ALGORÍTMOS PARALELOS
(Aula 7)

Neyval C. Reis Jr.

OUTUBRO/2004

Laboratório de Computação
de Alto Desempenho

DI/UFES

Tópico	20 janeiro	27 janeiro	3 fev	10 fev	17 fev	24 fev	3 março
Paradigma de Paralelismo de Dados	■			Carnaval			
Memória Compartilhada e Distribuída		■					
Memória Compartilhada (OPEN MP)		■					
Trabalho - Análise de complexidade de algoritmos			■				
Modelos de Programação Paralela			■				
Trabalhos - Super LU					■		
Trabalhos - Otimização					■		
Trabalhos – Medição de desempenho						■	
Trabalhos - POM						■	
Trabalhos - GMRES							■
Trabalhos – Algoritmos Assíncronos							■

Programa do Curso

1. Introdução
2. Arquitetura de Computadores
3. Arquiteturas de Sistemas Paralelos
4. Computação de Alto Desempenho
5. Programação Paralela (modelos e paradigmas)
6. Análise de Desempenho e Instrumentação
7. Aplicações

Tipos de Ambientes e Paradigmas de Programação

Troca de Mensagens (Message Passing): é o método de comunicação baseada no envio e recebimento de mensagens através da rede seguindo as regras do protocolo de comunicação entre vários processadores que possuam memória própria. O programador é responsável pela sincronização das tarefas.

Paralelismo de Dados (Data Parallel): é a técnica de paralelismo de dados, normalmente automática ou semi-automática, ou seja, é o método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação. (Aulas anteriores)

Memória Distribuída e Compartilhada (Distributed-Shared Memory): Emula uma máquina de memória compartilhada em uma máquina de memória distribuída. O método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação.

Tipos de Ambientes e Paradigmas de Programação

Troca de Mensagens (Message Passing): é o método de comunicação baseada no envio e recebimento de mensagens através da rede seguindo as regras do protocolo de comunicação entre vários processadores que possuam memória própria. O programador é responsável pela sincronização das tarefas.

Paralelismo de Dados (Data Parallel): é a técnica de paralelismo de dados, normalmente automática ou semi-automática, ou seja, é o método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação.

Memória Distribuída e Compartilhada (Distributed-Shared Memory): Emula uma máquina de memória compartilhada em uma máquina de memória distribuída. O método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação.

Memória Distribuída e Compartilhada

- Motivação
- Paradigma
- Implementações
  - Lazy Release consistency
  - Multiple-Writer Protocols
  - Home-based vs. Homeless
  - Comparação de Performance
- Programando
  - Exemplo Treadmarks
- Tendências

Troca de Mensagens

Extremamente flexível, porém ...

- Codificação e debugging são tarefas difíceis
- Programador é completamente responsável pela produção de um código eficiente e em manter os processadores ocupados, balanceados e em constante comunicação.

Paradigma do Memória Distribuída e Compartilhada

Motivação:

- **Tecnologias de rede cada vez mais rápidas (maior bandwitdh e menor latência).**
- **Processadores cada vez mais rápidos e baratos.**
- **Facilidade de programação.**

Sequential =
single thread + single address space

Shared memory =
multiple threads + single address space

Message passing =
multiple threads + multiple address spaces

Paradigma do Memória Distribuída e Compartilhada

Motivação:

- **Permite ao programador um maior foco na aplicação.**
- **Paralelismo focado na distribuição de tarefas, a memória do sistema é compartilhada. Detalhes de comunicação não são percebidos pelo programador.**

- **Programação para memória compartilhada:**

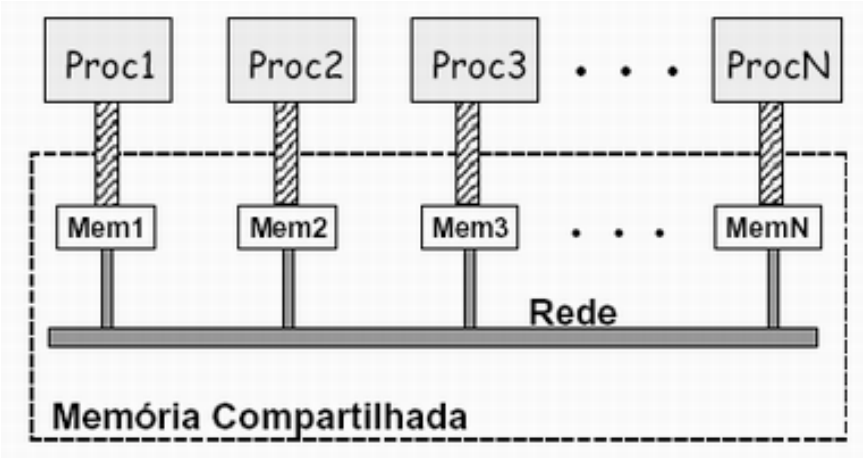
Threads

Synchronization

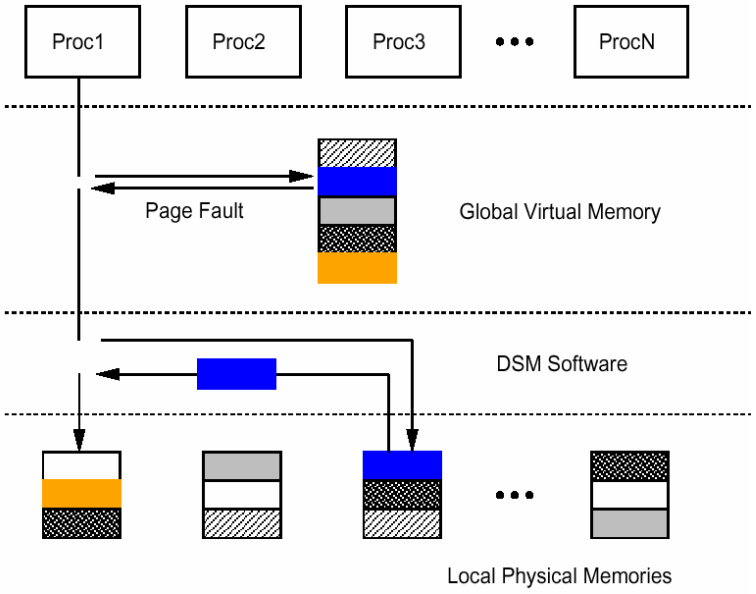
- Locks
- Barriers
- Flags

Shared memory allocation

Paradigma do Memória Distribuída e Compartilhada

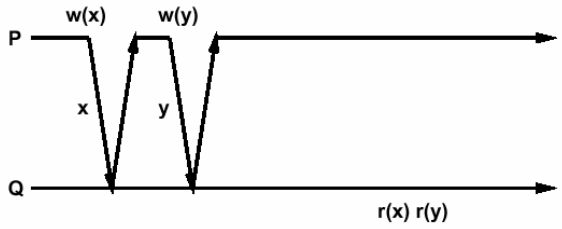


Implementação Convencional do Paradgma de Memória Compartilhada Distribuída (DSM)



Performance Problem: Sequential Consistency

Every write visible "immediately"



Problems:

- Number of messages
- Latency

Performance Problems: Solutions

Goal:

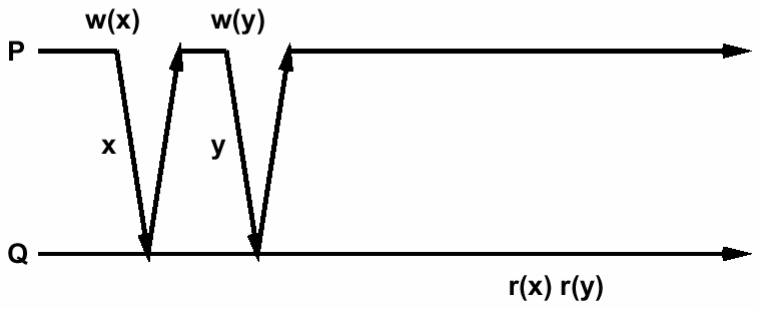
- Reduce communication
- Keep shared memory model

Techniques:

- Lazy release consistency [Keleher 92]
- Multiple writer protocol [Carter 91]

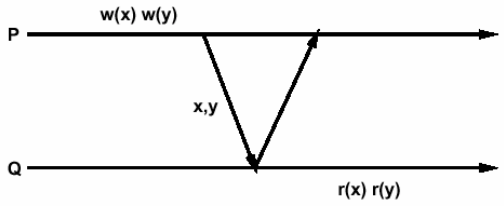
Sequential Consistency

Every write visible "immediately"



Relaxed Consistency Models

Delay making writes visible



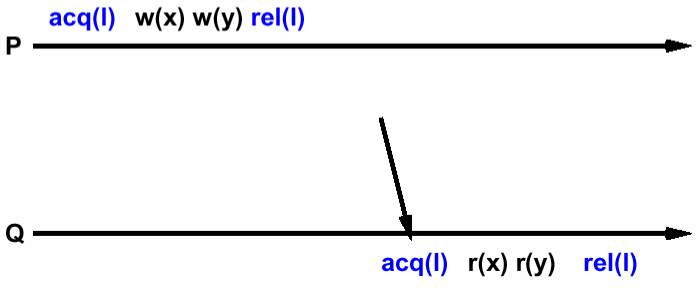
Goal:

- Reduce number of messages
- Hide latency

Delay until when?

Release Consistency (RC)

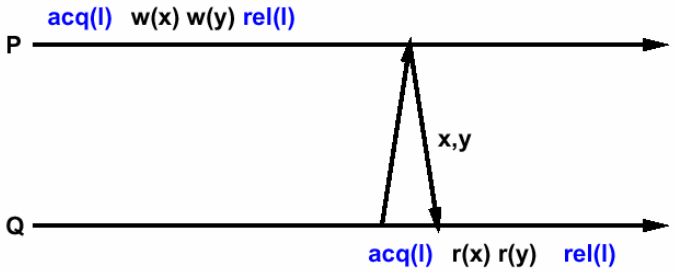
Delay until Q synchronizes with P



Lazy RC

Pull modifications at acquire

(rather than push them at release)



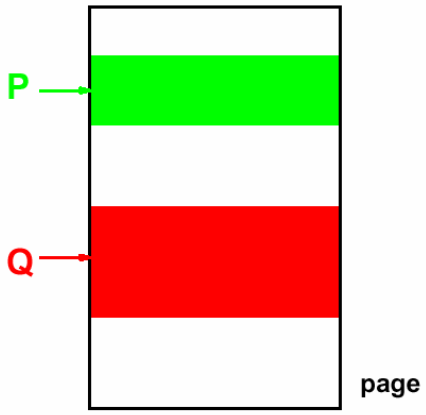
Fewer messages

Lazy Release Consistency

Lazy release consistency (LRC) [6] is an algorithm that implements the release consistency (RC) [4] memory model. The LRC algorithm [6] delays the propagation of modifications to a processor until that processor performs an *acquire*. An acquire marks the beginning of a critical section. Specifically, LRC insures that the memory seen by the processor after an acquire is consistent with the happened-before-1 partial order [1].

False Sharing

Pieces of the same page updated by different processors



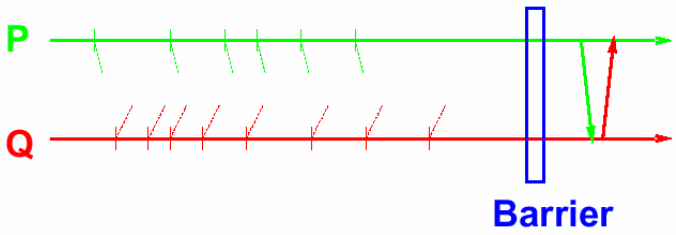
Multiple Writer Protocol

Addresses false sharing

Buffer writes until synchronization

Create diffs

Synchronize → pull in modifications

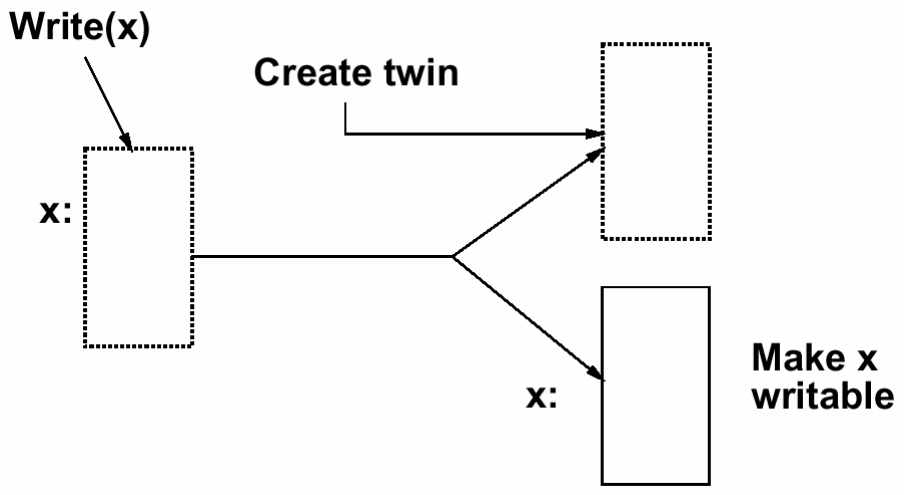


Multiple-Writer Protocols

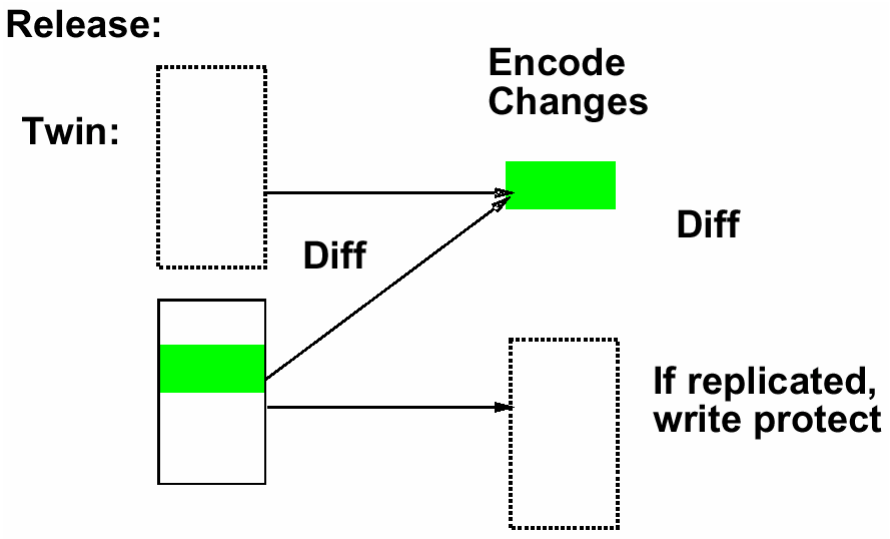
With a multiple-writer protocol, two or more processors can simultaneously modify their copy of a (shared) page. The two most popular multiple-writer protocols that are compatible with LRC are the TreadMarks protocol *(Tmk)* [6] and the Princeton home-based protocol *(HLRC)* [9].

In both protocols modifications to (shared) pages are detected by virtual memory faults *(twinning)* and captured by comparing the page to its twin *(diffing)* [5]. The protocols differ, however, in the location where modifications are kept and in the method by which they get propagated.

Diff Creation e Twinnig



Diff Creation e Twinnig



TreadMarks

In Tmk, a diff is not created for a modified (shared) page until a processor requests that diff in order to update its copy of the page. This *lazy* creation of diffs results in greater aggregation of changes, less diffing overhead and a reduced amount of consistency information.

When a processor faults, accessing an invalid page, it examines the write notices for that page. The write notices specify which processors have modified the page. Each write notice contains a vector timestamp that specifies the time of the modification. If one write notice's vector timestamp dominates the others, then the processor that created that write notice has all of the diffs required by the faulting processor to update its copy of the page. Thus, the faulting processor can obtain those diffs with a single request message. If, however, a page doesn't have a dominant write notice, that is, two or more write notices have concurrent vector timestamps, then the faulting processor has to send a request message to each of the corresponding processors. These request messages are overlapped to reduce the time that the faulting processor will wait.

Princeton Home-based Protocol (HLRC)

In HLRC, every shared page is statically assigned a home processor by the program. At a *release*, which marks the end of a critical section, a processor immediately generates the diffs for the pages that it has modified since its last release. It then sends these diffs to their home processor(s), where they are immediately applied to the home's copy of the page. The home's copy of a page is never invalid, but it may be write protected.

When a processor faults, accessing an invalid page, it sends a request to the page's home processor. In current implementations, the home processor always responds with a complete copy of the page, instead of one or more diffs. Thus, a diff can be discarded by the creating and home processors as soon as it is applied to the home processor's copy of the page.

Comparação de Desempenho

A Performance Comparison of Homeless and Home-based Lazy Release Consistency Protocols in Software Shared Memory, A.L. Cox, E. de Lara, Y.C. Hu, and W. Zwaenepoel, Proceedings of the Fifth High Performance Computer Architecture Conference, pp. 279-283, January 1999.

Application	Tmk			HLRC		
	8	16	32	8	16	32
Barnes-Hut	4.84	5.83	4.84	4.85	6.51	7.59
Water	5.63	9.18	11.4	5.36	8.09	9.45
IS	7.1	12.7	17.9	6.99	12.3	16.6
RB SOR	7.62	14.8	25.5	7.65	14.5	25.4
Gauss	6.43	8.98	8.32	6.35	8.80	7.98
3D FFT	4.37	8.29	15.1	4.40	8.28	15
TSP	7.41	13.2	21.2	7.36	13.3	21.1

Table 2: Speedups on 8, 16, and 32 processors for Tmk and HLRC.

Comparação de Desempenho

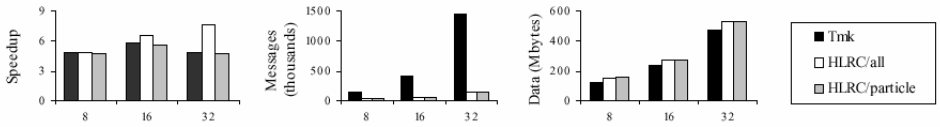


Figure 1: Speedup, messages, and data comparison among HLRC/all, HLRC/particle, and Tmk for Barnes-Hut.

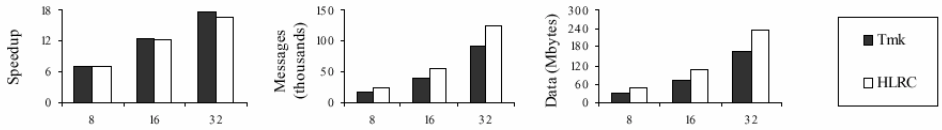


Figure 2: Speedup, messages, and data comparison among HLRC and Tmk for IS.

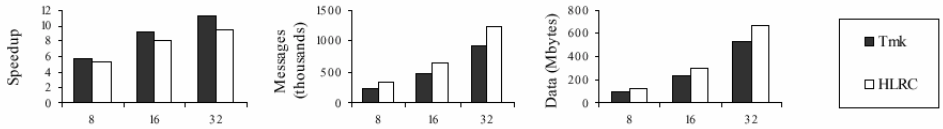
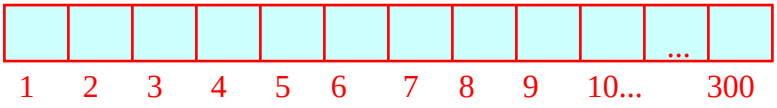


Figure 3: Speedup, messages, and data comparison among HLRC and Tmk for Water.

Exemplo

Soma dos elementos de um vetor



Poderíamos pensar em uma máquina paralela (com vários processadores) onde cada processador calcularia a soma dos elementos de uma parte do vetor

Solução do problema serial

```
      PROGRAM EXEMPLO
C-----------------------------------------------------
      IMPLICIT NONE
      INTEGER ID
      PARAMETER(ID=300)
      INTEGER VET(ID), I, SOMA
C-------------Inicio do programa principal --------------
      SOMA=0
      OPEN (UNIT=10,FILE='b.dat',STATUS='OLD')
      DO I=1, ID
        READ (10,*) VET(I)
      ENDDO
      DO I=1,ID
        SOMA=SOMA+VET(I)
      ENDDO
      WRITE(*,*) 'SOMA DOS ELEMENTOS DO VETOR E:', SOMA
      END
```

Treadmarks – Implementação Paralela

```
      PROGRAM SOMA_VETOR
C-----------------------------------------------------------
      IMPLICIT NONE
      INCLUDE 'Tmk_fortran.h'
      INTEGER ID,NP
      PARAMETER(NP=4)
      PARAMETER (ID=300)
      INTEGER INICIO, FIM, SOMAL, SOMAG, I, VET(ID),SOMAP(NP)
      COMMON /Tmk_shared_common/ VET,SOMAG,SOMAP
C-----------------Inicio do programa principal ---------------
      CALL Tmk_startup()
      INICIO = (ID*Tmk_proc_id)/Tmk_nprocs + 1
      FIM = (ID*(1 + Tmk_proc_id))/Tmk_nprocs
      IF (Tmk_proc_id.EQ.0) THEN
        OPEN (1,FILE='b.dat',STATUS='OLD')
        DO I=1,ID
          READ(1,*) VET(I)
        ENDDO
        SOMAG=0
      ENDIF
      CALL Tmk_barrier(0)
```

Treadmarks – Implementação Paralela

```
      PROGRAM SOMA_VETOR
C-----------------------------------------------------------
      IMPLICIT NONE
      COMMON /Tmk_shared_common/ VET,SOMAG,SOMAP
      PARAMETER(NP=4)
      PARAMETER (ID=300)
      INTEGER INICIO, FIM, SOMAL, SOMAG, I, VET(ID),SOMAP(NP)
      COMMON /Tmk_shared_common/ VET,SOMAG,SOMAP
C-----------------Inicio do programa principal ---------------
      CALL Tmk_startup()
      INICIO = (ID*Tmk_proc_id)/Tmk_nprocs + 1
      FIM = (ID*(1 + Tmk_proc_id))/Tmk_nprocs
      IF (Tmk_proc_id.EQ.0) THEN
        OPEN (1,FILE='b.dat',STATUS='OLD')
        DO I=1,ID
          READ(1,*) VET(I)
        ENDDO
        SOMAG=0
      ENDIF
      CALL Tmk_barrier(0)
```

Treadmarks – Implementação Paralela

```
      PROGRAM SOMA_VETOR
C-----------------------------------------------------------
      IMPLICIT NONE
      INCLUDE 'Tmk_fortran.h'
      INTEGER ID,NP
      P
      P
      I                                                    )
      COMMON /Tmk_shared_common/ VET,SOMAG,SOMAP
C-----------------Inicio do programa principal ---------------
      CALL Tmk_startup()
      INICIO = (ID*Tmk_proc_id)/Tmk_nprocs + 1
      FIM = (ID*(1 + Tmk_proc_id))/Tmk_nprocs
      IF (Tmk_proc_id.EQ.0) THEN
        OPEN (1,FILE='b.dat',STATUS='OLD')
        DO I=1,ID
          READ(1,*) VET(I)
        ENDDO
        SOMAG=0
      ENDIF
      CALL Tmk_barrier(0)
```

INICIO = (ID*Tmk_proc_id)/Tmk_nprocs + 1
FIM = (ID*(1 + Tmk_proc_id))/Tmk_nprocs

Treadmarks – Implementação Paralela

```
      PROGRAM SOMA_VETOR
C-----------------------------------------------------------
      IMPLICIT NONE
      INCLUDE 'Tmk_fortran.h'
      INTEGER ID,NP
      PARAMETER(NP=4)
      PARAMETER (ID=300)
      INTEGER INICIO, FIM, SOMAL, SOMAG, I, VET(ID),SOMAP(NP)
      COMMON /Tmk_shared_common/ VET,SOMAG,SOMAP
C-----------------Inicio do programa principal ---------------
      CALL Tmk_startup()
      INICIO = (ID*Tmk_proc_id)/Tmk_nprocs + 1
      FIM = (ID*(1 + Tmk_proc_id))/Tmk_nprocs
      IF (Tmk_proc_id.EQ.0) THEN
        OPEN (1,FILE='b.dat',STATUS='OLD')
        DO I=1,ID
          READ(1,*) VET(I)
        ENDDO
        SOMAG=0
      ENDIF
      CALL Tmk_barrier(0)
```

CALL Tmk_barrier(0)

Treadmarks – Implementação Paralela

```
      SOMAL=0
      DO I=INICIO,FIM
        SOMAL=SOMAL+VET(I)
      ENDDO
      SOMAP(Tmk_proc_id+1)=SOMAL
      CALL Tmk_barrier(0)
      IF (Tmk_proc_id.EQ.0) THEN
        DO I=1,Tmk_nprocs
          SOMAG=SOMAG+SOMAP(I)
        ENDDO
      ENDIF
      CALL Tmk_barrier(0)
      WRITE(*,*) 'SOMA GLOBAL:',SOMAG
      CALL Tmk_exit(0)
      END
```

```
DO I=INICIO,FIM
  SOMAL=SOMAL+VET(I)
ENDDO
```

Treadmarks – Implementação Paralela

SOMAP(Tmk_proc_id+1)=SOMAL

```
SOMAL=0
DO I=INICIO,FIM
  SOMAL=SOMAL+VET(I)
ENDDO
SOMAP(Tmk_proc_id+1)=SOMAL
CALL Tmk_barrier(0)
IF (Tmk_proc_id.EQ.0) THEN
  DO I=1,Tmk_nprocs
    SOMAG=SOMAG+SOMAP(I)
  ENDDO
ENDIF
CALL Tmk_barrier(0)
WRITE(*,*) 'SOMA GLOBAL:',SOMAG
CALL Tmk_exit(0)
END
```

Treadmarks – Implementação Paralela

```
IF (Tmk_proc_id.EQ.0) THEN
  DO I=1,Tmk_nprocs
    SOMAG=SOMAG+SOMAP(I)
  ENDDO
ENDIF
```

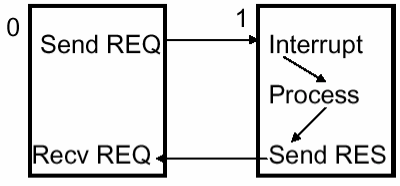
```
SOMAL=0
DO I=INICIO,FIM
  SOMAL=SOMAL+VET(I
ENDDO
SOMAP(Tmk_proc_id+1)
CALL Tmk_barrier(0)
IF (Tmk_proc_id.EQ.0) THEN
  DO I=1,Tmk_nprocs
    SOMAG=SOMAG+SOMAP(I)
  ENDDO
ENDIF
CALL Tmk_barrier(0)
WRITE(*,*) 'SOMA GLOBAL:',SOMAG
CALL Tmk_exit(0)
END
```

Tendências Futuras

- Software DSM
  - HLRC/VIA (Rutgers), TreadMarks (Rice), JIAJIA (ICT China)
- Depends on user and software layer
- Depends on communication protocols provided by the system such as TCP, UDP, etc.
- Degraded performance because of false sharing and high overhead of communication
- Has scaling problems

Tendências Futuras

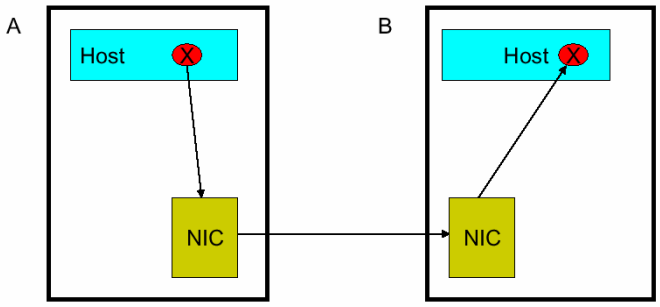
- Traditional DSM
  - Uses Request / Response Communication Model (asynchronous)
  - Separate signal handler thread needed
  - Application Processing interrupted
  - Cache Effects



- Can network based features be used to reduce interrupt overhead ?

Tendências Futuras

RDMA Write Example

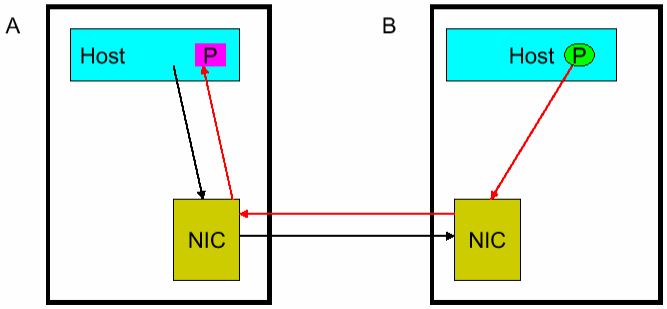


Designing High Performance DSM Systems using InfiniBand Features

Ranjit Noronha
and
Dhabaleswar K. Panda
The Ohio State University

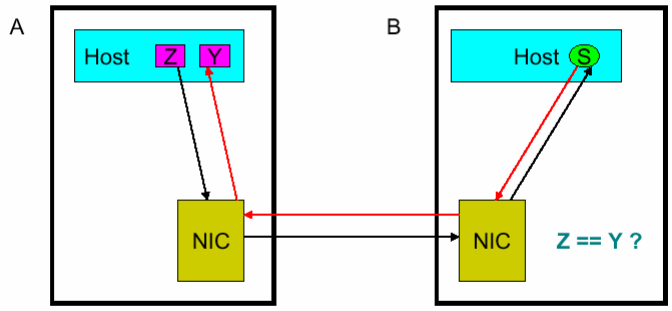
Tendências Futuras

RDMA Read Example



Tendências Futuras

Remote Atomic Operations Example



• Compare and Swap

Tendências Futuras

- Modern Interconnects (InfiniBand, Myrinet, Quadrics)
- Low Latency (InfiniBand 5.0 µs)
- High Bandwidth (InfiniBand 4X upto 10 Gbps)
- Programmable NIC
- User Level Protocols (VAPI, GM)
- Can deliver performance close to that of the underlying hardware
- RDMA Write/Read, Atomic Operations, Service Levels, Multicast

INFINIBAND OVERVIEW

The InfiniBand Architecture (IBA) [12] defines a System Area Network (SAN) for interconnecting processing nodes and I/O nodes. It provides the communication and management infrastructure for inter-processor communication and I/O. In an InfiniBand network, processing nodes and I/O nodes are connected to the fabric by Channel Adapters (CA). Channel Adapters usually have programmable DMA engines with protection features. There are two kinds of channel adapters: Host Channel Adapter (HCA) and Target Channel Adapter (TCA). HCAs sit on processing nodes. TCAs connect I/O nodes to the fabric.

InfiniBand Architecture supports both channel and memory semantics.

In memory semantics, RDMA write and RDMA read operations are used instead of send and receive operations. These operations are one-sided and do not incur software overhead at the other side. The sender initiates RDMA operations by posting RDMA descriptors. A descriptor contains both the local data source addresses (multiple data segments can be specified at the source) and the remote data destination address. At the sender side, the completion of an RDMA operation can be reported through CQs. The operation is transparent to the software layer at the receiver side. Although InfiniBand Architecture supports both RDMA read and RDMA write, in the following discussions we focus on RDMA write because in the current hardware RDMA write usually has better performance.

Tendências Futuras

High Performance RDMA-Based MPI Implementation over InfiniBand*

Jiuxing Liu Jiesheng Wu Dhabaleswar K. Panda

Computer and Information Science
The Ohio State University
Columbus, OH 43210

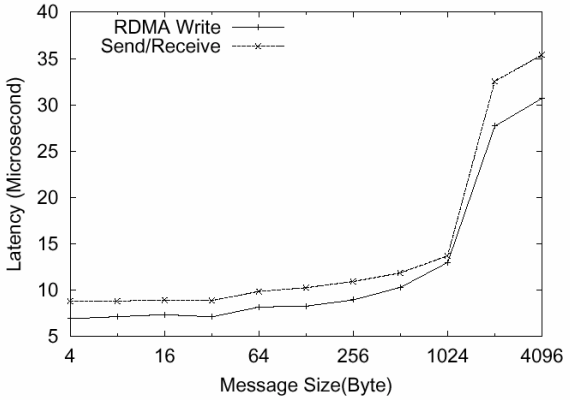


Figure 12: MPI Latency Comparison

Tendências Futuras

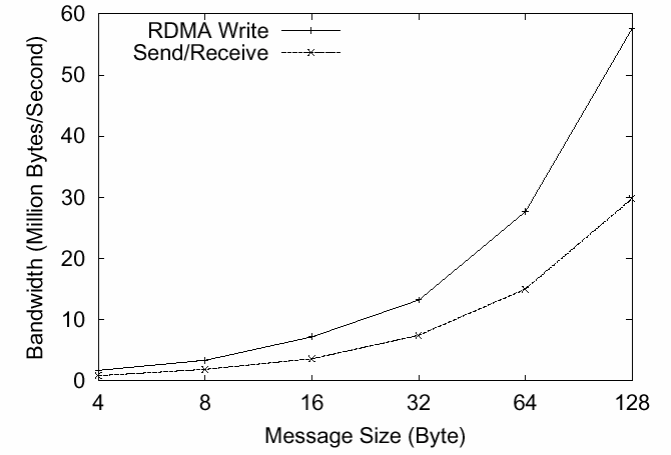


Figure 13: MPI Bandwidth Comparison (Small Messages)

Tendências Futuras

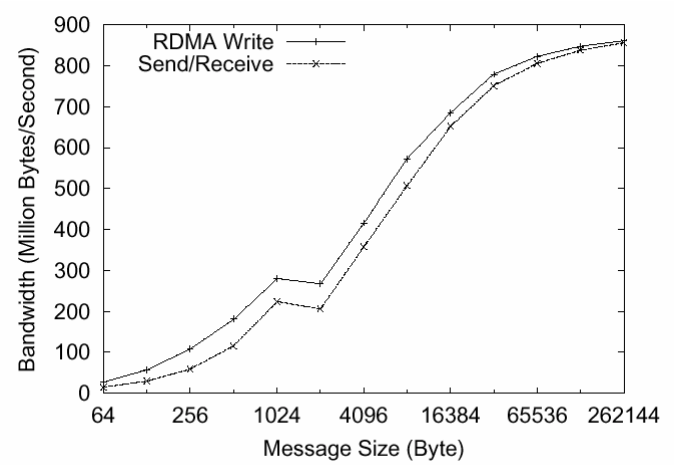


Figure 14: MPI Bandwidth Comparison

Tendências Futuras

Muitos pesquisadores apontam que o modelo DSM será o mais utilizado no futuro, devido aos bons níveis de performance obtidos, à relativa facilidade de programação e aos grandes avanços das tecnologias de rede.