HetNOS Threads: Programação Multithreaded Distribuída

Vinicius Mantay, Cláudio Fernando Resin Geyer

Instituto de Informática, Universidade Federal do Rio Grande do Sul,
Avenida Bento Gonçalves 9500, Bloco IV, CP 15064, CEP 91501-970
Agronomia, Porto Alegre, RS, Brasil
{mantay, geyer} @inf.ufrgs.br

Resumo—

O pacote de programação HetNOS Threads visa a união de duas importantes tecnologias de desenvolvimento de aplicações paralelas e distribuídas, que são os pacotes de programação *multithreaded* e a utilização de programação distribuída em redes de computadores com alta velocidade de comunicação.

O modelo de programação adotado tem por objetivo oferecer ao desenvolvedor desta classe de aplicações ferramentas para a exploração de paralelismo tanto de alta quanto de baixa granulosidade. Essas metas são alcançadas com a combinação de um pacote de programação *multithreaded*, as Pthreads, com o sistema operacional de rede HetNOS.

São oferecidas primitivas de comunicação entre processos e *threads* distribuídos via troca de mensagens, manipulação remota destas entidades e variáveis de sincronização distribuídas, de maneira a proporcionar as funcionalidades necessárias ao desenvolvimento de aplicações paralelas distribuídas aliando facilidade de uso, programação intuitiva e desempenho eficiente.

Palavras-chave— **comunicação, programação multithreaded, threads**

I. INTRODUÇÃO

A evolução de duas tecnologias, microprocessadores e redes de computadores, está permitindo uma nova geração de sistemas distribuídos, que busca unir as vantagens de baixos custos de *hardware* com altas velocidades de comunicação entre os nodos, para aproximar-se das máquinas paralelas em desempenho porém sem apresentar seus altos custos.

O novo modelo utiliza suas características para implementar um novo paralelismo em dois níveis. Em primeiro lugar, as novas tecnologias de redes de computadores, como ATM [ONV 95] e Myrinet[PAK 96], tornam possível a construção de *clusters* de computadores com alto desempenho e o primeiro nível de paralelismo, entre os nodos do sistema. Esse paralelismo é de alta granulosidade devido aos altos custos das operações executadas pelos sistemas operacionais, como por exemplo as chamadas de sistema ou o acesso a recursos (manipulação de sinais assíncronos, mecanismos de comunicação, descritores de arquivos) através de estruturas mantidas no interior do núcleo do sistema operacional, à falta de memória fisicamente compartilhada e à decorrente velocidade de comunicação mais baixa, além de mais complexa.

O segundo nível do paralelismo aparece dentro de cada nodo do sistema. Arquiteturas multiprocessadas, também referidas como SMP (*Symmetric Multi Processors*, Multiprocessadores Simétricos) [SUZ 92], possibilitam que múltiplas *threads* de controle contidas em um mesmo processo possam ser executadas paralelamente em diferentes processadores, compartilhando seu espaço de memória. O paralelismo aqui expresso é de baixa granulosidade devido ao baixo custo associado às operações de manipulação das *threads*, como criação, destruição e mudanças de contexto, à capacidade de compartilhamento de recursos, explorado em operações como o controle de comunicações assíncronas e à maior eficiência na utilização do tempo de processador dos processos dentro deste tipo de arquitetura.

A união dessas abordagens permitiria a eficiente exploração de ambos os níveis de paralelismo e a execução de uma grande gama de aplicações que poderiam tirar vantagem dessa facilidade. No entanto existe a dificuldade representada pela impossibilidade de comunicação direta entre *threads* que não compartilham o mesmo espaço de memória e, portanto, não adaptariam-se a ambientes de memória distribuída.

A proposta das HetNOS Threads, aqui descritas, é de, justamente, prover um canal de comunicação entre *threads* e variáveis de sincronização para estas *threads*, mesmo que localizadas em nodos remotos do sistema, através da união do HetNOS (*Heterogeneous Network Operating System*, Sistema Operacional de Rede Heterogêneo) [BAR 94] com o pacote de programação *multithreaded* Pthreads [LEW 98] [PRO 99]. O sistema HetNOS provê a camada de comunicação e de integração da rede distribuída, enquanto o pacote Pthreads disponibiliza toda a funcionalidade das *threads*, seguindo o padrão POSIX [LEV 91] [POS 90]

[SUN 95] de interoperabilidade e portabilidade, características também presentes no próprio HetNOS.

Este artigo apresenta nas seções seguintes a descrição do modelo de programação utilizado pelas HetNOS Threads, seguido de um relatório do estágio atual do processo de implementação do protótipo juntamente com algumas análises preliminares de seus testes de desempenho, finalizando com a apresentação de conclusões e considerações finais até aqui alcançadas.

II. Modelo de Programação.

O ambiente de programação proposto pelas HetNOS Threads baseia-se em uma arquitetura heterogênea formada por uma rede de computadores cooperantes sem compartilhamento de memória entre si. Pode-se descrever esse conjunto como uma arquitetura onde cada nodo da rede HetNOS é considerado uma unidade autônoma, que depende do sistema HetNOS para sua integração e comunicação e das Pthreads para a execução de seu processamento paralelo.

Visando alcançar o melhor desempenho unido à menor complexidade de uso, o paradigma de comunicação adotado é a troca de mensagens entre as entidades que realizam o processamento do sistema, i.e. os processos e as *threads* HetNOS. A comunicação depende do substrato provido pelo sistema HetNOS que garante um espaço de nomes global a todos os nodos da rede e diversas facilidades na forma de primitivas de comunicação ponto-a-ponto, *broadcasting* e comunicação em grupo.

As HetNOS Threads propõem um conjunto de primitivas na forma de uma biblioteca de funções em linguagem ANSI C. Estas são totalmente integradas ao modelo de comunicação existente no sistema, para a comunicação ponto-a-ponto das *threads* utilizadas por processos integrantes de uma rede HetNOS, juntamente com funções que também permitam a individualização, sincronização e cooperação eficiente de *threads* remotas dentro do sistema.

Essas primeiras decisões de implementação foram motivadas pela predominância desse paradigma de comunicação sobre os demais quando analisados diversos exemplos de aplicações distribuídas, pela manutenção da compatibilidade com aplicações existentes, bibliotecas de comunicação de largo uso (MPI [MES 94], p4 [BUT 94], PVM [GEI 94]) e diversos aplicativos de programação assistida [MAL 99], além da complexidade, altos custos e suporte necessários a outros métodos de comunicação como memória compartilhada distribuída [BAL 98] ou ponteiros globais [FOS 96].

Como características de programação adotadas pelas HetNOS Threads mantiveram-se as funcionalidades do pacote de *threads* escolhido, assim como as da camada de comunicação utilizada. As chamadas de criação, manipulação local, suspensão e acesso aos dados locais das *threads* são aquelas definidas no padrão POSIX e presentes no pacote de *threads* Pthreads. As chamadas de comunicação entre processos e máquinas cooperantes, obtenção de informações e dados sobre a rede de máquinas utilizadas, formação de grupos, invocação remota de aplicações e demais operações do gênero também permaneceram inalteradas, mantendo-se as chamadas HetNOS já existentes com total compatibilidade entre as novas aplicações capazes de utilizar *threads* em suas execuções e as aplicações HetNOS desenvolvidas antes das modificações aqui descritas.

A. Primitivas de Programação

As primeiras funções necessárias são as de cadastramento das *threads* como entidades do sistema HetNOS, sempre associadas ao seu processo criador. O cadastramento servirá para a identificação remota das novas *threads* na rede HetNOS em que estiverem sendo inseridas, o que deverá propiciar a sua identificação e interação remota pelos processos e *threads*, não locais do restante do anel HetNOS. O reconhecimento das *threads* remotas é fator fundamental para permitir a comunicação entre essas novas entidades pois é através do cadastramento no Espaço de Nomes que o HetNOS poderá descobrir a real localização da entidade remota e rotear as mensagens enviadas a ela.

Ao ser efetivado o cadastramento da nova *thread* através da chamada

```
thr_init("nome_thread");
```

a *thread* que executou essa chamada passará a ser conhecida na rede HetNOS pelo nome "nome_thread" definido pelo usuário juntamente com demais informações relevantes sobre a identificação e estado atual da *thread*. Da mesma forma será criada uma fila de mensagens associada à nova *thread* onde serão armazenadas as mensagens que venham a ser enviadas com destino a ela e referências a variáveis distribuídas de sincronização compartilhadas pela *thread*.

O envio de mensagens será feito através das mesmas primitivas já existentes no sistema HetNOS. A diferenciação, se a mensagem é destinada a um determinado processo ou a uma certa *thread* será resolvida internamente, já que a identificação de processos e *threads* é única dentro do sistema HetNOS, sem que o usuário deva preocupar-se com esta questão. As primitivas para o envio de uma mensagem de forma assíncrona são

```
h_send("destino", &msg);
```

enquanto que para a comunicação síncrona deve ser utilizada

```
h_sync_send("destino", &msg);
```

O próprio HetNOS deve resolver, através da pesquisa da identificação do destino da mensagem se é o caso de uma comunicação entre processos, entre *threads* ou ambos.

Para o caso do recebimento de mensagens é necessária uma diferenciação no tocante às primitivas utilizadas. Deve-se utilizar a chamada

h_recv("emissor", &msg);

para o recebimento assíncrono de uma mensagem endereçada ao processo e a chamada

h_sync_recv("emissor", &msg);

para utilizar a forma síncrona da função. Porém se o objetivo é a recepção de uma mensagem endereçada à *thread* que faz a chamada deve-se utilizar as primitivas

thr_recv("emissor", &msg);

thr_sync_recv("emissor", &msg);.

É necessária a diferenciação entre as primitivas de recebimento de mensagens devido à possibilidade de ler-se mensagens tanto da fila pertencente ao processo pai da *thread* quanto da fila da própria *thread*. Essa possibilidade advém do fato das *threads* compartilharem a fila de mensagens do processo pai, já que faz parte de seus dados locais, além de possuírem suas próprias filas de mensagens individualizadas.

```
main(argc,argv) {
 h_init();  /* Cadastramento do processo na rede HetNOS */

 h_get_net_config(estacoes, &nro_estacoes); /* Requisição de informações
                                               sobre a rede HetNOS */
 estacao = estacoes[nro_estacoes];
 sprintf(proc_id, "%s_%d", whois, i);
 sprintf(comando,"proc_remoto %d", i);
 if (h_rem_exec(proc_id, estacao, comando) == NULL) { /* Execução remota
                                                         de uma aplicação */
            h_print("erro\n");
            h_terminate(); exit(1); } }

 for (i = 0; i < 8; i++) {
            if((origem = h_recv("ANY", aux)) == NULL) h_print("erro\n");
/* Laço que faz o recebimento das respostas das threads remotas */
            else { /* Tratamento das respostas recebidas */ } }

 h_terminate(); /* Fim das funções HetNOS do processo e de suas threads */
 h_exit("ok"); }

proc_remoto(argc,argv) { /* Processo remoto responsável pela execução das
                                 threads de cálculo  */
 h_init();

 for (i = 0; i < 8; i++) pthread_create(&tid[i], NULL, calcula(i), NULL);

 h_terminate(); h_exit("ok"); }

calcula(int inic) {
 sprintf(nome_thread, "thread_%d", inic);
 thr_init(nome_thread);/* Cadastramento da thread na rede HetNOS */

 /* Cálculo das possíveis soluções */

 pai_id = h_get_parent_name(); /*Função que retorna a id do processo pai*/
 sprintf(saida, solucao);
 if (h_send(pai_id, saida) < 0) { /* Envio da mensagem da thread remota
                                     para o processo coordenador */
            sprintf(output, "%s: Erro enviando mensagem.\n", whois);
            h_syserr(); } }
```

Fig. 1 Aplicação exemplo das HetNOS Threads

Outra importante função a ser explorada pelas HetNOS Threads é a possibilidade de sincronização de *threads* remotas através do uso de variáveis remotas compartilhadas. Variáveis de exclusão mútua distribuídas podem ser utilizadas pelas *threads*, pois são mantidas no espaço de tuplas do sistema HetNOS, tendo por função permitir a cooperação entre *threads* remotas e o acesso compartilhado a recursos do sistema, como arquivos, por exemplo [TAN 92].

A variável é inicializada através da função

thr_mutex_init("nome_mutex");

que criará a variável "nome_mutex" no sistema HetNOS e coordenará o acesso a esta variável das *threads* que desejarem compartilhá-la, através de funções

thr_p("nome_mutex");

thr_v("nome_mutex");

sincronizando o processamento de *threads* que as utilizarem.

Além das primitivas até aqui apresentadas, que são as que têm maior relevância, também existem funções de apoio, como a

thr_display( );

que exibe informações sobre as *threads* ativas do sistema e que não necessitam de maiores apresentações, apenas esta referência à sua existência.

Uma aplicação exemplo do uso das primitivas de programação das HetNOS Threads é apresentada na figura 1. Nesta aplicação um processo coordenador executa remotamente outro processo HetNOS que criará threads trabalhadoras. Ao fim do processamento as threads enviarão seus resultados ao processo coordenador original, localizado em um nodo remoto da rede HetNOS.

Desta forma o modelo de programação apresentado pelas HetNOS Threads caracteriza-se por associar da melhor maneira as propriedades da computação *multithreaded* com a funcionalidade da comunicação remota das *threads* envolvidas no processamento, permitindo ao usuário a utilização de um paradigma simples e poderoso para a programação distribuída.

III. Implementação e Resultados Preliminares

As HetNOS Threads encontram-se parcialmente implementadas. Como já foi citado no capítulo referente ao modelo de programação adotado, a implementação é feita em linguagem de programação ANSI C na forma de uma biblioteca de funções disponível para desenvolvedores de aplicações que utilizem as facilidades providas pelo sistema operacional HetNOS.

Nesta biblioteca encontram-se chamadas de funções que interagem com o núcleo do sistema HetNOS utilizando-se de um esquema cliente-servidor para requisitar seus serviços. Uma vez dentro do núcleo, são efetuadas as operações invocadas pelo usuário, como por exemplo, o cadastramento de uma nova *thread*. Toda operação que demande comunicação com nodos remotos, execução remota de aplicativos, troca de mensagens ou comunicação da situação atual do sistema, passa pelos núcleos HetNOS presentes em cada nodo, sendo transmitidas através de um anel lógico configurado pelo sistema.

A primeira questão de implementação tratada foi a inserção, no sistema operacional HetNOS, da capacidade de ser executado concorrentemente com as *threads* do pacote de programação Pthreads. Ou seja, deveria ser possível a um sistema operacional que não possui a característica de ser *multithreaded* suportar a execução de aplicações com

esta característica que estariam utilizando seus serviços. Essa capacidade é denominada *thread-safe*, significando que operações com *threads* são "seguras" no sentido de que o resultado alcançado por elas seria o correto sem que fosse necessária a inclusão de operações de sincronização dentro do núcleo do HetNOS.

Para alcançar essa qualificação, que não é apresentada pela biblioteca MPI, por exemplo, foi preciso utilizar uma espécie de filtro de acesso ao núcleo, que localiza-se exatamente na biblioteca de funções utilizada pelas aplicações *multithreaded*, onde foram colocadas variáveis de exclusão mútua que disciplinam a utilização de serviços do núcleo HetNOS pelas *threads*. Como conseqüência dessa modificação, múltiplas *threads* de execução podem acessar serviços do sistema operacional HetNOS sem perigo de que seus dados e dos resultados de suas requisições sejam confundidos, resultando em uma execução incorreta dos aplicativos dos usuários.

As questões de implementação e projeto das HetNOS Threads são tratadas em dois níveis de decisão, primeiramente em nível teórico, como exemplifica a discussão apresentada no início do capítulo sobre o modelo de programação adotado. Em outro nível são realizados testes de desempenho de implementações alternativas das distintas partes do projeto, comparando-se seus resultados e lançando mão destes recursos para basear as futuras escolhas sobre a forma final das HetNOS Threads.

Um bom exemplo desta política de desenvolvimento do protótipo é o que se refere à forma de implementação da solução a ser utilizada para garantir a segurança do uso concorrente dos serviços do núcleo HetNOS por múltiplas *threads*. Existiam duas maneiras de se abordar o problema: toda a biblioteca de funções compartilharia a mesma variável de exclusão mútua, o que causaria uma serialização do acesso aos recursos do HetNOS; ou a adoção de variáveis de exclusão mútua locais em cada função disponível ao usuário, o que, teoricamente, deveria permitir um nível de concorrência maior dentro do sistema.

As execuções-teste foram realizadas em estações de trabalho Sun modelo Spark Ultra 10, com o sistema operacional SunOS *release* 5.7 Generic e utilizando somente chamadas compatíveis com o padrão POSIX do pacote de programação *multithreaded* Pthreads. A aplicação consistia de um simples algoritmo de envio de mensagens e requisição de informações do estado atual do sistema distribuído, variando o número de *threads* criadas e, também, o número de mensagens enviadas. Os resultados podem ser comparados nos gráficos da figura 2.

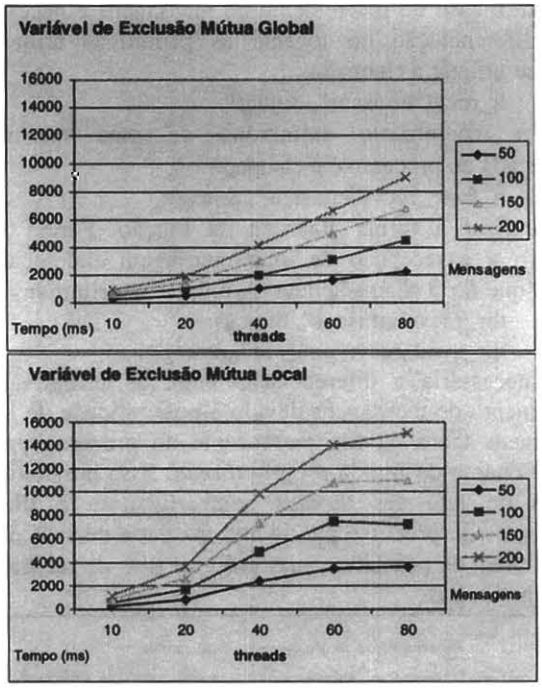

Fig. 2 Gráficos de desempenho de aplicação-teste das HetNOS Threads

O desempenho alcançado pela implementação contendo uma única variável de exclusão global a toda biblioteca de funções foi superior à alternativa com variáveis locais, sendo, por isso, a solução adotada no projeto do protótipo. Esse resultado, deve-se às características do núcleo HetNOS, que não é *multithreaded*, o que anula o ganho de desempenho que poderia ser obtido pelas instâncias locais da variáveis que, ao mesmo tempo, trazem uma grande sobrecarga ao sistema quando precisam alocar áreas de memória a cada chamada feita pelo usuário para essas variáveis, sem mencionar a inicialização necessária aos tipos *mutex* no próprio pacote Pthreads.

A implementação do protótipo conta com as primitivas de cadastramento de *threads*, implementadas em toda sua funcionalidade, isto é, permitindo que *threads* sejam inseridas em estruturas de dados localizadas no espaço de nomes global do sistema HetNOS. Através desta operação as *threads* passam a ser conhecidas por um nome único em toda a rede de estações integrante do sistema HetNOS e são passíveis de ser referenciadas, mesmo remotamente, por qualquer outra entidade do sistema. Estão em sua parte final de implantação as primitivas de comunicação assíncrona, garantindo a operacionalidade das HetNOS Threads, o que permite seu uso por desenvolvedores de aplicações distribuídas, como vem sendo feito por integrantes do projeto através de uma série de aplicações-exemplo.

IV. COMPARAÇÕES DO MODELO HETNOS THREADS

Existem outros sistemas que buscam os mesmos objetivos das HetNOS Threads: prover capacidade de comunicação para pacotes de programação *multithreaded*. Nesta seção as diferentes soluções para o problema serão comparadas com o modelo proposto.

Os pacotes que serão comentados de alguma forma implementam comunicação entre processos partindo do contexto de uma *thread*, como o Nexus [FOS 96], Chant [HAI 96] e Athapascan [BRI 98] [CAR 98].

Nexus é uma interface *runtime* projetada para dar suporte à interoperabilidade entre linguagens de programação em sistemas distribuídos sem memória compartilhada. A comunicação é feita através abstrações de *threads* que são capazes de comunicar-se através de chamadas remotas de procedimentos assíncronas. A desvantagem desta implementação encontra-se na sobrecarga causada para selecionar, verificar e chamar as rotinas de processamento de mensagens corretas e o tratamento aos chamados "ponteiros globais" sem que haja um suporte específico de *hardware* ou do sistema operacional para tanto.

A proposta do Chant difere da apresentada pelo Nexus por utilizar comunicação ponto-a-ponto e bibliotecas de programação *multithreaded* e de comunicação padrão, como POSIX-threads e MPI. São propostas duas extensões, *chanters* e *ropers*, que são utilizadas para a identificação das *threads* no sistema (*chanters*) ou de um conjunto de *threads* (*ropers*).

As mensagens são endereçadas a um *chanter* que, através de *threads* mantidas pelo pacote Chant, localiza a *thread* remota e faz com que esta receba a mensagem. O problema verificado nos algoritmos do Chant é a complexidade requerida para entregar as mensagens enviadas, pois como o sistema depende de outros pacotes de comunicação, i.e. MPI, p4, é necessária uma manipulação dos cabeçalhos das mensagens para adaptarem-se ao endereçamento Chant. Outra questão crítica é o modo de sinalizar às *threads* que existem mensagens para elas: o *polling*. Este processo é executado pelas próprias *threads*, que requisitam informações do sistema, e perdem tempo de processamento devido a este procedimento.

Por fim, Athapascan é um ambiente de programação paralela cujo objetivo é a execução transparente de aplicações paralelas portáveis em sistemas heterogêneos com bom desempenho. A ferramenta de comunicação adotada pelo Athapascan é a MPI, implementando assim funções de comunicação ponto-a-ponto e comunicação em grupo, utilizando-se de uma abstração chamada *communicator*. A parte relativa à programação *multithreaded* é provida por bibliotecas de *threads* POSIX-threads.

A troca de mensagens é realizada através de "portas", endereços associadas às *threads* que são capazes de enviar e receber mensagens. A camada inferior do sistema Athapascan, denominada Akernel, é responsável por oferecer ao programador uma interface homogênea tanto aos serviços de comunicação quanto às chamadas de programação *multithreaded*.

A questão do *polling* das mensagens também é um problema apresentado por este sistema, assim como algumas restrições associadas ao modelo de programação escolhido, o qual seria o de utilizar aplicações regulares, ou seja, o mesmo executável em todos os nodos, e a computação necessária para compensar a falta de segurança da biblioteca MPI na execução com *threads*. Essas características reduzem as possibilidades de programação do usuário e reduzem o desempenho do sistema.

A análise efetuada ao longo da seção teve por objetivo salientar os pontos fracos dos sistemas similares ao HetNOS Threads, que durante sua fase de projeto teve por objetivo implementar soluções para essas mesmas dificuldades.

O *polling* não é necessário, já que a camada de comunicação é *thread-aware*, e armazena as mensagens para as *threads* até que elas as requisitem, não havendo a necessidade de interromper seu processamento normal para isso. Não é preciso executar aplicações regulares nos diversos nodos do sistema, e a manipulação das mensagens trocadas não causa grande sobrecarga ao sistema pois a camada de comunicação é integrada aos processos responsáveis pela distribuição das aplicações e resolução dos endereços remotos, resultando em maior eficiência do conjunto do sistema.

Outra vantagem das HetNOS Threads, esta no tocante às facilidades de programação, está na utilização do paradigma de trocas de mensagens ponto-a-ponto para a comunicação entre as *threads* e processos do sistema, forma mais intuitiva e simplificada do que estruturas complexas como os "ponteiros globais" do Nexus e mais flexíveis que as "portas" do Athapascan.

V. CONCLUSÃO

A grande parte dos sistemas operacionais atuais provê pacotes de programação *multithreaded* em seus projetos e, além disso, existem muitos pacotes de *threads*, desenvolvidos por empresas ou pesquisadores, disponíveis tanto em ambiente acadêmico quanto comercial. Os referidos pacotes cobrem praticamente todas as funcionalidades que se podem exigir de um produto de sua classe, alguns com maior e outros com menor eficiência, permitindo o uso do paradigma de programação *multithreaded* em projetos de qualquer escala.

A falta de suporte à programação distribuída é a maior deficiência apresentada por esta classe de recursos de programação. Por mais que existam facilidades para a

comunicação, normalmente via memória compartilhada, entre *threads* de um mesmo nodo, faltam mecanismos de comunicação que sejam transparentes à localidade das *threads*. Sem mencionar os mecanismos de sincronização remota que, para a cooperação entre procedimentos remotos, possuem tanta importância.

A proposta do pacote das HetNOS Threads insere-se nesta lacuna, com o objetivo de apresentar uma alternativa, ao unir a um pacote de programação *multithreaded*, as Pthreads, a capacidade de execução distribuída de aplicações de um sistema operacional de rede, o HetNOS. O presente artigo expõe o modelo das HetNOS Threads como uma alternativa mais eficiente do que os demais pacotes de *threads*. Esta vantagem fica clara ao analisarmos o ganho em tempo de programação, manutenção e porte quando o aspecto de localidade é abstraído, sem mencionar os ganhos de funcionalidade atingidos pelas novas funções distribuídas.

Algumas comparações com outros modelos propostos para a resolução do mesmo tipo de problema foram apresentadas para demonstrar os acertos de arquitetura e a importância do protótipo das HetNOS Threads como ferramenta de desenvolvimento de aplicações distribuídas que também tenham por objetivo utilizar as possibilidades dos novos sistemas computacionais através da programação *multithreaded*. Sobretudo, as HetNOS Threads visam oferecer um meio de integração de tecnologias para a construção de aplicações distribuídas de melhor desempenho, sobre plataformas eficientes, utilizando primitivas de programação intuitivas que facilitem seu desenvolvimento.

AGRADECIMENTOS

Agradecemos ao grupo de ProcPar e ao Instituto de Informática da UFRGS pela ajuda e material necessário à nossa pesquisa além do CNPq pelo apoio financeiro.

REFERÊNCIAS

[BAR 94] BARCELLOS, A. M. P. *HetNOS - Um Sistema Operacional de Rede como Ferramenta de Apoio ao Desenvolvimento de Sistemas Distribuídos*. Porto Alegre, Relatório de Pesquisa, UFRGS, 1994.

[BRI 98] BRIAT, J., CARISSIMI,A. S. *Intégration de Threads et Communication: Etude de Cas*. Grenoble, LMC – IMAG, 1998.

[BUT 94] BUTLER, R., LUSK, E. *Monitors, Messages, and Clusters: The P4 Parallel Programming System*. Parallel Computing, vol. 20, pp. 547-64, 1994.

[CAR 98] CARISSIMI, A., PASIN, M. *Athapascan: An Experience on Mixing MPI Communications and Threads*. Grenoble, APACHE Project, 1998.

[GEI 94] GEIST, A., BEGUELIN, A., DONGARRA, J., JIANG, W., MANCHECK, R., SUNDERAM, V. *PVM: Parallel Virtual Machine*. MIT press, 1994.

[BAL 98] BAL, H. E., BHOEDJANG, R., HOFMAN, R., JACOBS, C., LANGENDOEN, K., RÜHL, T., KAASHOEK, M. F. *Performance Evaluation of the Orca Shared-Object System*. New York, ACM Transactions on Computer Systems, vol. 16, nr. 1, pp. 1-40, 1998.

[FOS 96] FOSTER, I., KESSELMAN, C., TUECKE, S. *The Nexus Approach to Integrating Multithreading and Communication*. Journal of Parallel and Distributed Computing, vol. 37, nr. 1, pp. 70-82, 1996.

[HAI 96] HAINES, M., CRONK, D., MEHROTA, P. *Chant: Lightweight Threads in a Distributed Memory Environment*. Journal of Parallel and Distributed Programming, 1996.

[LEV 91] LEVINE, D. *POSIX Programmer's Guide - Writing Portable UNIX Programs*. Sebastopol, O'Reilly & Associates, 1991.

[LEW 98] LEWIS, B., BERG, D. J. *Multithreaded Programming with Pthreads*. Mountain View, Prentice Hall, 1998.

[MAL 99] MALACARNE, J., GEYER, C. F. R. *Ambientes de Programação Visual Paralela e Distribuída*. Porto Alegre, CPGCC, UFRGS, 1999.

[MES 94] Message Passing Interface Forum. *Document for a Standard Message Passing Interface*. International Journal of Supercomputer Applications and High Performance Computing Tech., vol. 8, 1994.

[ONV 95] ONVURAL, R. *Asynchronous Transfer Mode Networks. Performance Issues*. Norwood, Artech House, 1995.

[PAK 96] PAKIN, S., LAURIA, M., CHIEN, A. *High Performance messaging on workstations: Illinois Fast Message for Myrinet*. In Proceedings of SuperComputing '95, IEEE Computer Society Press, 1996.

[POS 90] POSIX P1003.4a. *Threads Extension for Portable Operating Systems*. IEEE, 1990.

[PRO 99] PROVENZANO, C. *Pthreads: an implementation of POSIX 1003.1.c*. Disponível em http://www.mit.edu:8001/people/proven/pthreads

[RIV 97] RIVIERE, M. *Introduction to Concurrent Programming Thread and Communication Libraries*. IMAG – LMC, 1997.

[SUN 95] SUNSOFT. *POSIX.1c / D10 Summary*. Mountain View, Sun Microsystems, 1995.

[SUZ 92] SUZUKI, N. *Shared Memory Multiprocessing*. MIT Press, 1992.

[TAN 92] TANENBAUM, A. S. *Modern Operating Systems*. Englewood Cliffs, Prentice Hall, 1992.