharia Elétrica, PUC-MG
luizedu@uai.com.br[1] capsm@puminas.br[2]

Resumo

As funções MPI (Message Passing Interface) de comunicação coletiva (FMCC) são comumente implementadas em bibliotecas que utilizam algoritmos invariáveis. Nem sempre elas apresentam o melhor desempenho em todos os tipos de aplicações e ambientes de execução. Neste artigo, apresentamos, simulamos, modelamos analiticamente, verificamos e analisamos FMCC que apresentam estruturas e comportamentos que podem ser alterados para fornecer configurações, flexibilidade e desempenho otimizados. Nossos principais ***objetivos*** *são: propor e apresentar um conjunto de FMCC (reconfiguráveis) otimizadas; apresentar, simular, modelar analiticamente, verificar e analisar as funções propostas. Nossas* ***metas*** *são: simular diferentes versões de FMCC, incluindo uma versão reconfigurável, e fornecer uma análise comparativa entre implementações fixas e reconfiguráveis. Os* ***resultados*** *mostram que a reconfiguração no nível de algoritmo realmente produz ganhos de flexibilidade e de desempenho em FMCCs.*

1. Introdução

Na última década, os aglomerados de computadores (*clusters of workstations*) se tornaram uma tendência em computação de alto desempenho. Eles são alternativas de baixo custo aos supercomputadores, pois normalmente são construídos com hardware de prateleira e software gratuito, além de apresentarem alto desempenho. Os aglomerados são compostos de nós individuais de processamento com módulos privativos de memória interconectados por uma ou mais redes [1].

As aplicações para aglomerados são comumente compostas por processos que trocam mensagens entre si para a comunicação de dados ou para realizarem sincronizações. O MPI (*Message Passing Interface*) [2] é um padrão *de-facto* para o suporte à passagem de mensagens. Ele fornece um conjunto de funções que permitem comunicações ponto-a-ponto ou coletivas entre processos.

Funções de comunicação coletiva permitem que um grupo de processos contribua com dados ou com sinais de controle para produzir um resultado ou um comportamento. De acordo com a sua cardinalidade, as funções podem ser classificadas como: 1xN; Nx1; e NxN [2] [3], e podem ser implementadas usando-se padrões de comunicação que representam os modelos de transmissão de dados em um grupo de processos.

O desempenho das funções MPI de comunicação coletiva (FMCC) é um fator crítico para a maioria das aplicações baseadas em MPI. Esforços anteriores foram feitos visando ganhos de desempenho através de otimizações específicas. Elas permitiam que a aplicação tirasse vantagem dos recursos computacionais de um dado sistema [3] [4] [5] [6] [7].

O nosso principal problema é o fato de que as implementações do padrão MPI geralmente usam algoritmos fixos (invariáveis) que não fornecem o melhor desempenho em todos os tipos de aplicações e ambientes de execução. Em nossa pesquisa, não foram encontrados trabalhos que utilizassem uma abordagem reconfigurável no MPI no nível de algoritmo [8] [9]. Um trabalho recente do nosso grupo de pesquisa aplicou esta abordagem em algoritmos paralelos de escalonamento em sistemas paralelos [10]. Assim, nossa **motivação** é melhorar o desempenho das FMCC aumentando a flexibilidade de seus algoritmos usando conceitos de Computação Reconfigurável. Este novo paradigma combina o alto desempenho do hardware e a flexibilidade das implementações de software [8] [9].

Nossos **principais objetivos** são: propor e apresentar um conjunto de FMCC reconfiguráveis otimizadas em desempenho; simular, modelar analiticamente, verificar e analisar as funções propostas. Nossas **metas** são: simular diferentes versões de FMCC, incluindo uma versão reconfigurável, e fornecer uma análise comparativa entre implementações fixas e reconfiguráveis.

2. FMCC Reconfiguráveis (FMCCR)

Atualmente, algoritmos reconfiguráveis de vários tipos estão sendo desenvolvidos em nosso grupo de pesquisa [9] [10] [11]. Eles combinam os conceitos de: *framework*, composição de software [12] e hardware reconfigurável [9]. Nossa proposta de FMCCR é baseada em camadas hierárquicas, de forma que a reconfiguração pode ocorrer em diferentes níveis arquiteturais (aplicação, algoritmo, *middleware*, pilha de protocolos e camada física). Entretanto, neste artigo, nós nos concentramos no nível de algoritmo. Um algoritmo reconfigurável é composto de blocos construtivos que possibilitam a alteração do comportamento por meio de mudanças estruturais em sua configuração. Ele é organizado em três camadas: a Básica (CB), a Reconfigurável (CR) e a de Controle de Configuração (CCC), representadas na Figura 1.

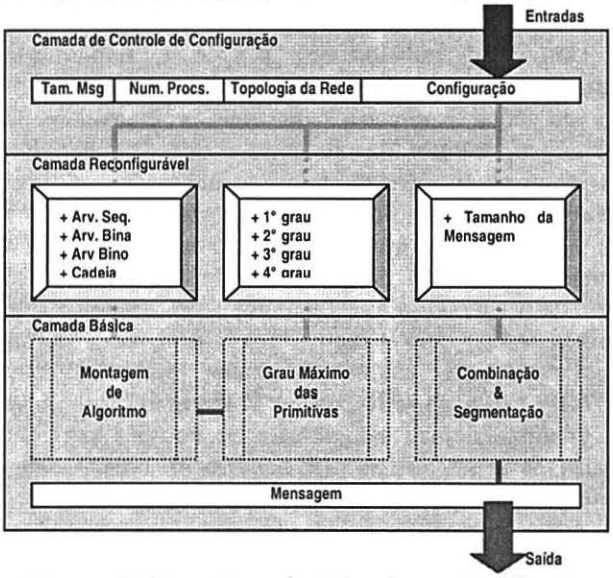

Figura 1. Uma Possível Implementação da Função MPI_Bcast Reconfigurável.

A **Camada Básica** (CB) é um framework composto de estruturas de dados (para armazenamento) e um conjunto de quadros. Um quadro de ação representa uma parte ou fase de um algoritmo, enquanto um quadro de controle controla uma característica específica de uma estrutura de dados. Neste trabalho, nos concentramos no nível de algoritmo que foi dividido em três partes: Montagem de Algoritmo; Combinação e Segmentação de Mensagens; e Grau Máximo das Primitivas. A partir dessas partes, nós fizemos quadros para a CB (Figura 1).

O quadro de Montagem de Algoritmo (ação) é usado para criar uma topologia lógica entre as entidades comunicantes selecionando, combinando e configurando os padrões de comunicação. O quadro de Combinação e Segmentação (controle) é responsável por combinar mensagens, para reduzir a latência de comunicação, ou segmentá-las, para favorecer: o chaveamento de pacotes, o controle de fluxo, o controle de erros e os armazenamentos temporários (*buffering*). O quadro de Grau Máximo das Primitivas (ação) configura o grau das primitivas de comunicação para se beneficiar quando a rede apresentar um grau de conexão maior que um.

A **Camada Reconfigurável** (CR) pode ser vista como uma camada de interface que configura a CB. Na verdade ela é uma instância de CB na qual cada frame é preenchido com um ou mais blocos construtivos (implementações de quadros) compatíveis com o tipo de operação realizada em um dado momento.

A **Camada de Controle de Configuração** (CCC) é responsável por selecionar e trocar os blocos construtivos que preenchem os quadros em um dado momento. Essas decisões podem ser feitas em função de parâmetros de entrada, informação dinâmica da carga de trabalho, comandos do sistema operacional, escolha do usuário etc. Neste artigo, focalizamos a proposta das FMCCR. Assim, não nos preocupamos em descrever uma camada CCC sofisticada e em discutir a sobrecarga da reconfiguração, que está diretamente relacionada com a complexidade dos algoritmos usados nessa camada. Sugerimos uma CCC simplificada, implementada como uma tabela contendo informações previamente coletadas e processadas. A tabela mantém a melhor configuração disponível de acordo com informações dinâmicas e estáticas (ex: topologia da rede) do sistema e com parâmetros da carga de trabalho (ex: tamanho da mensagem e número de processos).

Para simplificar e reduzir o número de simulações, o primeiro quadro foi simulado com blocos configuráveis contendo diferentes opções e os dois últimos quadros com blocos contendo uma configuração. Assim, as funções reconfiguráveis simuladas transmitem apenas mensagens de tamanho original, usando primitivas de primeiro grau. Na modelagem analítica, variamos também o grau das primitivas. Os blocos construtivos para o primeiro frame são os padrões de comunicação descritos analiticamente abaixo.

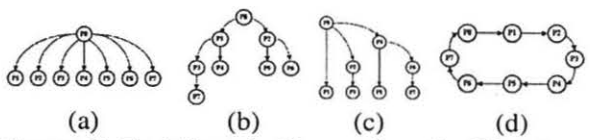

Figura 2. Padrões de Comunicação Simples: (a) Árvore Seqüencial, (b) Árvore Binária, (c) Árvore Binomial e (d) Anel.

A **Árvore Seqüencial** é um padrão centralizado no qual o processo raiz ou coordenador envia seus pacotes (mensagens combinadas) para os outros N-1 processos, como pode ser visto na Figura 2a [4] [6].

No **Full-FanOut**, todos os processos enviam seus pacotes simultaneamente e então esperam por N-1 pacotes dos outros. Entretanto pode ser necessário o armazenamento temporário de pacotes, consumindo uma grande quantidade de memória [5].

Na **Árvore Binária**, o processo R ∈ [0,...,N-1] tem dois filhos (descendentes): R1=(2*R)+1 e R2=(2*R)+2 somente se R1 e R2 são menores que N. Neste padrão, um processo encaminha os pacotes recebidos de seu pai para seus filhos e então envia seu próprio pacote para os filhos (veja Figura 2b) [3] [4] [6].

Na **Árvore Binomial**, o grupo de processos é dividido recursivamente em dois. Quando todos os grupos tiverem um elemento, o primeiro processo de cada grupo envia seus pacotes para o primeiro processo do segundo grupo. Assim, as mensagens atingem recursivamente todos os N processos em $\log_2 (N-1)+1$ passos [3] [4] [6].

No padrão **Estrela**, o processo raiz recebe pacotes de todos os processos do grupo de maneira seqüencial. Então, ele as envia em uma mensagem combinada de maneira reversa [3].

No **FanIn-FanOut**, todas as mensagens são combinadas em árvore binária reversa até atingirem o raiz. A mensagem final combinada é transmitida a todos os processos em árvore binária [3] [5].

No padrão **Cadeia**, o processo R1 ∈ [0,...,N-2] envia seus pacotes para R2=(R1+1)modN, enquanto R3 ∈ [0,...,N-1] recebe uma mensagem de R4=(N+R3-1)modN [4] [5]. O **Anel** (Figura 2d) é semelhante ao padrão Cadeia. A diferença é que no Anel, R ∈ [0,...,N-1] envia pacotes para R1=(R+1)modN e recebe pacotes de R2=(N+R-1)modN, possibilitando explorar o paralelismo temporal [6].

O **Circular** é uma versão diferente de anel, no qual em um dado passo S ∈ [1,...,N], o processo R envia um pacote para R1=(R+S)modN e recebe outro de R2=(N+R-S)modN. Neste padrão ocorrem trocas bidirecionais em alguns passos, que pode ser benéfico se a rede oferecer suporte [6].

O **Shuffle** ocorre em S ∈ [1,...,teto($\log_2$ N)] passos e apresenta o melhor desempenho possível quando a rede permite trocas bidirecionais simultâneas. Em cada passo, o processo R se comunica com outro a uma distância d=2^S, trocando dados recebidos anteriormente, de modo que os pacotes podem dobrar de tamanho em cada passo dependendo do tipo de operação coletiva implementada com o Shuffle [3] [5].

O **Pairwise** é semelhante ao Shuffle, mas consome S ∈ [1,...,N-1] passos. O grupo de processos é dividido recursivamente ao meio até que só haja subgrupos com dois elementos. Cada processo desses subgrupos troca pacotes diretamente com cada processo do grupo vizinho. Como resultado, todos os processos trocam pacotes com os outros de modo que nenhuma troca bidirecional simultânea acontece. Além disso, todas as mensagens trocadas têm o mesmo tamanho [3].

3. Trabalhos Relacionados

Nós encontramos vários trabalhos correlatos em diferentes aspectos. Todos eles tinham como objetivos melhorar o desempenho da comunicação coletiva no MPI [7]. Alguns deles se dedicam ao desenvolvimento de algoritmos otimizados para arquiteturas específicas e tentaram minimizar a contenção nos links e a latência entre nós comunicantes [4] [5] [7] [14].

Um outro trabalho [3] propõe e implementa otimizações estáticas em software selecionadas de acordo com parâmetros de entrada (tamanho da mensagem, número de processos e tipo da rede). Em [6], essas otimizações são feitas de maneira automática. Realizam-se testes e medem-se parâmetros do sistema para ajustar os algoritmos que implementam operações coletivas. Ambas abordagens são adaptativas e permitem a escolha dinâmica de padrões de comunicação previamente implementados. Outras implementações adaptativas estão presentes em: FT/MPI [6], Scali MPI Connect [13] e em [7].

O trabalho mais relacionado ao nosso é [14], no qual uma arquitetura de MPI baseada em componentes é proposta e um framework de suporte é implementado. O framework permite seleção estática e dinâmica, ligação e configuração de módulos que possibilitam a coexistência de mais de um módulo de um mesmo componente. Utilizam-se componentes com granularidades grossas, como: operações ponto-a-ponto ou coletivas, checkpoint-restore e boot.

Entretanto, com a nossa proposta seria possível decompor uma função em módulos menores. Teríamos um conjunto de blocos construtivos que poderiam ser dinamicamente usados para criar várias funções completas coexistentes, economizando memória. Os blocos poderiam ser implementados como: classes, macros, procedimentos, funções, módulos etc. Nossa proposta apresenta uma menor granularidade de reconfiguração, e neste aspecto se assemelha, mas não se limita, aos algoritmos híbridos [7] [19]. O algoritmo reconfigurável pode ser modificado dinamicamente com a adição de novos padrões de comunicação não implementados ou disponíveis em tempo de projeto.

4. Resultados da Simulação

Nesta seção, apresentamos, analisamos e comparamos os resultados obtidos na simulação das FMCC com implementações fixas e reconfiguráveis. Nosso método é composto por seis fases: (1) seleção de FMCCs para reconfiguração; (2) implementação das funções selecionadas usando diferentes padrões de comunicação; (3) seleção do ambiente de simulação; (4) simulação de desempenho; (5) análise de resultados; e (6) comparação entre FMCC com implementações fixas e reconfiguráveis (FMCCR).

Qualquer função coletiva pode ser reconfigurável, mas para simplificar as simulações, selecionamos uma função representante de cada classe de cardinalidade. MPI_Bcast(1xN), MPI_Reduce(Nx1) e MPI_Allgather (NxN) foram selecionadas por serem de mais fácil compreensão e implementação. A carga de trabalho simulada é composta por *jobs* paralelos contendo diferentes padrões de comunicação que implementam as funções selecionadas. Os padrões usam primitivas de comunicação ponto-a-ponto de primeiro grau que transmitem mensagens cujos tamanhos são potências-de-dois (1Byte a 256KB). Nas simulações, os tempos de resposta obtidos para cada tamanho de mensagem mantiveram um desempenho relativo constante entre si. Para evitar redundâncias, apresentamos apenas os resultados obtidos com mensagens de 256KB.

As simulações foram executadas em uma ferramenta validada anteriormente em [17] e a carga de trabalho modelada foi verificada através de *logs* e de modelagem analítica. Assim simulamos mais precisamente o comportamento, o funcionamento e o desempenho da carga de trabalho em diferentes arquiteturas paralelas e medimos valores de métricas que são de difícil obtenção em cargas de trabalho reais.

O ambiente simulado foi um aglomerado homogêneo de 16 nós. Cada nó era equipado com um processador Pentium III 0.938GHz e um módulo de memória principal ilimitado (taxa de transferência de 11.146MB/s). Esses valores foram obtidos utilizando o Papi [18]. A rede de interconexão é uma Fast Ethernet com latência (179ì s) e largura de banda (11.0516MB/s) fixas. Os valores foram obtidos com um *benchmark* de *round-trip-time* desenvolvido. A unidade de transmissão da rede tem um *payload* de 1460 Bytes e um cabeçalho de 58 Bytes. Dois tipos de dispositivos de interconexão foram simulados: um barramento e um switch (comutador). Eles permitiam apenas primitivas de comunicação ponto-a-ponto de primeiro grau e não possuíam mecanismos em hardware para broadcast.

4.1. MPI_Reduce vs. RMReduce

No aglomerado com barramento, a redução em Árvore Seqüencial apresentou os melhores resultados e o Anel apresentou os piores (Figura 3). O modelo de barramento privilegia o padrão com o menor número de pacotes (mensagens combinadas). Todos os padrões testados transmitiram N-1 pacotes, mas no Anel, houve uma sobrecarga adicional provocada pela não sobreposição entre as latências de envio e recepção de mensagens subseqüentes. O Seqüencial obteve um desempenho um pouco melhor que a Árvore Binária e que a Binomial porque nestes últimos, existe dependência entre alguns passos. Além disso, as simulações não consideravam os efeitos do armazenamento temporário. Os pacotes eram consumidos assim que uma primitiva de recepção correspondente era emitida no nó destino.

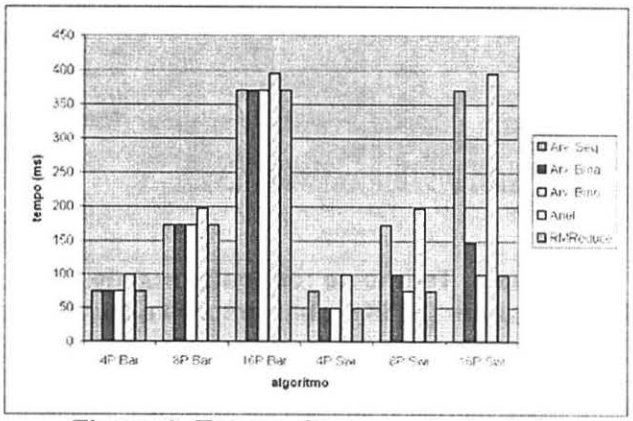

Figura 3. Tempo de resposta (ms) de MPI_Reduce (256KB) em barramento e switch.

No aglomerado com switch, tanto o padrão Seqüencial quanto o Anel obtiveram os mesmos resultados que no barramento, pois a lógica de ambos é seqüencial. Eles não se beneficiam com o paralelismo do switch (Figura 3). A Árvore Binomial apresentou os melhores resultados para todos os tamanhos de mensagens, pois o número de processos comunicantes dobrava a cada passo. Esse padrão se beneficiou com o paralelismo existente no switch, consumindo menos passos lógicos que a Árvore Binária.

O RMReduce se reconfigura, alterando seu comportamento para Árvore Seqüencial e Binomial respectivamente no barramento e no switch. Desta forma, ele apresentou os melhores resultados em nossas simulações. Se tivéssemos uma aplicação paralela que executasse 50% das chamadas a MPI_Reduce no barramento e o restante no switch, a função reconfigurável teria um ganho de 57,8% sobre a função

fixa implementada com o padrão Seqüencial e 0,0013% sobre a implementação com o Binomial.

4.2. MPI_Allgather vs. RMAllgather

No aglomerado com barramento, o padrão Circular e o Pairwise apresentaram o melhor resultado (Figura 4). Eles são logicamente similares e transmitem o menor número de pacotes de tamanho fixo (N^2-N) entre todos os padrões. O Shuffle é concluído em piso($\log_2$ N) passos, mas apresentou resultados um pouco piores que os outros dois porque o tamanho dos pacotes dobra a cada passo. Além disso, como o switch e o barramento não permitem trocas bidirecionais, os passos precisam ser divididos em dois.

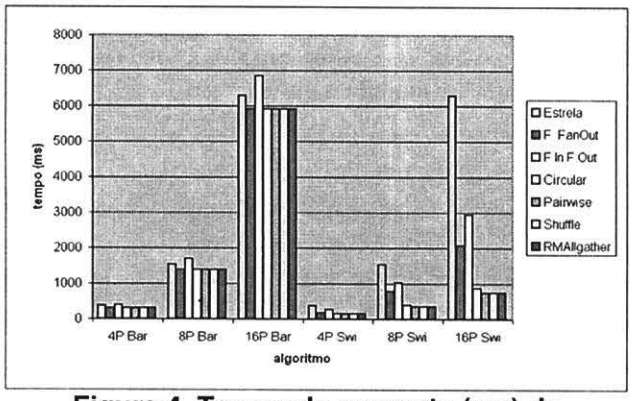

Figura 4. Tempo de resposta (ms) de MPI_Allgather (256KB), barramento e switch.

No switch, o Pairwise foi o melhor porque utiliza os recursos de paralelismo do dispositivo de rede evitando trocas bidirecionais. Ele alcança o desempenho ótimo em 2*(N-1) passos, trocando piso(N/2) pacotes de tamanho T. Neste caso, o Shuffle foi melhor que o Circular. Por exemplo, com 8 processos, o padrão é concluído em 3 passos que foram divididos em seis devido à falta de suporte a trocas bidirecionais (total de 2*piso($\log_2$ N) passos). Também por isto, o Circular consome 2*N passos lógicos em vez de 2*(N-1), como o Pairwise.

No barramento e no switch, o RMAllgather usou uma configuração Pairwise e também apresentou o melhor resultado entre todos os padrões.

4.3. MPI_Bcast vs. RMBcast

No aglomerado com barramento, três padrões apresentaram o melhor resultado para a operação de broadcast (Figura 5). Os padrões: Árvore Seqüencial, Árvore Binária e Árvore Binomial apresentaram o menor tempo de resposta para todos os números de processos e tamanhos de mensagens. Suas N-1 transmissões não foram prejudicadas por dependências. Por outro lado, o Cadeia apresentou o pior resultado devido ao efeito semelhante ao do Anel na redução.

No switch, os padrões Árvore Seqüencial e Cadeia apresentaram os mesmos tempos que no barramento, devido à lógica seqüencial de ambos. Por sua vez, os padrões Árvore Binária e Árvore Binomial tiveram um desempenho melhor que no barramento devido ao paralelismo inerente a ambos. O padrão Binomial foi mais adequado para o paralelismo do switch, apresentando os melhores resultados.

Tanto no barramento como no switch, o padrão Binomial apresentou os melhores resultados, assim, o RMBcast se reconfiguraria para se comportar como ele em ambos os casos.

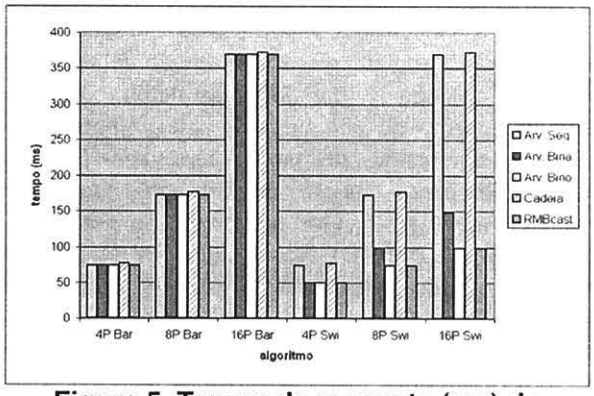

Figura 5. Tempo de resposta (ms) de MPI_Bcast (256KB) em barramento e switch.

5. Resultados da Modelagem Analítica

Nesta seção, apresentamos e analisamos resultados obtidos com a modelagem analítica da função MPI_Bcast, baseada em diferentes padrões de comunicação. Consideramos que eles foram executados com cinco diferentes topologias de redes de 16 nós modeladas (Figura 6). Por motivo de limitação de espaço, consideramos que todos os links dessas redes tinham a mesma latência e largura de banda.

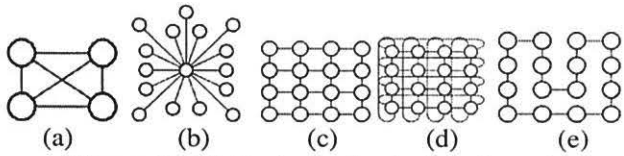

Figura 6. (a) Rede de 4 nós totalmente conectada, (b) Estrela, (c) Malha 2D 4x4, (d) Malha 2D 4x4 com wraparound, (e) Anel.

Na análise, nós comparamos os padrões de comunicação com base no número de saltos entre os

nós da rede. Quando o algoritmo usa uma primitiva de comunicação de primeiro-grau de acordo com sua lógica, um par de processos comunicantes não pode se comunicar com nenhum outro processo. Por outro lado, quando o grau aumenta, um processo pode transmitir seus dados simultaneamente para um ou mais processos, sempre que as conexões de rede o permitirem. Por exemplo, no padrão de Árvore Seqüencial, o raiz envia sua mensagem para todos os outros processos, que esperam bloqueados pelas mensagens. Se a rede permitir a transmissão de uma mensagem de cada vez (topologia com conexões de primeiro grau), a Árvore Seqüencial termina em N-1 saltos (unidades de tempo). Por outro lado, se a rede permitisse que cada processo se comunicasse com todos os outros ao mesmo tempo (grau N-1), a Árvore Seqüencial consumiria uma única unidade de tempo para transmitir todas as mensagens, pois todos os saltos seriam simultâneos. Assim, para cada algoritmo, existe uma primitiva de comunicação com grau ótimo, que nós exploramos a seguir. Destacamos que quando dois saltos podem acontecer em paralelo, com respeito à topologia da rede e à lógica do algoritmo, eles são contados como um único salto, porque consomem uma unidade de tempo.

5.1 Rede Totalmente Conectada

Neste modelo, utilizamos um aglomerado de 16 nós diretamente conectados entre si [15]. A Tabela 1 mostra o número de saltos consumidos por todos os padrões de comunicação em uma rede totalmente conectada usando primitivas de primeiro grau e de grau ótimo em operação de broadcast.

Tabela 1. Número de saltos do MPI_Bcast na rede Totalmente Conectada.

Procs.	Árv. Seq.		Árv. Bina		Arv. Bino		Cadeia		RMBcast	
Grau	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo
8	8	1	4	3	3	3	7	7	3	1
9	9	1	4	3	4	3	8	8	4	1
10	10	1	4	3	4	3	9	9	4	1
11	11	1	5	3	4	3	10	10	4	1
12	12	1	5	3	4	3	11	11	4	1
13	13	1	5	3	4	3	12	12	4	1
14	14	1	5	3	4	3	13	13	4	1
15	15	1	6	3	4	3	14	14	4	1
16	16	1	6	4	4	4	15	15	4	1

Nesta rede, a Árvore Binomial apresentou os melhores resultados com as primitivas de primeiro grau porque explorava o paralelismo da rede totalmente conectada. Por outro lado, o Seqüencial foi o melhor com primitivas de grau ótimo. Como este tipo de rede permite comunicação de um nó para qualquer outro a qualquer momento, o uso de primitivas de primeiro grau respeitando a lógica do algoritmo apenas atrasou as transmissões. No entanto, com primitivas de grau ótimo, ele se tornou um algoritmo de inundação e teve um desempenho melhor que o dos outros.

Nesta rede, o RMBcast se reconfiguraria para se comportar como Árvore Seqüencial usando primitivas de grau ótimo, apresentando os melhores resultados.

5.2 Rede em Estrela

Na rede em Estrela [15] [16], há N-1 processadores interconectados por um nó central através dos quais as mensagens são encaminhadas. Para tirar vantagem da topologia, nós posicionamos o processo raiz no nó central.

Tabela 2. Número de saltos do MPI_Bcast na rede em Estrela.

Procs.	Árv. Seq.		Árv. Bina		Arv. Bino		Cadeia		RMBcast	
Grau	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo
8	7	1	12	11	11	9	13	13	7	1
9	8	1	14	13	12	9	15	15	8	1
10	9	1	16	15	14	11	17	17	9	1
11	10	1	18	17	16	13	19	19	10	1
12	11	1	20	19	18	15	21	21	11	1
13	12	1	22	21	20	17	23	23	12	1
14	13	1	24	23	22	19	25	25	13	1
15	14	1	26	25	24	21	27	27	14	1
16	15	1	28	27	26	23	29	29	15	1

Na Tabela 2, vemos que o padrão de Árvore Seqüencial também foi o melhor, porque ele tem a maior semelhança com a topologia da rede entre os padrões analisados. Assim, o RMBcast se reconfiguraria para se comportar como Árvore Seqüencial, obtendo os melhores resultados.

5.3 Rede em Malha 2-D 4x4

Na malha modelada sem wraparound Figura 6c [15] [16], os nós podem ter graus: 2 (cantos), 3 (canto do meio) ou 4 (meio). Os nós estão dispostos como apresentado na Figura 6c e os processos estão alocados seqüencialmente da esquerda para a direita e do topo para baixo (mas poderia ser diferente). Também modelamos outra versão de malha com wraparound, onde todos os nós têm grau 4 (Figura 6d).

No primeiro modelo de rede, a Árvore Binomial foi o melhor algoritmo (Tabela 3a). Destacamos que com 8 processos, o padrão Cadeia teve desempenho melhor que o outro e para alguns números de processos, os dois apresentaram o mesmo desempenho. Assim, a Árvore Binomial não apresentou o melhor desempenho em todas as situações nesta rede. Para obter o melhor resultado, o RMBcast se comportaria como o melhor padrão para cada número de processos.

Tabela 3. Número de saltos do MPI_Bcast em malha 2-D (a) sem e (b) com wraparound.

Procs.	Árv. Seq.		Árv. Bina		Arv. Bino		Cadeia		RMBcast	
Grau	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo
8	16	16	11	11	11	11	10	10	10	10
9	20	20	14	14	13	13	14	14	13	13
10	22	22	16	16	14	14	15	15	14	14
11	25	25	19	19	16	16	16	16	16	16
12	29	29	21	21	17	17	17	17	17	17
13	34	34	22	22	18	18	21	21	18	18
14	37	37	25	25	19	19	22	22	19	19
15	42	42	28	28	21	21	23	23	21	21
16	48	48	30	30	22	22	24	24	22	22

(a)

Procs.	Árv. Seq.		Árv. Bina		Arv. Bino		Cadeia		RMBcast	
Grau	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo
8	12	12	10	10	9	9	8	8	8	8
9	14	14	13	13	11	11	10	10	10	10
10	17	17	15	15	12	12	11	11	11	11
11	21	21	18	18	14	14	12	12	12	12
12	24	24	20	20	15	15	13	13	13	13
13	27	27	21	21	16	16	15	15	15	15
14	31	31	24	24	17	17	16	16	16	16
15	36	36	27	27	18	18	17	17	17	17
16	40	40	29	29	19	19	18	18	18	18

(b)

Com wraparound, o Cadeia apresentou os melhores resultados, assim, o RMBcast se configuraria como Cadeia (Tabela 3b). Neste tipo de rede é possível utilizar outros algoritmos que obteriam um desempenho melhor que o do Cadeia. Por exemplo, na mesma malha, um padrão de comunicação por inundação [19] consumiria 4 saltos para alcançar os 16 processos, sendo melhor que todos os padrões analisados. O RMBcast deveria incorporar o novo padrão para obter os melhores resultados.

5.4 Rede em Anel

No modelo de rede em Anel unidirecional não-segmentado [16], cada nó está conectado a outros dois (grau 2). Assim, as primitivas utilizadas eram de primeiro grau (grau ótimo neste caso) e o melhor padrão foi o Cadeia. Os outros padrões requeriam roteamento adicional para que os nós que não se alcançavam diretamente pudessem se comunicar.

Tabela 4. Número de saltos do MPI_Bcast em rede em Anel.

Procs.	Árv. Seq.		Árv. Bina		Arv. Bino		Cadeia		RMBcast	
Grau	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo	1°	ótimo
8	28	28	16	16	12	12	7	7	7	7
9	36	36	18	18	20	20	8	8	8	8
10	45	45	23	23	21	21	9	9	9	9
11	55	55	29	29	23	23	10	10	10	10
12	66	66	37	37	24	24	11	11	11	11
13	78	78	38	38	28	28	12	12	12	12
14	91	91	45	45	29	29	13	13	13	13
15	105	105	52	52	31	31	14	14	14	14
16	119	119	60	60	32	32	15	15	15	15

Na árvore Binomial, cada processo tende a transmitir seus pacotes para processos que estão próximos. Por isto ele obteve desempenho melhor que o Seqüencial e que a Árvore Binária. Na rede em Anel, o RMBcast se reconfiguraria para se comportar como Cadeia usando primitivas de primeiro grau para obter o melhor resultado (Tabela 4).

6. Conclusões

Os resultados mostraram que o melhor padrão para uma dada situação é aquele cuja topologia lógica melhor se ajusta à topologia física da rede.

Os resultados de simulação do MPI_Reduce mostraram que a reconfiguração pode gerar ganhos de desempenho. Por outro lado, com os resultados, do MPI_Bcast e do MPI_Allgather, seria possível concluir de forma errada que um único algoritmo invariável forneceria sempre os melhores resultados. Na verdade em outras situações, pode haver mais de uma solução melhor que o algoritmo invariável. Isto fica mais evidente quando comparamos os resultados da simulação com os da modelagem analítica. A Árvore Binomial sempre apresentou os melhores resultados nas simulações de broadcast, mas em certas topologias, apresentou resultados piores que o padrão Cadeia (o pior de todos nas simulações). Assim, foi possível perceber que um algoritmo invariável que é o melhor em média pode ser pior que um ou mais algoritmos em outras situações.

Por exemplo, se os dispositivos de rede utilizassem um mecanismo de broadcast em hardware, seria mais adequado utilizá-lo do que usar algoritmos baseados em um conjunto de primitivas de comunicação ponto-a-ponto. O RMBcast se reconfiguraria para incorporar essa característica e apresentar um desempenho melhor que o do melhor algoritmo na média. Algo semelhante aconteceria se o hardware de rede fosse dinamicamente reconfigurável (como em [12]).

Em um caso em que a rede tivesse características como: uma alta taxa de erros ou escassez de buffers e recursos de armazenamento, o melhor algoritmo provavelmente utilizaria as técnicas de segmentação de mensagens. Por outro lado, se a rede tivesse uma alta latência, o uso da combinação (agrupamento) de mensagens melhoraria o desempenho.

É importante ressaltar que neste artigo, não utilizamos os padrões de comunicação mais otimizados e atuais que existem. O nosso objetivo é mostrar os benefícios da reconfiguração através do uso de padrões de comunicação simples comumente empregados em operações coletivas. Lembramos que os padrões mais complexos, híbridos, otimizados estáticos e dinâmicos

podem ser utilizados da mesma forma que os padrões simples no algoritmo reconfigurável.

Sumarizando as idéias expostas nas seções anteriores, concluímos que os nossos objetivos e metas foram alcançados. Neste trabalho, um conjunto de FMCCR (Funções MPI de Comunicação Coletiva Reconfiguráveis) foi proposto, apresentado, simulado, modelado analiticamente, verificado e analisado.

A principal **contribuição** é o conjunto de FMCCR, que obteve melhores resultados que as funções fixas em todos os casos testados. Em algumas situações, houve ganhos consideráveis de desempenho.

A principal vantagem do algoritmo reconfigurável é a sua capacidade de utilizar configurações que fornecem desempenho otimizado para os diferentes sistemas computacionais onde é executado. Sua principal desvantagem está ligada à camada CCC, que pode apresentar grande custo e complexidade computacional ou consumir uma grande quantidade de memória dependendo das variações do sistema, exigindo simplificações.

Os resultados apresentados no artigo são parte de uma pesquisa em andamento e mostraram que a reconfiguração de algoritmos realmente gera ganhos de desempenho e de flexibilidade em funções MPI de comunicação coletiva.

Como **trabalhos futuros**, destacamos: a aplicação de reconfiguração em outras camadas arquiteturais, adicionar mais quadros à proposta de FMCCR, explorar outras configurações de quadros, implementar um framework de suporte para a nossa proposta, adicionar uma Camada de Controle de Configurações inteligente e avaliar a sobrecarga de reconfiguração gerada por essa camada em diferentes situações.

7. Referências

[1] Hwang, K., Xu, Z., *Scalable Parallel Computing: Technology, Architecture, Programming*, McGraw-Hill, 1998.

[2] MPI Forum. "*The MPI Message Passing Interface Standard*", Technical Report, the University of Tennessee, 1994.

[3] M.X.T. Delgado, "Soluções Eficientes para Operações de Comunicação Coletiva em Aplicações Paralelas em Aglomerdos de Computadores", tese de doutorado, Engenharia Elétrica e Eletrônica, USP, São Paulo, 1999.

[4] L.P. Huse, "*Collective Communication on Dedicated Clusters of Workstations*", EuroPVMMPI, 1999,pp.469-476.

[5] W.B. Tan, P. Strazdins, "*The Analysis and Optimization of Collective Communications on a Beowulf Cluster*", ICPaDS '02, 2002, pp. 659-666.

[6] S.S. Vadhiyar, G.E. Fagg, J. Dongarra, "*Automatically Tuned Collective Communications*", SC 2000, CD-ROM, ISBN 0-7803-9802-5, 2000.

[7] R. Thakur, W. Gropp, "*Improving the Performance of Collective Operations in MPICH*", Euro PVM/MPI, 2003. pp. 257-267.

[8] K. Compton, S. Hauck, "*Reconfigurable Computing: A Survey of Systems and Software*", ACM Computing Survey, 34(2), 2002. pp. 171-210.

[9] Martins, C. A. P. S., Ordonez, E. D. M., Corrêa, J. B. T., Carvalho, M. B., "Computação Reconfigurável: Conceitos, Tendências e Aplicações", XXII JAI, 2003. pp.339-388.

[10] L. F. W. Góes, C. A. P. S. Martins, "*Reconfigurable Gang Scheduling Algorithm*", Workshop on Job Scheduling Strategies for Parallel Processing, LNCS, 2004.

[11] H.C. Freitas, "*Proposta e Desenvolvimento de um Processador de Rede com Chave Crossbar Reconfigurável*", dissertação de mestrado, PPGEE, PUC-MG, 2003.

[12] L. Murray, D. Carrington, P. Strooper, "*An approach to specifying software frameworks*", 27th Conference on Australasian Computer Science, v.26, 2004. pp. 185-192.

[13] L.P. Huse, O.W.Saastad, "*The network agnostic MPI - Scali MPI Connect*", Euro PVM/MP, 2003. pp 294-301.

[14] J.M. Squyres, A. Lumsdaine, "*A Component Architecture for LAM/MPI*", Euro PVM/MPI, 2003. pp. 379-387.

[15] Roosta, S.H., *Parallel Processing and Parallel Algorithms-Theory and Computation*,Springer-Verlag, 1999.

[16] Quinn, M.J., *Parallel Compuing – Theory and Practice*, McGraw-Hill, 2nd edition, 1994.

[17] L. F. W. Góes, L.E.S. Ramos, C. A. P. S. Martins, "*ClusterSim: A Java-Based Parallel Discrete-Event Simulation Tool for Cluster Computing*", Cluster 2004.

[18] S. Browne, C. Deane, G. Ho, P. Mucci, "*PAPI: A Portable Interface to Hardware Performance Counters*", Department of Defense HPCMP Users Group Conference, 1999.

[19] M. Barnett, et al. "*Building a High-Performance Collective Communication Library*". SC, 1994. pp. 107-116.