Linguagens para Programação Paralela

- Fatores que devem ser considerados para a escolha de uma linguagem de programação paralela:
  - Tipo de aplicação: Similar a programação seqüencial;
  - Arquitetura a ser utilizada: Granulação e Comunicação;
  - Usuários: Profissionais da área de computação X outros usuários;
  - Tempo de aprendizagem e desenvolvimento;
  - Desempenho.

Linguagens para Programação Paralela

- Tipos de linguagens disponíveis:

Declarativas

- Lógicas e Funcionais
- O que fazer
- Granulação fina
- Compilador explora paralelismo
- Exemplos: Ling. de fluxo de dados -
  - Sisal, Val, Haskell

Imperativas

- Como Fazer
- Paralelismo explorado pelo programador
- Mais utilizadas
- Diversas opções disponíveis

Linguagens para Programação Paralela

- Abordagens para Linguagens Declarativas
  - Linguagens Lógicas - Prolog
    - Paralelismo entre clausulas - AND / OR
  - Linguagens Funcionais
    - Funções são avaliadas em paralelo
    - h(f(3, 4), g(j))
      - 1o. Passo - avalia f(3,4) e g(j)
      - 2o. Passo - avalia h

Linguagens para Programação Paralela

- Abordagens para Linguagens Declarativas
  - Linguagens Fluxo de Dados
    - Grafo de Fluxo de Dados - difícil de implementar
    - Deve permitir determinar as dependências de dados do programa
    - Ordem do programa - limitada apenas pela dependência de dados
    - Val - Paralelismo implícito - nenhum comando de execução paralela
    - Haskell - Proposta de unificação das linguagens funcionais

Linguagens para Programação Paralela

- Abordagens para Linguagens Imperativas
  - Compiladores que oferecem paralelização automática;
  - Extensões das linguagens seqüenciais, através de compiladores especiais;
  - Linguagens específicas para programação paralela;
  - Biblioteca de troca de mensagens para linguagem de uso geral.

Linguagens para Programação Paralela

- Paralelização automática
  - Requer pouco ou nenhum conhecimento do usuário;
  - Normalmente explora granulação fina;
  - Baixa flexibilidade;
  - *Speedup* depende da aplicação;
  - Exemplo: Computadores vetoriais.

Linguagens para Programação Paralela

- Extensões das linguagens seqüenciais através de compiladores especiais
  - Comandos adicionais para implementação de programas paralelos;
  - Opção adequada para paralelização de programas existentes;
- Vantagem:
  - Não é necessário o aprendizado de uma linguagem nova;
  - Disponibilidade de compiladores.
- Desvantagem:
  - Dificuldade em integrar as construções paralelas de forma clara;
  - Extensões podem interferir na otimização do compilador e diminuir portabilidade e desempenho.

Linguagens para Programação Paralela

- Extensões das linguagens seqüenciais através de compiladores especiais
  - Exemplos: Fortran 90, Fortran D, HPF (*High Performance Fortran*), C* (C paralelo para a *Connection Machine*), *Sequent C*, etc.
  - Paralelismo de Dados:
    - Loops
    - Operações Vetoriais
  - Exemplo
    - Cálculo do valor de π
    - π = Integral de 0 até 1 de $4/(1+x^2)$
    - Área sob a curva de $4/(1+x^2)$
    - Cálculo através de determinação da área de n retângulos

Linguagens para Programação Paralela

- Fortran 90
  - Para máquinas vetoriais
  - Memória Compartilhada

```
INTEGER, PARAMETER :: N = 131072
INTEGER, PARAMETER :: LONG = SELECT_REAL_KIND (13,99)
REAL (KIND=LONG) PI, WIDTH
INTEGER, DIMENSION (N) :: ID
REAL (KIND=LONG), DIMENSION (N) :: X, Y
WIDTH = 1.0_LONG / N
ID = (/ (I, I=1, N)/)
X = (ID - 0.5) * WIDTH
Y = 4.0 / (1.0 + X*X)
PI = SUM(Y) * WIDTH
FORMAT ('ESTIMATION OF PI WITH',I6, & 'INTERVALS IS', F14.12)
PRINT 10, N, PI
END
```

Linguagens para Programação Paralela

- Extensões das linguagens seqüenciais através de compiladores especiais
  - Fortran D
  - Usuário pode especificar a decomposição dos dados
  - Mapeamento do problema - define arrays que podem estar no mesmo processador
    - ALIGN A1, A2 with P2
    - A1 e A2 deverão executar no processador P2
  - **Distribuição**

```
FORALL i = 1, 25 on HOME(x(I))
    x(I+ 25) = F(x(I))
ENDFOR
```

  - **Uma versão do Fortran D - HPF - High Performance Fortran**

Linguagens para Programação Paralela

- Extensões das linguagens seqüenciais através de compiladores especiais
  - C - Várias implementações
  - Sequent C
  - DYNIX - Unix + Rotinas para processamento paralelo
  - Variáveis Compartilhadas
  - SHARED INT a[10] - todos processadores compartilham a
  - Primitivas:
    - m_fork (name[.....]) - pai e filho executam função “name”
    - m_get_myid
    - m_get_numprocs
    - m_lock e m_unlock (criação de monitores)
    - m_kill_procs
    - m_set_procs

Linguagens para Programação Paralela

- Extensões das linguagens seqüenciais através de compiladores especiais

```
#include <stdio.h>
#include <parallel/microtask.h>
#include <parallel/parallel.h>

shared double pi;

main (argc, argv)
int argc, char *argv[];
{
    void computepi();
    int intervals;
    int numprocs;

    numprocs = atoi (argv[1]);
    intervals = atoi (argv[2]);
    m_set_procs (numprocs);
    pi=0.0;
    m_fork (computepi, numprocs, intervals);
    printf("Estimation of pi is %14.12f.\n",pi);
    m_kill_procs ();
    return 0;
}
```

Exemplo de código na linguagem *Sequent C*

Linguagens para Programação Paralela

- Linguagens específicas para Programação Paralela:
  - Possuem recursos para ativação e coordenação de processos mais naturais e com implementação mais eficientes;
  - Apresentam maior flexibilidade: permitem diferentes tipos de paralelismo;
  - Ferramentas para depuração e detecção de erros;
  - Normalmente apresentam melhor desempenho;
  - Baixa portabilidade;
  - Exemplos: Ada e Occam.

Linguagens para Programação Paralela

- Linguagem Ada:
  - Nome: Homenagem a Augusta Ada Byron – 1a. programadora
  - Unidade de Paralelismo:
    - *TASK*: Especificação - Interface
      Corpo - Declarações e Programas (*Tasks*)
    - *Tasks* especificadas no corpo de outra executam em paralelo com o pai;
  - Comunicação:
    - Variável compartilhada: para memória compartilhada - não aconselhável;
    - *Rendezvous*: Troca de Mensagens
      Receive - *Entry* (especificação)
      Accept (corpo)
      Send - chamada a rotina
      Aceita mensagem de qualquer fonte

Linguagens para Programação Paralela

- Linguagem Ada

```
task SharedCounter is
   entry Increment;
   entry Value (V: out Integer);
end;
task body SharedCounter is
  Count: Integer;
  begin
    Count :=0;
    loop
     select
       accept Increment do
         Count:= Count + 1;
       end Increment;
     or
       accept Value (V: out Integer)  do
         V := Count;
       end Value;
     end select;
    end loop;
  end SharedCounter;
```

```
loop
  select
   when NumberOfSpaces > 0 =>
     accept Send (Item: in ItemType) do
       ...
     end;
  or
   when NumberOfSpaces < BufferSize =>
     accept Receive (Item: out ItemType)
do
       ...
     end;
  end select;
end loop;
```

Linguagens para Programação Paralela

Linguagem Ada

- Fonte:
  - Especifica o destino
  - Fica suspenso até receber resposta (bloqueante)
- Comando Select
  - Comando de guarda
  - Seleciona uma alternativa entre várias para comandos accept
  - Escolha não determinística
- Exemplos

Linguagens para Programação Paralela

- Linguagem Ada - Características Gerais:
  - Rica e Poderosa;
  - Construções em Pascal + PC;
  - Não é user-friendly;
  - Tempo de comunicação alto;
  - Complexa;
  - Não possui ferramentas para distribuição de processos;

Linguagens para Programação Paralela

- Linguagem OCCAM
- Baseada em CSP (Hoare)
- Comunicação e Sincronismo
  - Troca de mensagens
  - Ponto a Ponto
  - Canais lógicos unidirecionais
  - Receive: ch? X (recebe o valor X pelo canal ch)
  - Send: ch!X (envia o valor de X pelo canal ch)
- Unidade de paralelismo:
  - Comandos
  - Ativação de Comandos em paralelo:

```
PAR
    Comando1
    Comando2
```

- Oferece ferramentas para mapeamento
- Exemplo

Linguagens para Programação Paralela

Bibliotecas para troca de mensagens:

- Oferecem o suporte necessário para o desenvolvimento de aplicações paralelas;
- Em linhas gerais são bibliotecas para comunicação entre processos;
- Extensão de linguagens de programação de propósito geral;
- Originalmente foram desenvolvidas para máquinas paralelas - MPP (*Massively Parallel Processing*);
- Não havia padrão→ Cada fabricante criava sua biblioteca para troca de mensagens;
- Problema: Ausência de portabilidade!
- Solução: Plataformas portáteis!

Linguagens para Programação Paralela

- Plataformas Portáteis:
  - Conjunto de funções independentes de máquina, que executam em diversas arquiteturas de hardware;
  - Aplicações recuperam a portabilidade perdida ao longo do desenvolvimento da Computação Paralela;
  - Possibilidade de utilização de ambientes heterogêneos - processadores diferentes / sistemas operacionais diferentes;
  - Exemplos de plataformas ou ambientes portáteis: P4, Parmacs, Express, Linda, PVM, etc;
  - Devido aos diversos ambientes disponíveis, há necessidade de padronização $\rightarrow$ MPI;

Ambientes para Troca de Mensagens

- Mensagem
- Transferência de Informação entre um elemento de processamento fonte e um destino
- Conteúdo: Origem, Destino, Dados, Tipo dos dados, Tamanho dos dados, Onde o dado será armazenado no receptor, Qual a quantidade suportada pelo receptor, etc.
- Tipo de mensagem – Identificador associado a cada mensagem para selecionar uma mensagem particular para recebimento.

Ambientes para Troca de Mensagens

- Uma biblioteca “genérica” para troca de mensagens deve manipular:
  - Identificação de processos;
  - Comunicação “Ponto-a-Ponto”;
  - Comunicação Coletiva;
  - Verificação;
  - Informação.

Ambientes para Troca de Mensagens

- Funções para Identificação de Processos:
  - `myid()` → retorna o identificador de processo corrente;
  - `nprocs()` → retorna o número de processos em uma aplicação;

Exemplo:

```
program hello
    print("eu sou=",myid(), "# de
          processos=",nprocs())
    stop
end
```

Saída:

```
eu sou=2 # processos=4
eu sou=3 # processos=4
eu sou=0 # processos=4
eu sou=1 # processos=4
```

  - Importante para aplicações SPMD.

Ambientes para Troca de Mensagens

- Funções de comunicação Ponto-a-Ponto:
  - `send(msgtype, x, tamanho, destino)` → envia a mensagem `<x>` para `<destino>`;
  - Alguns tipos de funções *SEND*
    - *Send* bloqueante: Copia `<x>` no buffer de mensagens e espera até que o buffer possa ser reutilizado com segurança;
    - *Send* não bloqueante: Copia `<x>` no buffer de mensagens e libera o processo para desenvolver outras operações;
  - Existem ainda os tipos **síncrono** e **assíncrono**.

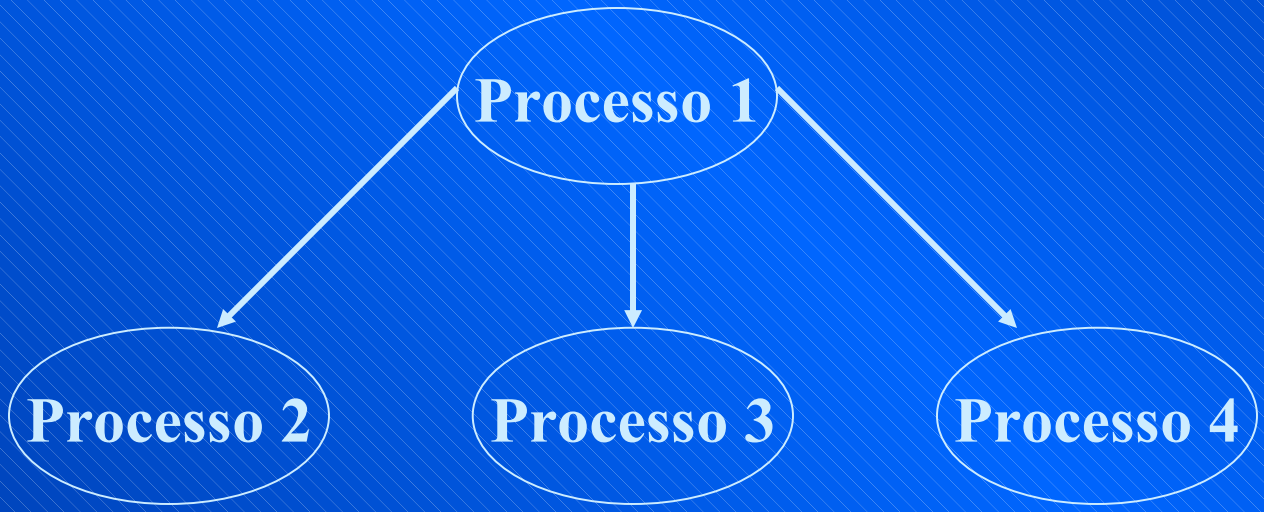
Ambientes para Troca de Mensagens

- Funções de comunicação Ponto a Ponto:
  - `receive (msgtype, x, tamanho)` → Recebe uma mensagem do tipo `<msgtype>` e copia esta em `<x>`;
  - Alguns tipos de funções *RECEIVE*
    - *Receive* bloqueante: Se a mensagem não chegou, bloqueia-se até que chegue. Copia a mensagem recebida na área de dados `<x>`;
    - *Receive* não bloqueante:Se a mensagem chegou, copie para a área de dados `<x>`. Se ainda não chegou, adie a recepção.

Ambientes para Troca de Mensagens

- Funções de comunicação coletiva:

  `bcast(msgtype, x, tamanho, raiz)` → Envia uma mensagem de um processo `<raiz>` para todos os processos;



Ambientes para Troca de Mensagens

- Funções de comunicação coletiva:
  - `bcast(msgtype, x, tamanho, raiz)` → Envia uma mensagem de um processo `<raiz>` para todos os processos;
  - Esta rotina é normalmente bloqueante
  - Todos os processos devem executar a operação de recebimento
  - Se não executar -> deadlock

Ambientes para Troca de Mensagens

- Funções de verificação:
  - `probe (tag_msg)` → Verifica se a mensagem identificada por `<tag_msg>` existe na fila de mensagens;
    - Não é uma operação bloqueante;
    - Pode retornar 1 se a mensagem existe na fila e 0 caso contrário.

Ambientes para Troca de Mensagens

- Funções Informativas:
  - `info_len()` → Retorna o tamanho da mensagem;
  - `info_type()` → Retorna o tipo da mensagem.

- Funções de sincronização:
  - `barrier ()` → Todo o processo que executa "barrier" fica aguardando até que os demais também cheguem a esse mesmo ponto.

Exemplos de Ambientes para Troca de Mensagens

- RPROC - 1982
  - Já possuía características como:
    - Uso em plataformas heterogêneas;
    - Mensagens ativas;
    - Conversão e empacotamento de dados;
  - Principal objetivo: Interconectar computadores que possuíam diferentes sistemas operacionais;
  - Desenvolvido para mais de dez arquiteturas de computadores.

Exemplos de Ambientes para Troca de Mensagens

- P4 - Argonne National Laboratory
  - 1ª tentativa de plataforma portátil
    - 1984 - Monmacs;
    - 1989 - Reescrito para máquinas heterogêneas;
  - Biblioteca de Macros;
  - Eficiência: Cada implementação do P4 explora a plataforma escolhida;
  - Simplicidade: Número reduzido de conceitos, mas suficientes para uma gama de algoritmos paralelos;
  - Paradigmas de Programação: SPMD e Mestre-Escravo.

Exemplos de Ambientes para Troca de Mensagens

- Parmacs - Argonne National Laboratory
  - 1987 - Herdou características do P4;
  - Primeira biblioteca de passagem de mensagens em C;
  - Migrou de Macros para Funções;
  - Modelo de Programação:
    - Memória Distribuída;
    - Comunicação por troca de mensagens;
  - Suporte para ambientes heterogêneos - XDR (*eXternal Data Representation*).

Exemplos de Ambientes para Troca de Mensagens

- Express (Parasoft)
  - Baseado no *Crystalline Operating System* - Caltech;
  - Conjunto de ferramentas para o desenvolvimento de software paralelo;
  - Características importantes:
    - Balanceamento de carga dinâmico;
    - E/S Paralelo;

Exemplos de Ambientes para Troca de Mensagens

- Linda - Yale University
  - 1ª versão - 1980;
  - Abordagem diferente dos demais sistemas - espaço de tuplas;
  - Modelo de programação Mestre/Escravo;
  - Atualmente:
    - Serviu de base para a definição de *JavaSpaces*;

Exemplos de Ambientes para Troca de Mensagens

- PVM - *Parallel Virtual Machine*
  - 1ª versão - 1989;
  - ORNL, Univ. of Tenessee e Emory University;
  - Conjunto integrado de bibliotecas e ferramentas de software;
  - Emula um sistema de computação concorrente, flexível e de propósito geral;
  - Padrão "de fato";
  - Paradigmas de Programação adotados:
    - SPMD $\rightarrow$ Paralelismo de Dados;
    - MPMD $\rightarrow$ Paralelismo Funcional.

Exemplos de Ambientes para Troca de Mensagens

- PVM - *Parallel Virtual Machine*
  - Possibilita o desenvolvimento de aplicações em C/C++ e Fortran;
  - Todos os programas PVM devem ser "linkados" com a biblioteca PVM;
  - É um ambiente *freeware* e *open-source*;
  - Existem diversas ferramentas para "depurar" programas PVM e também verificar o estado da máquina paralela virtual.

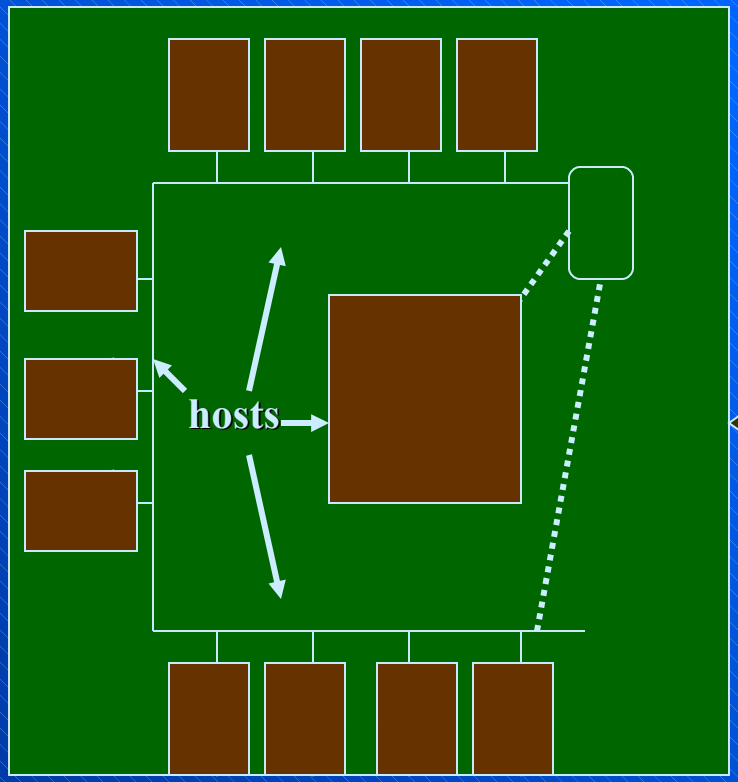
Exemplos de Ambientes para Troca de Mensagens

- PVM - *Parallel Virtual Machine*
  - implementado em diversas plataformas:

Alliant FX/8	DEC Alpha
Sequent Balance	Bbn butterfly TC2000
80386/486/Pentium com Unix (Linux ou BSD)	Thinking Machines
Convex C-Series	C-90. Ymp, Cray-2 e Cray S-MP
HP 9000	Intel Paragon
DECstation 3100, 5100	IBM/RS6000
Sillicon Graphics	DEV Micro VAX
Sun 3, Sun 4, SPARCStation	PVM-W95

Exemplos de Ambientes para Troca de Mensagens

- PVM - *Parallel Virtual Machine*



Visão Uniforme de uma máquina virtual paralela

Visão arquitetural

Exemplos de Ambientes para Troca de Mensagens

- MPI - *Message Passing Interface*
  - Início: 1992;
  - Proposta de um **Padrão** de interface de passagem de mensagens para computadores com memória distribuída e NOWs (*Networks Of Workstations*);
  - Objetivos:
    - Unir portabilidade e facilidade de uso;
    - Fornecer uma especificação precisa para o desenvolvimento de ambientes de passagem de mensagem;
    - Possível crescimento da indústria de software paralelo;
    - Difusão do uso de computadores paralelos.

Exemplos de Ambientes para Troca de Mensagens

- MPI - *Message Passing Interface*
  - Padrão "de direito" para o desenvolvimento de aplicações paralelas distribuídas;