

Universidade Federal do Rio
de Janeiro
Instituto de Matemática
Departamento de Ciência da
Computação



MPI I/O

Parte 1

Vinicius Silva
viniciussilva@nce.ufrj.br



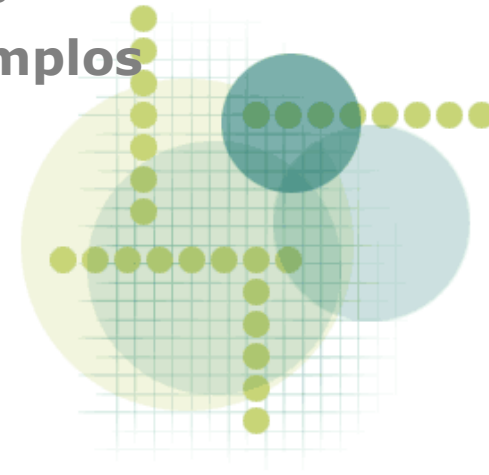
Organização

- **I/O Paralelo - Introdução**

- **Importância**
- **Requisitos**
- **RAID**
- **Sistemas de arquivos distribuídos**
 - NFS
- **Sistemas de arquivos paralelos**
 - PVFS
- **I/O em programas paralelos**
 - Single process I/O
 - Remontagem *post-mortem*

- **MPI I/O – Parte 1**

- **História**
- **Motivação**
- **Características básicas**
- **Terminologia**
- **Funções básicas**
- **Exemplos**



I/O Paralelo – Importância

- **Definição:**
 - **Supercomputador, sm:** *Um computador que transforma um problema CPU-bound em um problema I/O-bound.*
- À medida que os supercomputadores se tornam mais rápidos e mais paralelos, o I/O se torna um gargalo.



Requisitos de I/O em computadores paralelos

- **Alto desempenho**
 - Obter vantagens das arquiteturas paralelas.
 - Permitir o refinamento da performance no nível da aplicação.
- **Integridade**
- **Única imagem do sistema**
 - Todos os nós vêm os mesmos sistemas de arquivos.
- **Facilidade de uso**
 - Sempre que possível, um sistema I/O paralelo deve ser facilmente acessível.



RAID

- Desenvolvido no final da década de 80 como uma solução à crescente demanda por capacidade de armazenamento, performance e redundância.
- Para aumentar a capacidade, um disco lógico é mapeado em um conjunto (*array*) de discos, embora visto como um sistema único.



RAID

- A melhoria na performance é obtida repartindo o dado a ser gravado em blocos menores e efetuando a gravação de cada bloco em um disco independente (*striping*). Assim, as operações de escrita e leitura do mesmo arquivo podem ser feitas de forma paralela.
- Alguns tipos comuns de RAID:
 - RAID 0 – apenas *striping*.
 - RAID 1 – espelhamento; escreve o mesmo dado em N discos.
 - RAID 5 – *striping* e paridade.



Sistemas de arquivos distribuídos

- Um sistema de arquivos distribuído é um sistema de arquivos armazenado localmente em um servidor e acessível por processos em outros sistemas (clientes).
- Alguns exemplos de sistemas de arquivos distribuídos:
 - NFS (Sun)
 - AFS (Andrew File System – Carnegie Mellon University)
 - DCE/DFS (Distributed File System - Transarc/IBM)
 - Coda (Carnegie Mellon University)
 - InterMezzo (Carnegie Mellon University / Stelias)
 - LegionFS (University of Virginia)



Sistemas de arquivos distribuídos

- **Sistemas de arquivos distribuídos podem ser utilizados por programas paralelos, mas eles têm desvantagens significativas:**
 - A largura de banda do servidor é um fator limitante da performance.
 - Para manter a compatibilidade semântica com os sistemas de arquivos UNIX-like, alguma forma de locking deve ser implementada (outro fator com impacto negativo na performance).



Sistemas de arquivos distribuídos - NFS

- **Único servidor, múltiplos clientes**
 - Mesmo servidor pode exportar vários sistemas de arquivos.
 - Clientes podem montar sistemas de arquivos de diferentes clientes.
- **Transporte: RPC sobre UDP/IP**
 - Conexão sem estado.
 - Necessário possuir um mecanismo de retransmissão.



Sistemas de arquivos distribuídos - NFS

- **NFS v3 adicionou algumas funcionalidades:**
 - Leituras e escritas assíncronas.
 - Locking (`rcp.lockdandrpc.statd`).
 - Suporte a arquivos maiores de 2GB em arquiteturas de 32-bit
 - Alternativa de utilizar RPC sobre TCP/IP como transporte.
- **NFS não implementa:**
 - Replicação do arquivo no lado do cliente.
 - ACLs (Access Control Lists – Listas de Controle de Acesso).



Sistemas de arquivos paralelos

- Sistema no qual existem múltiplos servidores atendendo a múltiplos clientes, para um dado sistema de arquivos.
- Sistemas de arquivos paralelos são otimizados para alto desempenho em detrimento a uso geral (embora você possa, provavelmente não vai armazenar seu diretório *home* em um deles).



Sistemas de arquivos paralelos

- **Características**

- Blocos longos ($\geq 64\text{KB}$).
- Operações sobre metadados relativamente lentas quando comparadas a leituras e escritas.
- APIs específicas para acesso.

- **Exemplos**

- GFS (Sistina)
- GPFS (IBM)
- PFS (Intel)
- PVFS (Clemson/ANL)
- Lustre (SUN)



Sistemas de arquivos paralelos - PVFS

- **Arquitetura com 3 componentes:**
 - 1 Servidor de metadados.
 - Múltiplos servidores de dados, cada um utilizando um sistema de arquivos local.
 - Múltiplos clientes.
- Um mesmo nó pode acumular papéis, i.e., pode atuar como servidor e/ou cliente e/ou servidor de metadados.
- “Equivalente” a RAID 0.



Sistemas de arquivos paralelos - PVFS

- **Não implementa redundância – a perda um servidor de dados significa a perda de todo o sistema de arquivos.**
- **Transporte via sockets TCP/IP, embora outras formas de transporte estejam em desenvolvimento.**



I/O em programas paralelos

Single Process I/O

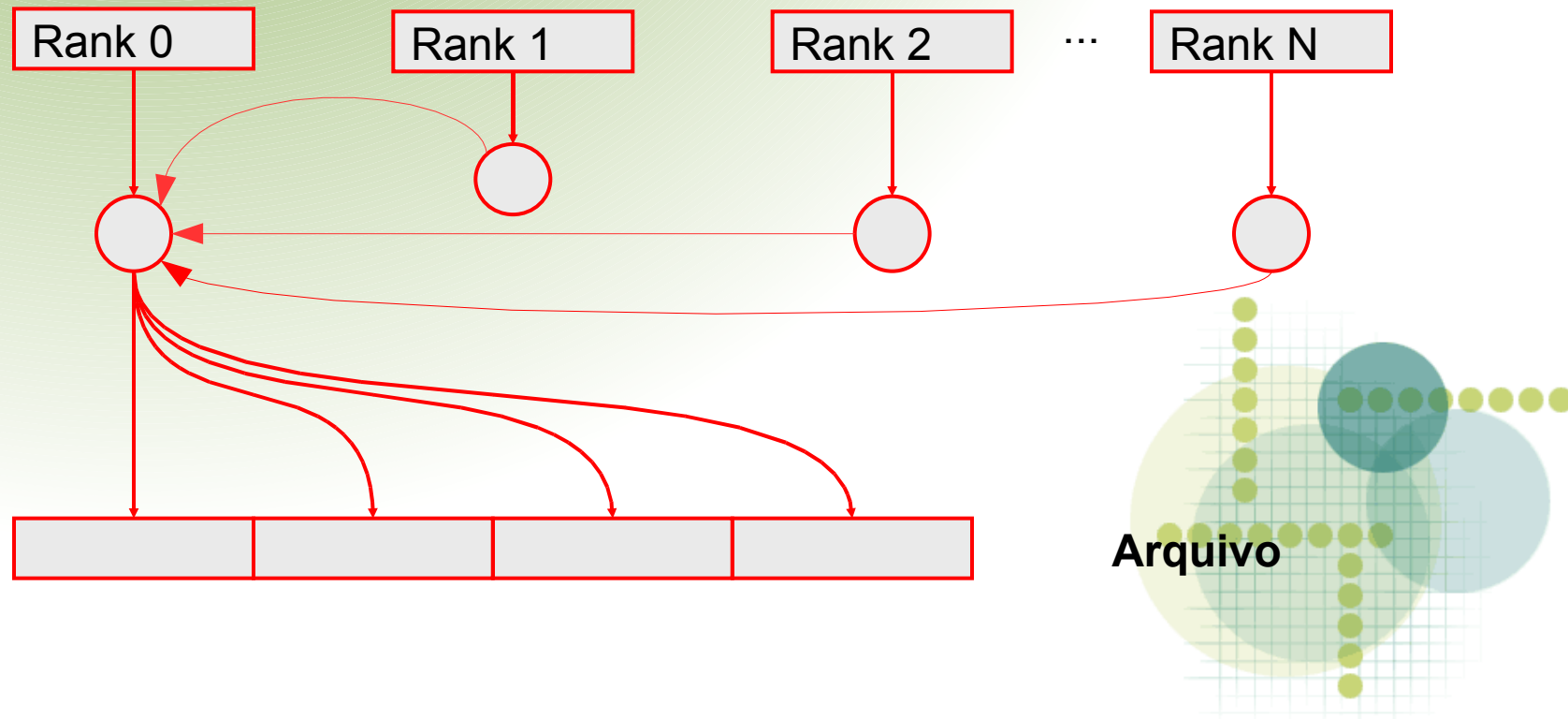
- **Somente um processo ou thread da aplicação paralela (normalmente rank=0) realiza I/O.**
 - Dados globais são enviados para outros processo através de broadcast (utilizando `MPI_Bcast()`, p.e.).
 - Dados locais são distribuídos utilizando outras funções de troca de mensagem, como `MPI_Send()` e `MPI_Recv()` ou `MPI_Scatter()` e `MPI_Gather()`.



I/O em programas paralelos

Single Process I/O

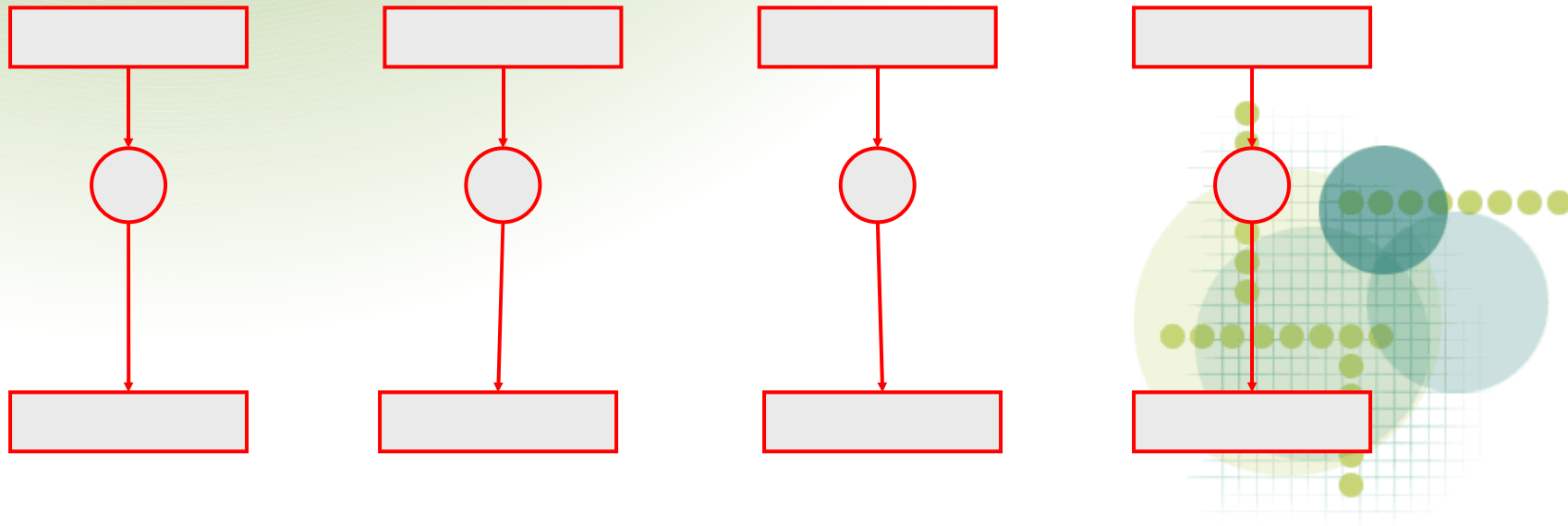
- Todos os processos enviam dados para o processo de rank 0 e ele realiza o I/O.



I/O em programas paralelos

Remontagem *post-mortem*

- Na remontagem *post-mortem*, cada processo ou thread da aplicação paralela lê e escreve em seus próprios arquivos. A saída de cada processo é remontada após a aplicação terminar utilizando-se um programa separado.



I/O em programas paralelos

Remontagem *post-mortem*

- **Vantagens:**

- **Paralelismo.** Permite que cada processo acesse sua porção do arquivo independentemente.

- **Desvantagens:**

- **Requer o desenvolvimento de uma ferramenta de remontagem.**
- **Remontar centenas ou mesmo milhares de arquivos de saída pode custar muito tempo.**
- **Não oferece muita possibilidade de uso coordenado com operações coletivas.**



Organização

- **I/O Paralelo - Introdução**

- **Importância**
- **Requisitos**
- **RAID**
- **Sistemas de arquivos distribuídos**
 - NFS
- **Sistemas de arquivos paralelos**
 - PVFS
- **I/O em programas paralelos**
 - Single process I/O
 - Remontagem *post-mortem*

- **MPI I/O – Parte 1**

- **História**
- **Motivação**
- **Características básicas**
- **Terminologia**
- **Funções básicas**
- **Exemplos**



História

- **1994:** Inicialmente desenvolvido no IBM T. J. Watson Research Center como projeto de pesquisa.
- **1996:** MPI Forum vota por adicionar um subcomitê de I/O paralelo. Originalmente não foi considerado pertencente ao escopo do projeto MPI-2.
- **1996:** MPI-IO adicionado como um capítulo no rascunho do padrão MPI-2.
- **1997:** Publicado no padrão MPI-2.
- **1998:** Implementações comerciais e de código-aberto se tornam disponíveis.



Motivação

- **Portabilidade**

- Provê uma interface padrão para I/O paralelo que não existe de outra forma.
- Interface tradicional do UNIX não é adequada para I/O paralelo.
- Produtos proprietários geralmente não são portáteis.



Motivação

- **MPI-like**

- Possui o "jeito" MPI, i.e., sintaxe de funções e semântica parecidas. O programador com alguma experiência com o MPI rapidamente incorpora os recursos do MPI I/O à sua aplicação.
- **Escrever é como enviar e ler como receber uma mensagem.**
- Versatilidade dos MPI *datatypes* (abstração de dados).



Motivação

- **Eficiência / Performance**

- Operações coletivas de I/O resumem várias pequenas operações de acesso aos dados em uma única operação.
- Provê “*user hints*” para otimizações no nível da aplicação.
- Sobreposição de computação e I/O proporcionado por operações de I/O não-bloqueantes.

- **Interoperabilidade**

- Provê um padrão para representação de dados.
- Suporte para representações de dados do usuário.



Características Básicas

- **Vistas de arquivos (File Views)**
 - Vistas (views) definem que parte de um arquivo cada processo irá ver e manipular.
 - Processos podem ver dados de forma independente ou sobreposta.
 - Posicionamento dos dados através de tipos de dados definidos. Permite acesso não-contíguo.
 - Vistas podem ser modificadas a qualquer momento.



Características Básicas

- **Posicionamento**

- Acesso aos dados através de ponteiros individuais.
- Acesso coletivo aos dados através de ponteiros compartilhados.

- **Sincronização**

- Operações bloqueantes e não-bloqueantes.

- **Coordenação de acesso**

- Acesso não-coordenado ("free-for-all") ou seqüencial.
- Acesso coordenado através de operações coletivas (ordenadas através da ordem do rank).



Características Básicas

- **Interoperabilidade**

- Provê meios de garantir a portabilidade da representação dos dados.
 - Native – ambientes homogêneos.
 - Internal – ambientes heterogêneos utilizando uma representação de dados definida.
 - External32 – ambientes heterogêneos utilizando uma representação de dados padrão.
 - User defined – para uso fora de uma implementação MPI.

- **Suporte a C/C++ e Fortran**



Terminologia

- **Arquivo (file)**

- Um arquivo MPI é uma coleção ordenada de itens de dados denominados *etypes*. Os itens de dados podem ser predefinidos (como *byte*, *integer* ou *float*) ou definidos pelo usuário.
- O MPI suporta acesso seqüencial ou aleatório a esses tipos.
- Um arquivo é aberto coletivamente por um grupo de processos.
- O acesso aos dados do arquivo pode ser realizado de forma coletiva ou não-coletiva.



Terminologia

- ***etype* (elementary datatype)**
 - Um *etype* é a unidade de dados que é acessada ou posicionada. Pode ser um tipo MPI predefinido ou derivado.
 - O acesso aos dados é realizado em unidades de *etypes*.
 - Offsets são expressos em função de unidades de *etypes*.
 - Ponteiros para arquivos apontam para o início dos *etypes*.



Terminologia

- ***filetype***

- Um *filetype* é a base para particionar um arquivo através dos processos e definir um *template* para acessar o arquivo.
- É constituído de um número de *etypes* e '*holes*' (que deve ser um múltiplo do tamanho do *etype*).
- O *filetype* básico é repetido várias vezes preenchendo o arquivo e criando regiões de acesso permitido (onde os *etypes* estão definidos) e de acesso restrito (onde os *holes* estão definidos).



Terminologia

- **vista (*view*)**
 - Uma vista define um conjunto ordenado de dados (*etypes*) visível e acessível a um processo.
 - Cada processo tem sua própria vista do arquivo, definido em função de três parâmetros:
 - Deslocamento (*displacement*);
 - Um *etype*;
 - Um *filetype*.
- O padrão definido no *filetype* é repetido, a partir do deslocamento, para definir a vista.

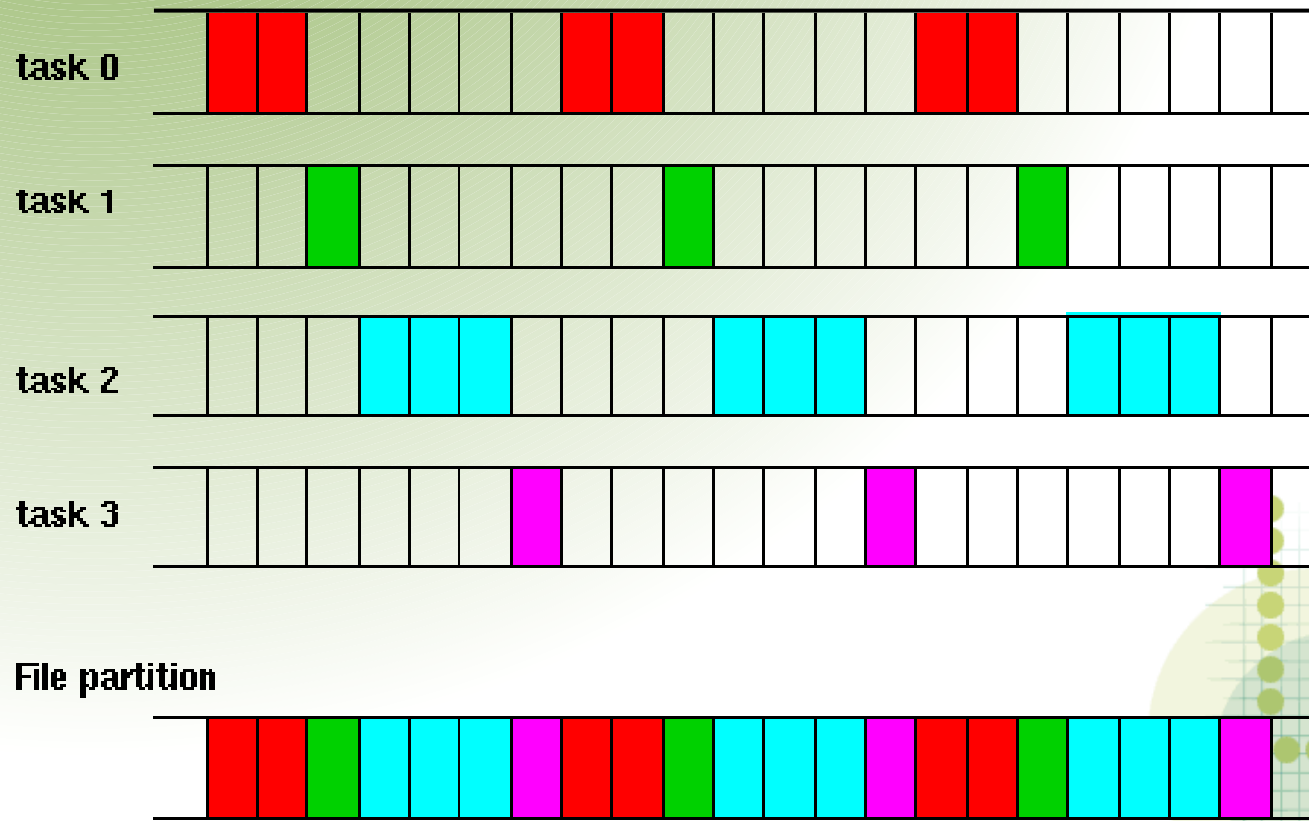


Terminologia

- As vistas podem ser alteradas pelo usuário no tempo de execução. A vista padrão é um fluxo linear de bytes.
- A figura a seguir demonstra como múltiplos processos, usando vistas diferentes, compostas de *filetypes* complementares, podem ser utilizadas para particionar os dados de forma eficiente. Cada bloco individual é um *etype*. Cada grupo colorido representa um *filetype*.

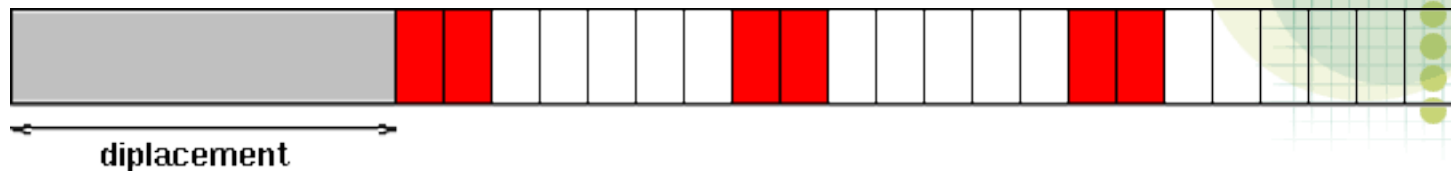


Terminologia



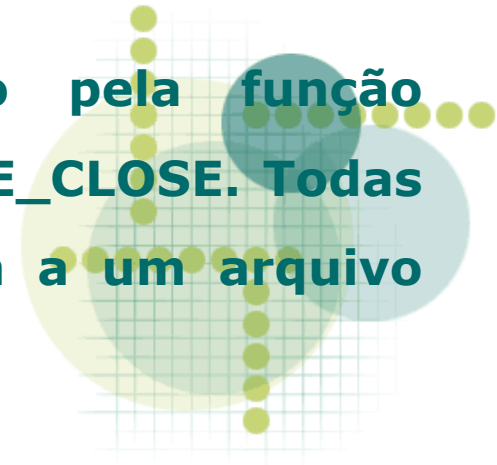
Terminologia

- ***offset***
 - Um *offset* é uma posição, medida em unidades de *etypes*, no arquivo relativo a vista corrente (*holes* não são considerados).
 - O *offset* 0 é a posição do primeiro *etype* da vista.
- ***deslocamento (displacement)***
 - Posição absoluta, em bytes, a partir do início do arquivo.
 - O deslocamento define onde uma vista começa.
 - Útil para saltar um cabeçalho ou uma região do arquivo já acessada como diferentes *filetypes*.



Terminologia

- **ponteiro de arquivo (*file pointer*)**
 - Um ponteiro de arquivo é um *offset* implícito mantido pelo MPI. Ponteiros de arquivos individuais são locais para cada processo que abriu o arquivo.
 - Um ponteiro de arquivo coletivo é compartilhado pelo grupo de processos que abriu o arquivo.
- **file handle**
 - Um file handle é um objeto criado pela função `MPI_FILE_OPEN` e destruído por `MPI_FILE_CLOSE`. Todas as operações feitas na referência aberta a um arquivo ocorrem através do file handle.



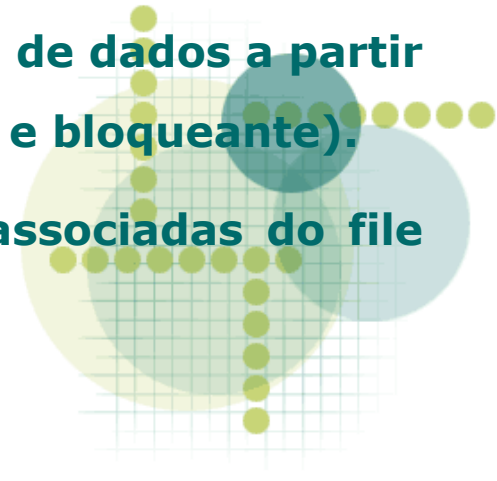
Funções Básicas

- No total existem 15 funções para ler e 15 para escrever em arquivos, além de inúmeras outras para criação de vistas, estrutura de dados etc.
- No entanto, a estrutura dessas funções é bem parecida. Em geral, são variações para lidar com diferentes tipos de posicionamento (ponteiros individuais ou coletivos), sincronismo (bloqueante ou não-bloqueante) e coordenação (coletivas e não-coletivas).



Funções Básicas

- Nesta parte, serão apresentadas as seis funções básicas para manipulação de arquivos no MPI.
 - `MPI_File_open()` – associa um *file handle* a um arquivo.
 - `MPI_File_seek()` – move a posição do ponteiro no arquivo.
 - `MPI_File_read()` – lê uma quantidade fixa de dados a partir da posição do ponteiro do arquivo. (não coletiva e bloqueante).
 - `MPI_File_write()` – escreve uma quantidade fixa de dados a partir da posição do ponteiro do arquivo. (não coletiva e bloqueante).
 - `MPI_File_sync()` – força a escrita das caches associadas do *file handle* no sistema de arquivos.
 - `MPI_File_close()` – fecha o arquivo.



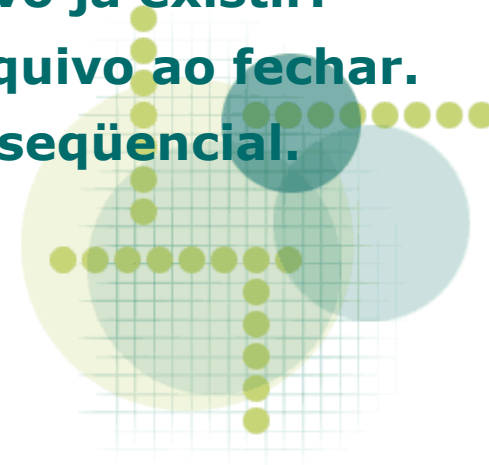
MPI_File_open

- **int MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh)**
 - **fh** é o *file handle*, que será usado por todas as outras funções MPI-IO para referenciar o arquivo.
 - **filename** é uma string do nome do arquivo que será aberto. O caminho relativo ou absoluto pode ser especificado.



MPI_File_open

- ***amode*** especifica o modo de acesso ao arquivo. Pode ser uma combinação dos modos abaixo. Para combinar mais de um modo, utilize o operador *or* binário.
 - MPI_MODE_RDONLY – somente leitura.
 - MPI_MODE_RDWR – leitura e escrita.
 - MPI_MODE_WRONLY – somente escrita.
 - MPI_MODE_CREATE – cria o arquivo, caso não exista.
 - MPI_MODE_EXCL – retorna erro se o arquivo já existir.
 - MPI_MODE_DELETE_ON_CLOSE – apaga arquivo ao fechar.
 - MPI_MODE_SEQUENTIAL – apenas acesso seqüencial.



MPI_File_open

- **MPI_MODE_UNIQUE_OPEN** – não permite que outros processos acessem o arquivo.
- **MPI_MODE_APPEND** – define a posição inicial do ponteiro de arquivo no final do arquivo.
- **info** é um conjunto de file hints. Sua criação será discutida posteriormente. Por enquanto, utilizar **MPI_INFO_NULL**.
- Essas funções operam sobre os ponteiros individuais de cada processo MPI. Ponteiros compartilhados serão discutidos posteriormente.



MPI_File_seek

- **int MPI_File_seek(MPI_File fh, MPI_Offset offset, int whence)**
 - **offset** determina o deslocamento do ponteiro a partir da posição atual. Esse valor pode ser negativo, embora deslocar o ponteiro para posições anteriores ao início do arquivo (ou da vista) gere um erro.



MPI_File_seek

- **whence** determina com o ponteiro será atualizado:
 - **MPI_SEEK_SET** – o ponteiro é atualizado para offset.
 - **MPI_SEEK_CUR** – o ponteiro é atualizado para a posição atual + offset.
 - **MPI_SEEK_END** – o ponteiro é atualizado para o final do arquivo + offset.
- Os deslocamentos são realizados em termos do tipo de dados atual (**MPI_BYTE** por padrão).



MPI_File_read

- **int MPI_File_read(MPI_File fh, void *buf, int count, MPI_Datatype type, MPI_Status *status)**
 - Lê *count* valores do tipo *datatype* do arquivo associado a *fh* para *buf*. O buffer *buf* deve poder armazenar pelo menos tantos valores quanto **count*sizeof(type)**.
 - *status* pode ser utilizado para monitorar quantos bytes foram lidos até o momento.



MPI_File_write

- **int MPI_File_write(MPI_File fh, void *buf, int count, MPI_Datatype type, MPI_Status *status)**
 - Lê *count* valores do tipo *datatype* de *buf* para arquivo associado a *fh*. O buffer *buf* deve poder armazenar pelo menos tantos valores quanto *count*sizeof(type)*.
 - *status* pode ser utilizado para monitorar quantos bytes foram escritos até o momento.



MPI_File_sync

- **int MPI_File_sync(MPI_File fh)**
 - Força todos os caches associados ao *file handle* serem escritos imediatamente no disco.
 - Pode ser custoso em sistemas de arquivos lentos.
 - Função coletiva, i.e., deve ser chamada por todos os processos associados ao arquivo.



MPI_File_close

- **int MPI_File_close(MPI_File *fh)**
 - Fecha o arquivo associado ao *file handle* **fh**.
 - Função coletiva, i.e., deve ser chamada por todos os processos associados ao arquivo.

