Construção de um Mecanismo de Comunicação para Ambientes de Processamento de Alto Desempenho *

Evandro Clivatti Dall'Agnol † Lucas Correia Villa Real ‡ Daniela Saccol Peranconi §
Marcelo Augusto Cardozo Junior ¶ Gerson Geraldo H. Cavalheiro

{ecd,lucasvr,danielap,mcardozo,gersonc}@exatas.unisinos.br

Universidade do Vale do Rio dos Sinos
Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
Av. Unisinos, 950
São Leopoldo - RS

1. Resumo

O uso de aglomerados de computadores para o Processamento de Alto Desempenho (PAD) é uma alternativa bastante viável em relação aos custos de hardware. Contudo, surgem dificuldades na programação distribuída como a preocupação com a boa utilização dos recursos disponíveis neste tipo de arquitetura. Muitos trabalhos são desenvolvidos com o objetivo de suprir a necessidade de ferramentas que facilitem o trabalho do programador. Paradigmas como *Remote Procedure Call* e mecanismos de comunicação como Mensagens Ativas provêem características interessantes para o desenvolvimento de novas ferramentas para a programação paralela e distribuída. Uma biblioteca de comunicação é desenvolvida nesse contexto com o objetivo de ser integrada a Anahy, um ambiente de programação para PAD. Resultados de desempenho demonstram a boa escalabilidade da implementação realizada.

2. Introdução

A exploração eficiente do Processamento de Alto Desempenho (PAD) em aglomerados de computadores (também chamados *clusters*) passa necessariamente por bons mecanismos de comunicação. A eficiência na exploração da rede utilizada na comunicação interna do aglomerado é fator de grande peso no bom aproveitamento dos recursos de PAD disponíveis. Um paradigma para a implementação desses mecanismos é o RPC (*Remote Procedure Call*), que provê a execução remota de procedimentos utilizando dados providos pelo usuário, ou por outro procedimento, e retorna seus resultados. As implementações de RPC disponibilizam Interfaces de Programação Aplicativa (API's) com o objetivo de aumentar o nível de abstração na programação das aplicações e facilitar o uso do processamento de alto desempenho em ambientes de memória distribuída. As abstrações dizem respeito principalmente ao endereç amento de dados e funções em ambientes com esta configuração de memória.

No entanto, cabe considerar que o mecanismo de RPC foi desenvolvido com foco em arquiteturas distribuídas, tendo como objetivo primeiro oferecer uma interface de programação que facilite o desenvolvimento de aplicações nestas arquiteturas. Um mecanismo de comunicação que implementa de forma eficiente o paradigma RPC é o de Mensagens Ativas, tendo como foco de atenção ambientes de exploração do processamento de alto desempenho. A característica principal deste mecanismo é descrever juntamente com a mensagem a ser enviada a identificação de um serviç o a ser executado na sua chegada ao seu destino. Uma mensagem ativa contém a identificação de uma função correspondente ao serviç o a ser executado e os dados a serem manipulados por esse serviç o. No contexto de Mensagens Ativas, RPC provê a ativação remota de serviç os e comunicação de dados com independência da localização

* Projeto Anahy – CNPq (55.2196/02-9). Este trabalho foi parcialmente desenvolvido em colaboração com a HP Brasil P&D
† ITI - CNPq
‡ ITI - CNPq
§ Bolsista PROSUP/CAPES, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
¶ Programa Interdisciplinar de Pós-Graduação em Computação Aplicada

destes dados.

É citado em [13] que soluções existentes focam em eficiência pura no hardware, mais precisamente em processadores, e deixam as comunicações em segundo plano, negligenciando a relação entre processamento e a troca de dados. Um mecanismo de Mensagens Ativas pode executar eficientemente em arquiteturas baseadas em trocas de mensagens.

Este trabalho enfoca a inclusão, em nível aplicativo, de um mecanismo de Mensagens Ativas em Anahy, um ambiente de Processamento de Alto Desempenho em aglomerados. Especial atenção é dada à interação entre o mecanismo de comunicação proposto e o Núcleo de Execução *multithread* de Anahy.

O presente texto possui seções que discorrem sobre o conceito de Mensagens Ativas, sobre seu uso e aplicação em diferentes ambientes para o processamento de alto desempenho e sobre sua aplicação específica em Anahy. Comenta-se também sobre o escalonamento de Anahy e seus serviç os de comunicação. Finalmente, são apresentadas algumas medidas de desempenho e conclusão.

3. Mensagens Ativas

Mensagens Ativas são um mecanismo de comunicação assíncrono com o objetivo de explorar eficientemente os recursos de comunicação disponíveis. A idéia principal é que cada mensagem contém dados especificando a necessidade de execução de um serviç o bem como os parâmetros necessários a sua execução. Este serviç o, no receptor,é executado no momento do recebimento de uma mensagem. Cada mensagem contém em seu cabeç alho informações sobre uma função tratadora definida pelo usuário, cuja responsabilidade é retirar a mensagem da rede e inseri-la na computação existente para que o serviç o identificado possa ser executado. A função tratadora deve concluir sua execução o mais rapidamente possível.

Um aspecto de grande importância no mecanismo de Mensagens Ativas implementado em hardware [13] é a sobreposição de comunicação e computação, que ocorre quando o remetente envia mensagens para a rede e continua executando suas tarefas enquanto os dados trafegam até seus receptores. Na chegada da mensagem, o receptor é notificado ou interrompido, lanç ando imediatamente a função tratadora. Com este conjunto de características respeitadas, as Mensagens Ativas adquirem propriedades como a preocupação com o menor *overhead* de processamento possível e a não utilização de *buffers* a não ser os necessários pelo sistema operacional. Ou seja, ao ser criada, a mensagem é enviada o mais rapidamente possível e tratada assim que é lida pelo receptor. Nota-se que as mensagens Ativas não consistem de um novo paradigma de programação para ambientes distribuídos ou de uso de memória compartilhada; sua proposta é oferecer um mecanismo de comunicação para implementar estes e outros paradigmas eficientemente [13].

Muito embora, em sua origem, o mecanismo de Mensagens Ativas esteja baseado em exploração efetiva de recursos de hardware, este pode ser facilmente introduzido em nível aplicativo. Neste caso, a introdução de multiprogramação leve (uso de *threads*) pode vir a oferecer o suporte necessário a sobreposição de (parte dos) tempos envolvidos em comunicação com computação efetiva [12].

4. Ambientes de Programação Paralelos e Distribuídos

As subseções a seguir apresentam alguns ambientes de programação paralelos e distribuídos como Split-C, Chant, Nexus e Athapascan-0 e comentários acerca da utilização do conceito de Mensagens Ativas e comunicação nesses ambientes.

4.1. Split-C

Split-C [4] é uma extensão paralela da linguagem de programação C que suporta o acesso a um espaç o de endereç amento global em multiprocessadores com memória distribuída. Ele segue o modelo de programação SPMD (*Single Program Multiple Data*), onde todos os processadores iniciam sua execução em um mesmo ponto de um código comum, mas podem seguir um fluxo de controle distinto, sincronizando suas execuções em pontos pré-definidos com a utilização de barreiras. Uma das funcionalidades de Split-C consite em permitir que qualquer processador acesse qualquer localização em um espaç o de endereç amento global, através de ponteiros globais, e, ao mesmo tempo, possua uma região específica do espaç o de endereç amento global, que pode ser acessada por meio de ponteiros locais padrão da linguagem C.

Em Split-C, as Mensagens Ativas consistem de um recurso de implementação do próprio ambiente, não sendo oferecido acesso a estes recursos ao programador. A camada de Mensagens Ativas utilizada tem seus serviç os disponibilizados pela interface SPMA [3], a qual foi construída para manipular diretamente um hardware de comunicação (adaptador TB2, em máquinas IBM SP).

4.2. Chant

O ambiente para programação paralela Chant [11] foi desenvolvido para sistemas com memória distribuída, estando baseado em *threads* comunicantes (*talking threads*). Este termo foi introduzido para representar a noção de

duas *threads* em comunicação direta, através de primitivas *send/receive*, independente de estarem ou não no mesmo espaço de endereçamento. Chant estende a interface e as funcionalidades do padrão POSIX para *threads* (Pthreads) para uma categoria de objetos computacionais chamados *chanters*, correspondentes às *threads*, que podem estabelecer comunicação remota com outras *threads*. Para criar uma *thread chanter*, local ou remota, a primitiva *pthread_create* de Pthreads foi estendida para *pthread_chanter_create*. A comunicação entre *chanters* é baseada no padrão MPI, utilizando primitivas do tipo *send* e *receive*, para passagem de parâmetros e retorno de resultados, possibilitando o envio síncrono e o recebimento síncrono e assíncrono de mensagens. Para situações onde a comunicação entre *chanters* não pode ser realizada através da troca de mensagens ponto-a-ponto, Chant provê um mecanismo de RSR (*Remote Service Requests*), que permite que funções sejam executadas dentro de um contexto remoto.

O suporte a comunicação em Chant oferece a possibilidade de realizar troca de dados ponto-a-ponto entre *threads* e de invocação remota de *threads*. Este segundo serviço viabilizado pela introdução no ambiente de execução de uma *thread* especializada (*server thread*), a qual aguarda o recebimento de mensagens solicitando a execução de um serviço e, ao receber uma mensagem, passa a executá-lo, possuindo esta *thread* uma prioridade mais alta em relação às demais *threads*. Em Chant, os serviços de Mensagens Ativas estão disponíveis ao programador, o cuidado a ser tomado é que os serviços solicitados devem ser não bloqueantes, de forma a não ocorrer situações de *deadlock*.

4.3. Nexus

Nexus [7, 8] consiste em uma camada de suporte à execução de aplicações paralelas irregulares, que busca possibilitar que aglomerados de computadores geograficamente distribuídos e heterogêneos sejam utilizados como um único aglomerado global (conceito de *Grid Computing*). Neste ambiente, uma aplicação consiste em um conjunto de *threads* executando dentro de espaços de endereçamento denominados *contextos*, que são mapeados para um conjunto de *nodos*, que representam os recursos físicos de processamento. Dentro de um mesmo contexto, as *threads* comunicam-se por meio da memória compartilhada, enquanto que, em contextos diferentes, podem disparar *requisições de serviços remotos* através de *ponteiros globais*, que provocarão a execução de funções dentro de outros contextos. Um ponteiro global é uma estrutura de dados que contém uma referência direta a um objeto dentro de um contexto, que pode ser remoto, disponibilizando, com isso, um espaço de endereçamento global. Para que *threads* pertencentes a contextos diferentes possam comunicar-se são utilizadas as requisições de serviços remotos (RSR). Uma RSR resulta na execução de uma função especial, chamada *handler*, no contexto referenciado pelo ponteiro global. Esta função é invocada assincronamente dentro do contexto; nenhuma ação, tal como a execução de um *receive*, precisa ser tomada para que a função seja executada. Esta execução pode ocorrer de dois modos: em uma nova *thread* ou em uma *thread* pré-alocada, caso a função não seja bloqueante. Cabe salientar que não existe confirmação ou retorno de resultados para uma RSR e a *thread* chamadora não permanece bloqueada.

4.4. Athapascan-0

O núcleo executivo de Athapascan-0 [9, 1] realiza a integração de multiprogramação e comunicação, fornecendo um conjunto de primitivas para a realização de troca de mensagens. Athapascan-0 possibilita a criação de *threads* local e remotamente. No caso de serem criadas localmente, estas comunicam-se por meio de memória compartilhada e o núcleo oferece mecanismos de sincronização (como *mutexes*, semáforos e variáveis de condição) para garantir o acesso concorrente aos dados compartilhados. No caso de *threads* remotas, sua criação ocorre através de uma chamada a um procedimento remoto assíncrono, correspondendo à execução de um serviço.

Além de permitir a criação remota de *threads*, as quais não possuem nenhuma restrição no que diz respeito ao uso de primitivas de sincronização, Athapascan-0 introduz a possibilidade de uso de "mensagens urgentes". Mensagens urgentes são tratadas por um *deamon* especializado, o qual reage ao recebimento de uma mensagem pela ativação imediata do serviço solicitado – estes serviços não devem realizar nenhuma operação de sincronização.

4.5. Uso de Mensagens Ativas

A bibliografia apresenta dois casos típicos de uso de Mensagens Ativas, ou sob a forma de uma camada interna a um ambiente de processamento (como no caso de Split-C) ou como uma facilidade na interface de programação (Chant, Nexus, Athapascan-0). No entanto, o grande diferencial é viabilizar uma estrutura de comunicação que possa de um lado tirar benefícios de desempenho de comunicações assíncronas, oferecendo facilidades de encadeamento nas comunicações; e de outro, não havendo a possibilidade de retorno dos serviços invocados, o retorno dos dados pode ser dar através de uma nova invocação de serviço oferecendo os resultados de uma computação.

5. Mensagens Ativas em Anahy

Para a implementação do protótipo distribuído do ambiente Anahy[2] foi desenvolvida uma biblioteca de comunicação[5] que implementa o conceito de Mensagens Ativas. Esta biblioteca faz uso do serviç o de *sockets* do sistema operacional GNU/Linux e sua unidade básica de manipulação é a Mensagem Ativa, descrita na Seção 3.

Sua API disponibiliza chamadas para, entre outras ações, criação, empacotamento e envio de mensagens como act_msg_create(), act_msg_pack() e act_msg_send(). Para o registro de serviç os que o sistema usuário das Mensagens Ativas deseja disponibilizar, há chamadas como act_msg_regserv(). Para manipulação direta dos *sockets* Linux e estabelecimento de conexões entre os nodos do aglomerado foi desenvolvida uma sub-camada com o objetivo de modularização de código facilitando sua manutenção, entendimento e aperfeiç oamento, bem como portabilidade de código para uso de outros recursos de comunicação que forneç am serviç os análogos aos *sockets* Linux. Critérios como facilidade de uso e o padrão POSIX são considerados na elaboração destas primitivas e do modelo de comunicação implementado. A atenção a estes critérios e o uso de Mensagens Ativas derivam dos conceitos de portabilidade de código e desempenho descritos em [6].

A Figura 1 mostra a interação da biblioteca implementada com o ambiente Anahy. O módulo de Mensagens Ativas é responsável por fazer a interface entre Anahy e os recursos de comunicação do sistema operacional. Sendo Anahy um ambiente de programação e execução paralelo, ele se comunica diretamente com o sistema operacional para a realização de seus serviç os.

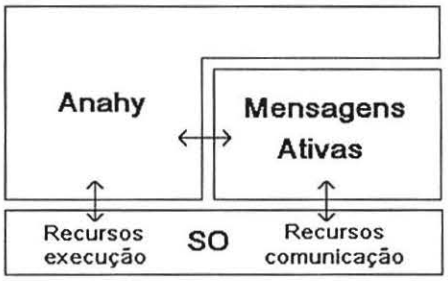

Figura 1. Localização do módulo de Mensagens Ativas na estrutura de Anahy

Em Anahy as Mensagens Ativas foram introduzidas para oferecer suporte a implementação do próprio ambiente de processamento. O conjunto de serviç os oferecidosé especializado em realizar as operações internas necessárias ao controle da execução do programa aplicativo, conforme discutido nas próximas seções.

6. Escalonamento em Anahy

Para usufruir dos mecanismos de mensagens ativas, o ambiente Anahy precisa ser capaz de comunicar-se entre os diferentes nodos usados no aglomerado virtual. Desta forma, ele possibilita que hajam execuções de tarefas em nodos remotos. O mecanismo interno ao ambiente responsável por atender às demandas de mensagens ativas é o escalonador Anahy.

O escalonamento em Anahy é baseado em um algoritmo de listas[10], de forma a explorar eficientemente um *grafo de dependências*. Para atingir tal eficiência o grafo é percorrido em profundidade e em ordem lexicográfica. O grafo é formado pela conexão entre as tarefas, com o objetivo de resolver a cadeia de dependências entre elas, como pode ser visto na Figura 2. Tendo como base o grafo de tarefas, o núcleo executivo de Anahy tem sua funcionalidade ligada diretamente a duas operações: atribuição de tarefas aos processadores e controle da dependência de dados.

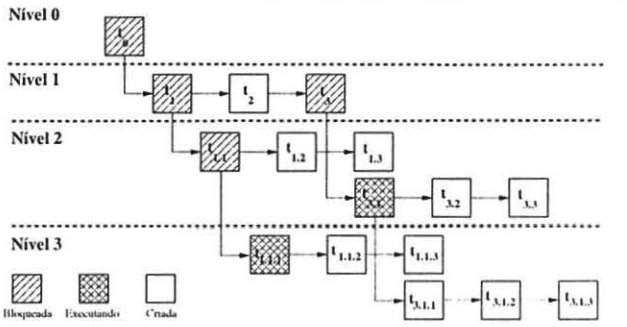

Figura 2. Grafo de dependências entre as tarefas Anahy

Desta forma obtém-se a localidade dos dados a cada processador envolvido na execução, pois no momento em que cada um inicia a execução de uma tarefa, potencialmente tem em mãos a execução de toda uma sub-árvore contendo tarefas geradas a partir desta primeira.

O escalonador utiliza-se deste grafo para obter uma ordem de execução que maximize a eficiência, evitando executar tarefas que irão bloquear esperando o término de outra. A decisão da ordem de execução sempre é tomada quando uma nova tarefa a ser executada é buscada, ou seja, ela sempre é calculada a cada busca na lista de tarefas prontas a serem executadas.

A operacionalidade deste conjunto de serviç os em um ambiente com memória compartilhada pode ser facilmente

modelado, considerando que todos os processadores compartilham o acesso à integridade do grafo de dependências sobre um mesmo espaç o de endereç amento. Esta mesma facilidade de modelagem pode ser obtida sobre um ambiente com memória distribuída, considerando o mapeamento dos serviç os realizados sobre recursos de comunicação adequados.

A próxima seção (Seção 7) comenta os serviç os de comunicação necessários para a realização do escalonamento em ambientes com memória distribuída, tomando como exemplo o ambiente Anahy. Em decisões que envolvam migrações de tarefas e através desses serviç os, o módulo de escalonamento potencialmente fará uso da implementação de Mensagens Ativas realizada.

7. Serviç os de Comunicação em Anahy

Anahy, através da implementação de Mensagens Ativas, faz uso de serviç os de comunicação que possibilitam a troca de tarefas e dados entre os nodos participantes do aglomerado de computadores utilizado. As trocas de tarefas realizadas são reflexos das decisões de escalonamento, onde o *runtime* de Anahy avalia os pedidos de migração de tarefas realizados entre os nodos. Para tanto foram implementados serviç os específicos de *i)* requisição de trabalho, *ii)* envio de trabalho e *iii)* envio de dados. Também foram implementados serviç os de troca de dados que podem ocorrer nas chamadas ao *Join* de Anahy, como *iv)* requisição de dados e *v)* retorno de dados.

A Figura 3 representa o serviç os de comunicação de Anahy comentados nas subseções a seguir.

Figura 3. Serviços de comunicação utilizados por Anahy

7.1. Requisição de Trabalho

A requisição de trabalho ocorre quando um nodo do aglomerado termina a execução das tarefas de sua lista local e sinaliza os outros nodos de que está ocioso e disponível para executar novas tarefas, contribuindo assim para o término da tarefa global ao aglomerado em menor tempo. Consiste de uma mensagem ativa contendo informações como origem, destino e código identificando a requisição de trabalho. Este serviç oé identificado na Figura 3 através do nodo **Nx** requisitando trabalho para o nodo **Ny**.

7.2. Envio de Trabalho

O envio de trabalho ocorre em resposta à requisição de trabalho descrita na Seção 7.1. A Figura 3 representa esta resposta com a comunicação do nodo **Ny** para o nodo **Nx** enviando trabalho como resposta à requisição anterior. O nodo que decidir migrar alguma de suas tarefas responde ao nodo requerente enviando uma mensagem ativa contendo informações confirmando ou negando a existência de trabalho a ser migrado naquele momento. Caso haja trabalho, a descrição deste, juntamente com os dados a serem manipulados, são enviadas nesta mesma resposta.

7.3. Envio de Dados

O envio de dados é realizado pelo nodo que, anteriormente, havia requisitado trabalho a outro nodo conforme descrito na Seção 7.1 e representado na Figura 3 pela comunicação do nodo **Nx** para o nodo **Ny**. Consiste na sinalização do nodo pai da tarefa (representado na Figura 3 por **Ny**) executada localmente comunicando o término de sua execução e lhe enviando os dados resultantes do trabalho realizado. Esta comunicação é efetuada através de uma mensagem ativa com identificação do serviç o de envio de dados e os dados propriamente ditos.

7.4. Requisição de Dados

Este serviç oé requisitado no momento em que um fluxo de execução necessita de dados produzidos por outro fluxo de execução, podendo estar no nodo local ou em um nodo remoto. O serviç o de requisição de dados implementa o conceito de *join*. E acrescenta à este conceito transparência de localização ao usuário programador no sentido de que ele não necessita preocupar-se com o local em que esses dados serão buscados, mas apenas com a identificação do fluxo em que eles foram manipulados. Na Figura 3 este serviç oé representado no sentido do nodo **Nx** para o nodo **Ny**.

7.5. Retorno de Dados

O serviç o de retorno de dados ocorre em resposta à requisição descrita na Seção 7.4. O nodo requisitado a retornar os dados de uma determinada tarefa verifica a existência desta tarefa e se seus dados resultantes já estão disponíveis para *join*. Caso estejam disponíveis, são enviados por meio de uma mensagem ativa para o requerente.

Caso não estejam, o nodo aguarda até o término da tarefa para então enviar os dados ao seu destino. Este serviç o também é representado na Figura 3 no sentido do nodo **Ny** para o nodo **Nx**.

8. Medidas de Desempenho

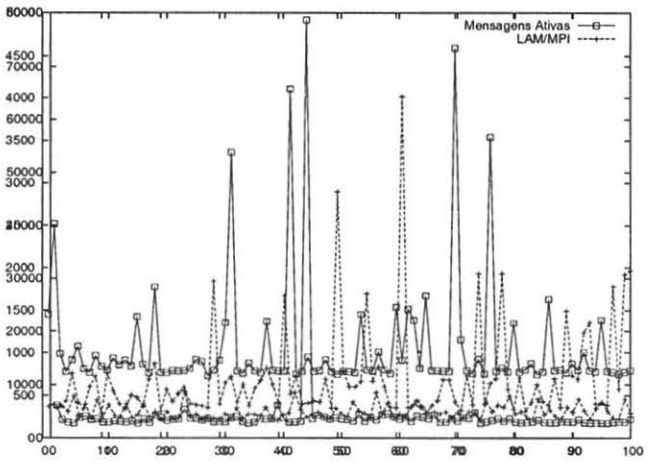

Figura 4. Tempos em micro-segundos para transmissões de array de 3.000 elementos

Com a finalidade de verificar o desempenho obtido com a biblioteca de mensagens ativas incorporada ao ambiente Anahy, foram implementadas aplicações para realizar a transferência de arrays de tamanhos pré-definidos entre dois *endpoints*. Os testes consistem em medir o tempo de envio destes arrays para um *endpoint* e a resposta de confirmação da recepção deste vetor pelo *endpoint* ao nodo que o enviou (*ACK*).

Para isto conexões síncronas foram utilizadas, com a medida de tempo inicial sendo feita antes do empacotamento dos dados (via `gettimeofday()`). O processo então consiste em aguardar o envio destes dados para o outro *endpoint*, que desempacota os dados e empacota um byte (enviando-o em seguida), confirmando a recepção dos dados. Assim que o remetente recebe este pacote com a confirmação, ele o desempacota, verifica se realmente é um *ACK* e encerra a medida de tempo, com outra chamada ao `gettimeofday()`.

Para fins de comparação, a mesma implementação foi realizada sobre a biblioteca LAM/MPI, na versão 7.0.2. Ela é uma implementação do padrão da interface para envio de mensagens (MPI), utilizada freqüentemente para fins

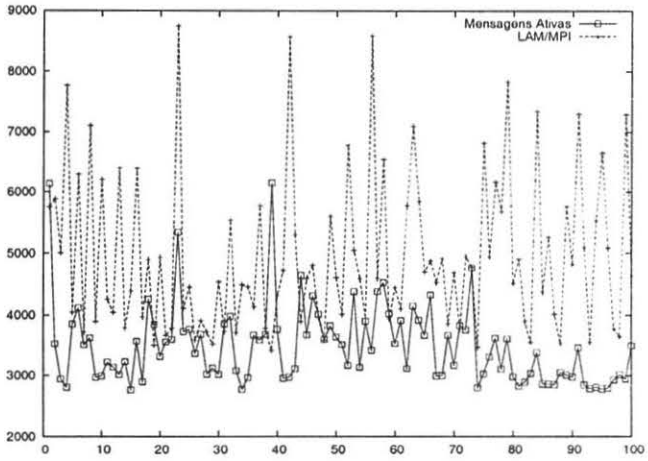

Figura 5. Tempos em micro-segundos para transmissões de array de 30.000 elementos

de processamento de alto desempenho em aglomerados de computadores.

Para desconsiderar a latência da rede e minimizar possíveis perdas e retransmissões de pacotes pelo TCP, a comunicação entre os dois *endpoints* foi realizada no mesmo nodo, sob a forma de processos concorrentes. Foram realizadas 100 transferências de arrays de inteiros de 3.000, 30.000 e 300.000 posições. Os tempos medidos podem ser constatados nas Figuras 4, 5 e 6. As execuções foram tomadas em uma máquina mono-processada Pentium III 1.1GHz, com 512MB de RAM. A unidade de medida está representada em micro-segundos.

Nestas execuções, salientam-se os resultados apresentados na Figura 4. A grande diferenç a de tempo de execução para as diferentes iterações são justificadas pelo escalonamento dos processos envolvidos pelo sistema operacional. Por serem processos concorrentes e pelas baixas unidades de tempo envolvendo a transferência, o tempo levado para escalonar os processos apresenta uma diferenç a de tempo notável quando considerados os tempos da transferência de dados em si, que são na unidade de micro-segundos. Por serem chamadas bloqueantes, o kernel pode ainda colocar os processos para dormir até que seja possível realizar a transferência de dados da interface para o aplicativo (e vice-versa), agregando uma penalidade nas medições finais.

Os resultados vistos nas Figuras 5 e 6, onde são utilizados arrays de tamanhos maiores, mostram que a biblioteca de Mensagens Ativas desenvolvida obteve vantagens de desempenho sobre a biblioteca LAM/MPI. Estes resultados eram esperados pelo fato de de se utilizar a camada de *sockets* do sistema operacional no desenvolvimento da bi-

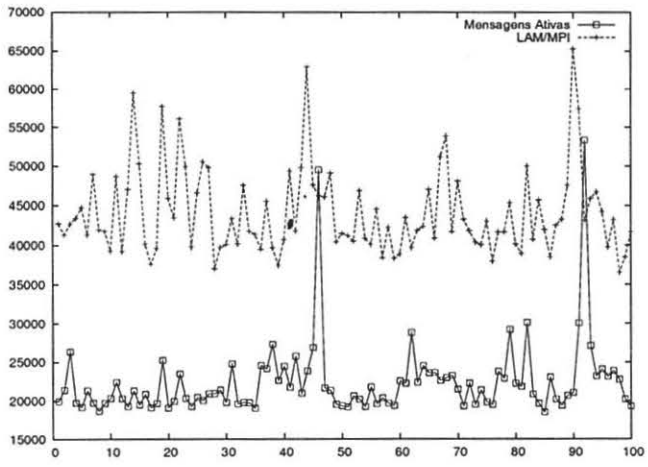

Figura 6. Tempos em micro-segundos para transmissões de array de 300.000 elementos

blioteca em questão. Estas vantagens, contudo, justificam o desenvolvimento de uma nova biblioteca com o objetivo de explorar os recursos de comunicação utilizando uma interface de programação de alto nível.

9. Conclusão

Este trabalho apresentou uma análise sobre os mecanismos de comunicação utilizados por diferentes bibliotecas de programação paralela e de alto desempenho. Estes trabalhos serviram como base para a implementação de uma biblioteca de mensagens ativas para o ambiente Anahy, conforme apresentado na Seção 7.

A biblioteca de mensagens ativas implementada mostrou-se capaz de escalar bem as requisições de transferência de grandes quantidades de dados, com resultados bastante próximos aos obtidos com LAM/MPI para pequenas transferências. Isto permitiu validar o protótipo até então implementado, que deverá manter seu conjunto de operações de manipulação de pacotes de mensagens.

A implementação de mensagens ativas incorporada ao Anahy não possui os elevados custos observados em outras bibliotecas. Estes custos estão geralmente associados aos objetivos de generalização do código para executar em diferentes ambientes, e verificações de tipos de dados em tempo de execução. A biblioteca de mensagens ativas implementada é focada na disponibilização de serviç os de execução, permitindo assim ganhos de desempenho como apresentados na Seção 8.

Entre os trabalhos futuros está a finalização da inclusão da biblioteca de Mensagens Ativas implementada em Anahy.

Referências

[1] J. Briat, I. Ginzburg, and M. Pasin. Athapascan-0b: un noyau exécutif parallèle. *Lettre du Calculateur Parallèle*, 10(3):273–293, 1998.

[2] G. G. H. Cavalheiro, L. C. V. Real, and E. C. Dall'Agnol. Uma biblioteca de processos leves para a implementação de aplicações altamente paralelas. In *IV Workshop em Sistemas Computacionais de Alto Desempenho*, São Paulo, SP, 2003.

[3] C.-C. Chang, G. Czajkowski, C. Hawblitzel, and T. von Eicken. Low-latency communication on the ibm risc system/6000 sp. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, page 24. ACM Press, 1996.

[4] D. E. Culler, A. C. Arpaci-Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. A. Yelick. Parallel programming in split-c. In *Supercomputing*, pages 262–273, 1993.

[5] E. C. Dall'Agnol and G. G. H. Cavalheiro. Biblioteca de comunicação com mensagens ativas. In *IV Escola Regional de Alto Desempenho*, Pelotas, RS, 2004.

[6] E. C. Dall'Agnol, L. C. V. Real, E. D. Benitez, and G. G. H. Cavalheiro. Portabilidade na programação para o processamento de alto desempenho. In *IV Workshop em Sistemas Computacionais de Alto Desempenho*, São Paulo, Brasil, 2003.

[7] I. Foster, C. Kesselman, and S. Tuecke. The Nexus task-parallel runtime system. In *Proc. 1st Intl Workshop on Parallel Processing*, pages 457–462. Tata McGraw Hill, 1994.

[8] I. Foster, C. Kesselman, and S. Tuecke. The Nexus approach to integrating multithreading and communication. *Journal of Parallel and Distributed Computing*, 37(1):70–82, Aug. 1996.

[9] I. Ginzburg. *Athapascan-0b: intégration efficace et portable de multiprogrammation légère et de communication*. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 1997.

[10] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, Mar. 1969.

[11] M. Haines, D. Cronk, and P. Mehrotra. On the design of Chant: A talking threads package. In *Proceedings of Supercomputing*, pages 350–359, Washington D.C., Nov. 1994. ACM/IEEE.

[12] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[13] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: a mechanism for integrated communication and computation. University of California, Berkeley, CA 94720.