CAP-387(2016) – Tópicos Especiais em Computação Aplicada: Construção de Aplicações Massivamente Paralelas

Aula 44: MPI IO

Celso L. Mendes, Stephan Stephany

LAC / INPE

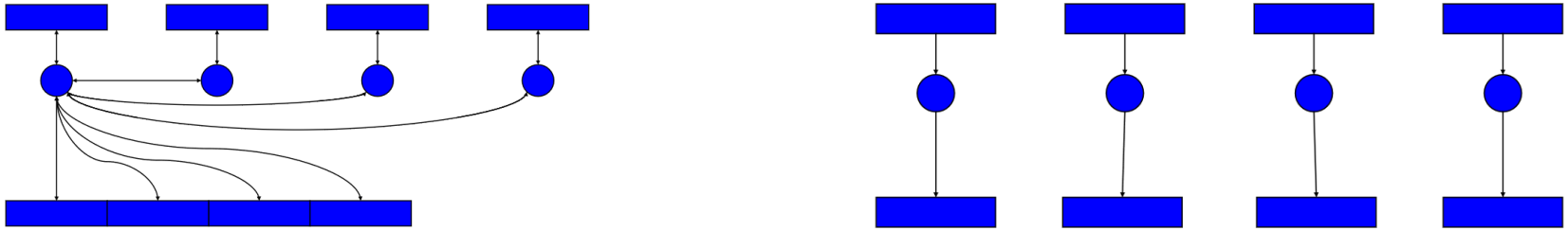
Emails: celso.mendes@inpe.br, stephan.stephany@inpe.br

MPI IO - Introdução

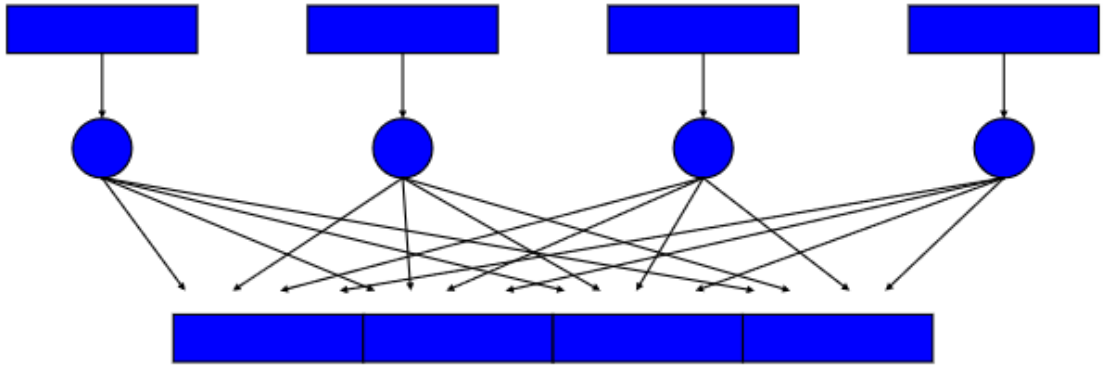
- MPI IO:
  - Funções de I/O definidas no padrão MPI (em MPI-2)
  - Motivação:
    - Melhor desempenho paralelo
    - Arquivo único (ao invés de um por processo)
    - Migração fácil para códigos usando Posix I/O
    - Suporte para múltiplos estilos de I/O
  - Implementação modelo: ROMIO (ANL)
    - Distribuída como parte de MPICH

MPI IO – Visão Geral

- Revendo: I/O via Posix



- MPI IO: E/S Cooperativa



- Paralelismo presente
- Um único arquivo
- Acesso pode ser otimizado pela biblioteca MPI

Conveniência de MPI IO

- Semelhanças de funcionalidades:
  - *write* é "similar" a enviar mensagem, *read* a receber
  - I/O paralelo pode usar operações coletivas
  - I/O paralelo pode usar operações sem bloqueio
  - Tipos de dados podem ser usados em I/O para definir estruturas dos dados na memória ou num arquivo
  - Comunicadores podem ser usados em mensagens relacionadas a I/O
- Infraestrutura de MPI: pode ser bem conveniente

Significado de I/O Paralelo

- Ao nível do programa:
  - *writes e reads* concorrentes a um arquivo comum, a partir de múltiplos processos do programa
- Ao nível do sistema:
  - Sistema de arquivos paralelo, e hardware para suportar os acessos concorrentes
  - Vários tipos de implementações possíveis

Estrutura de MPI IO

- I/O Independente:
  - Funções de I/O invocadas por cada processo individualmente
  - Modo similar a Posix I/O
- I/O Coletivo:
  - Funções coletivas, invocadas por todos os processos de um comunicador
  - Implementação (biblioteca MPI) pode otimizar desempenho

I/O Independente em MPI IO

- Funções Básicas: Similar a Posix I/O
  - Abrir arquivo: *MPI_File_open()*
  - Escrever arquivo: *MPI_File_write()*
  - Posicionar no arquivo: *MPI_File_seek()*
  - Ler arquivo: *MPI_File_read()*
  - Fechar arquivo: *MPI_File_close()*
- Argumentos:
  - Similares a Posix I/O, mais comunicador/status/info

MPI IO – Exemplo Básico

```
#include  <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
        MPI_File fh;
        int buf [1000], rank;
        MPI_Init (0,0);
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
        MPI_File_open(MPI_COMM_WORLD, "test.out",
                MPI_MODE_CREATE|MPI_MODE_WRONLY,
                MPI_INFO_NULL, &fh);
        if (rank == 0)
            MPI_File_write(fh, buf, 1000, MPI_INT, MPI_STATUS_IGNORE);
        MPI_File_close(&fh);
        MPI_Finalize();
        return 0;
}
```

Operação Independente

Operações Coletivas

Escrita em Arquivo com MPI IO

- Funções Principais:
  - *MPI_File_write()*: escrita a partir da posição atual
  - *MPI_File_write_at()*: escrita passando posição desejada
- Flags de modo:
  - *MPI_MODE_WRONLY, MPI_MODE_RDWR,* etc
  - Múltiplos flags compostos via "|" em C (em Fortran: +)
  - Se o arquivo não existe ainda, o flag *MPI_MODE_CREATE* deve ser passado em *MPI_File_open()*

Formas de Acesso

- *MPI_File_seek*
- *MPI_File_read*
- *MPI_File_write*
- *MPI_File_read_at*
- *MPI_File_write_at*
- *MPI_File_read_shared*
- *MPI_File_write_shared*

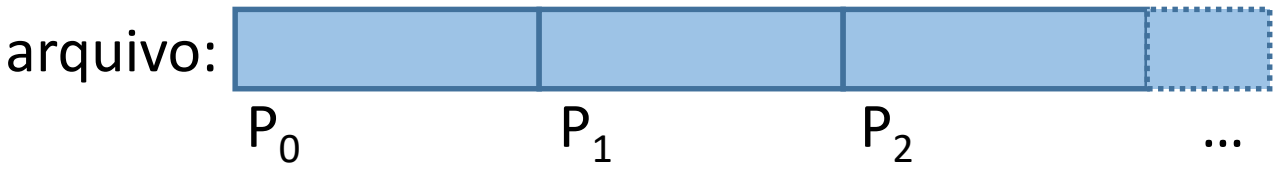
Similar a Posix I/O

Combinam *seek* e I/O (são *thread-safe*)

Similar a Posix I/O

Uso de Offsets Específicos

```
#include  <stdio.h>
#include "mpi.h"
int main(int argc, char *argv[])
{
        MPI_File fh;
        MPI_Status status;
        MPI_Offset offset;
        int buf [1000], rank, nprocs;
        MPI_Init (0,0);
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
        MPI_Comm_size (MPI_COMM_WORLD, &nprocs);
        MPI_File_open(MPI_COMM_WORLD, "/pfs/arquivo",
                        MPI_MODE_RDONLY, MPI_INFO_NULL, &fh);
        nints = FILESIZE / (nprocs*INTSIZE);
        offset = rank * nints * INTSIZE;
        MPI_File_read_at(fh, offset, buf, nints, MPI_INT, &status);
        MPI_Get_count(&status, MPI_INT, &count);
        printf("rank %d leu %d ints\n",rank,count);
        MPI_File_close(&fh);
        MPI_Finalize();
        return 0;
}
```



Operação Independente

Operações Coletivas

Acesso a Dados Não-Contíguos

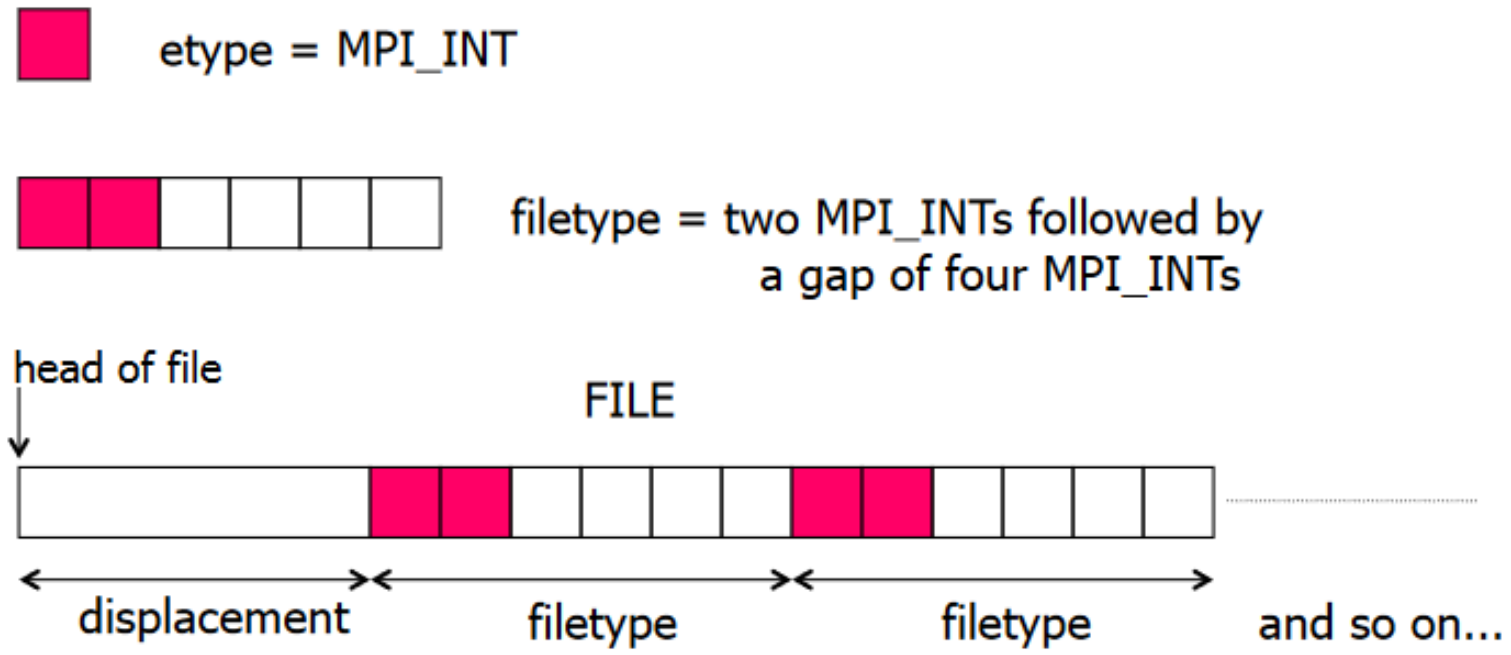
- Cada processo enxerga partes do arquivo
  - O conjunto destas partes chama-se *"file-view"*
    - Indica as partes do arquivo às quais o processo tem acesso
    - *reads* e *writes* do processo são sempre sobre sua file-view
  - Descrito em MPI por um offset (útil para pular cabeçalhos) e um *MPI_Datatype* (tipo de dados)
  - A *file-view* é útil para se definir acessos não-contíguos

File-View em MPI IO

- Especificação da *file-view*:
  - Definida por *{displacement, etype, filetype}*
    - *displacement*: número de bytes a serem pulados no início do arquivo
    - *etype:* unidade básica de acesso (tipo básico ou derivado)
    - *filetype:* indica quais partes são vistas pelo processo
  - Configurada com chamada a *MPI_File_set_view()*

File-View em MPI IO (cont.)

- Exemplo:

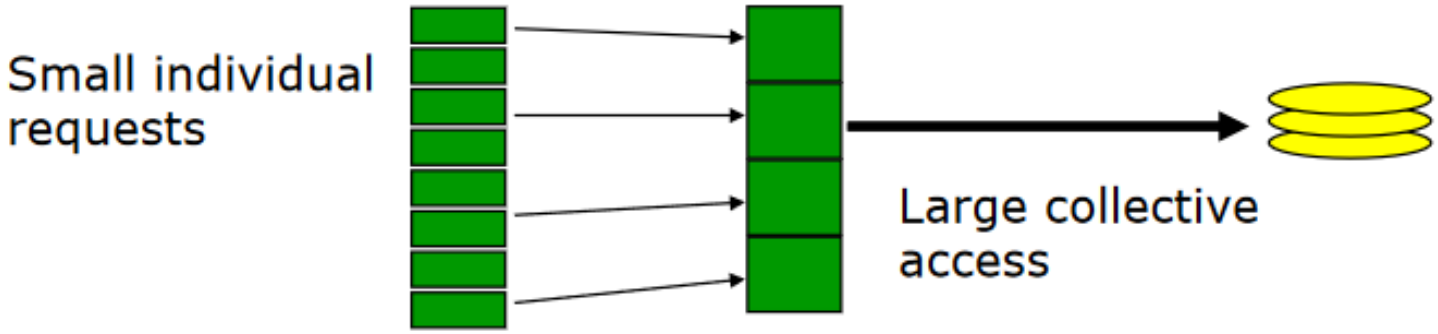


Uso da *File-View* do Exemplo

```
MPI_File fh;
MPI_Aint lb, extent;
MPI_Offset disp;
MPI_Datatype etype, filetype, config;
int buf[1000];
MPI_Type_contiguous(2,MPI_INT, &config);        << cria tipo de 2 inteiros >>
lb=0; extent=6*sizeof(int);
MPI_Type_create_resized(config, lb, extent, &filetype);   << cria tipo de 2 ints e 4 gaps >>
MPI_Type_commit(&filetype);                     << registra o novo tipo criado >>
disp = 5 * sizeof(int); etype=MPI_INT;
MPI_File_open(MPI_COMM_WORLD, "/pfs/arquivo",
              MPI_MODE_CREATE|MPI_MODE_RDWR, MPI_INFO_NULL, &fh);
MPI_File_set_view(fh, disp, etype, filetype, "native", MPI_INFO_NULL); << cria file-view >>
MPI_File_write(fh, buf, 1000, MPI_INT, MPI_STATUS_IGNORE);    << escreve 1000 ints >>
```

I/O Coletivo em MPI IO

- Funções de I/O coletivo:
  - Chamadas por todos os processos do comunicador
  - Objetivo: permitir otimizações pela biblioteca MPI
  - Idéia básica: agrupar I/O em blocos maiores
    - Bastante efetivo quando acessos se entrelaçam



Funções para I/O Coletivo

- Escrita: *MPI_File_write_at_all()*, etc
  - Argumentos similares aos das respectivas funções independentes, mas relativos a cada processo
  - *_all*: Indicaque todos os processos do comunicador usado ao abrir o arquivo vão chamar esta função
  - *_at*: Indica que a posição no arquivo é passada como parte da chamada; evita ter que chamar *seek*, e garante código mais claro, além de *thread-safety*

Exemplo de I/O Coletivo

- Exemplo: Acessar um array distribuído 2-D
  - Assume que 16 processos (P_0:P_{15}) vão acessar o arq.
  - Cada processador lê o arquivo e guarda os dados em sua memória local

P0	P1	P2	P3
P4	P5	P6	P7
P8	P9	P10	P11
P12	P13	P14	P15

- Dados armazenados no arquivo por linhas:

- Ex: Dados a serem lidos por P1:

Exemplo de I/O Coletivo (cont.)

- Leitura dos dados: 4 formas possíveis
  - Cada forma (ou *nível*) seleciona funções específicas de MPI, e as requisita do sist.arquivos
  - Diferentes formas (níveis) podem ter desempenhos distintos
  - Níveis são classificados de 0 a 3

Acesso em Nível 0

- Cada processo faz uma leitura independente a cada linha da sua parte no arquivo

```
MPI_File_open(..., file, ..., &fh);
for (i=0; i<n_local_rows; i++) {
        MPI_File_seek(fh,...);
        MPI_File_read(fh, &(A[i][0]), ...);
}
MPI_File_close(&fh);
```

Acesso em Nível 1

- Similar ao Nível-0, mas cada processo usa I/O coletivo para a leitura

```
MPI_File_open(..., file, ..., &fh);
for (i=0; i<n_local_rows; i++) {
        MPI_File_seek(fh,...);
        MPI_File_read_all(fh, &(A[i][0]), ...);
}
MPI_File_close(&fh);
```

Acesso em Nível 2

- Cada processo cria um tipo derivado que descreve seus acessos não-contíguos, define *file-view*, e chama uma função de I/O independente

*MPI_**Type_create_subarray**(..., &subarray, ...);*

MPI_Type_commit(&subarray); << registra novo tipo >>

MPI_File_open(..., file, ..., &fh);

*MPI_**File_set_view**(fh, ..., subarray, ...);*

*MPI_**File_read**(fh, A, ...);*

MPI_File_close(&fh);

Acesso em Nível 3

- Similar ao Nível-2, mas cada processo usa I/O coletivo para a leitura

MPI_Type_create_subarray(..., &subarray, ...);

MPI_Type_commit(&subarray); << registra novo tipo >>

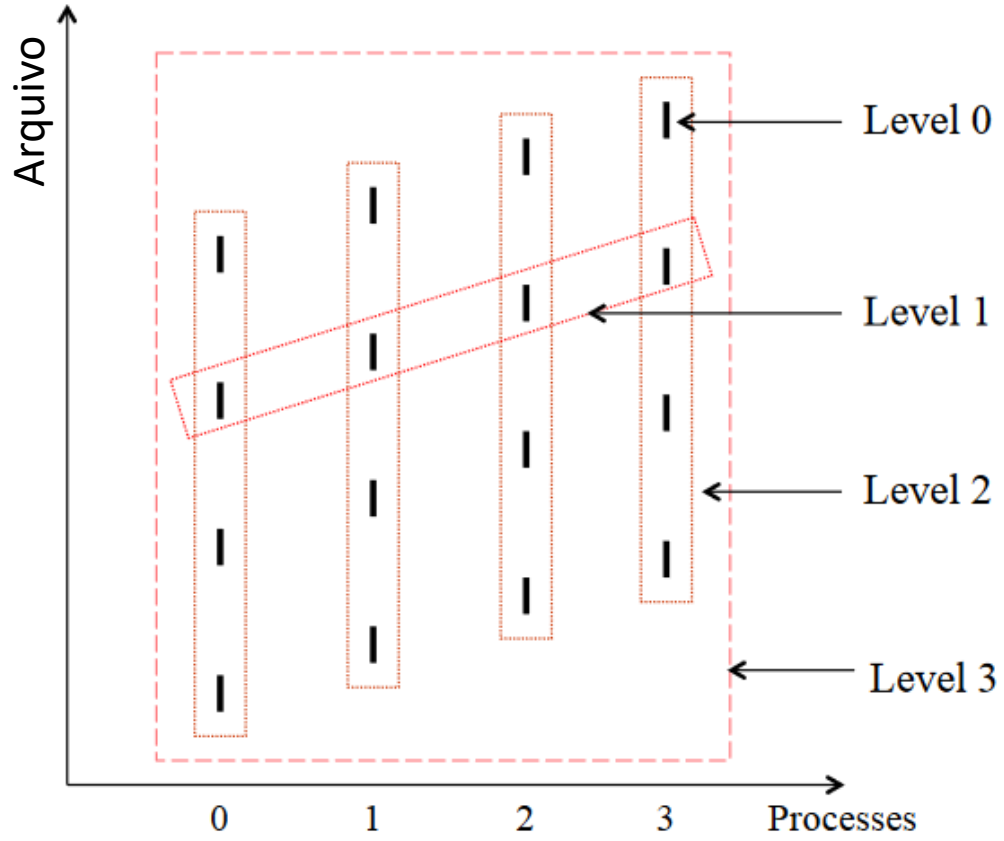
MPI_File_open(..., file, ..., &fh);

MPI_File_set_view(fh, ..., subarray, ...);

*MPI_File_read_**all**(fh, A, ...);*

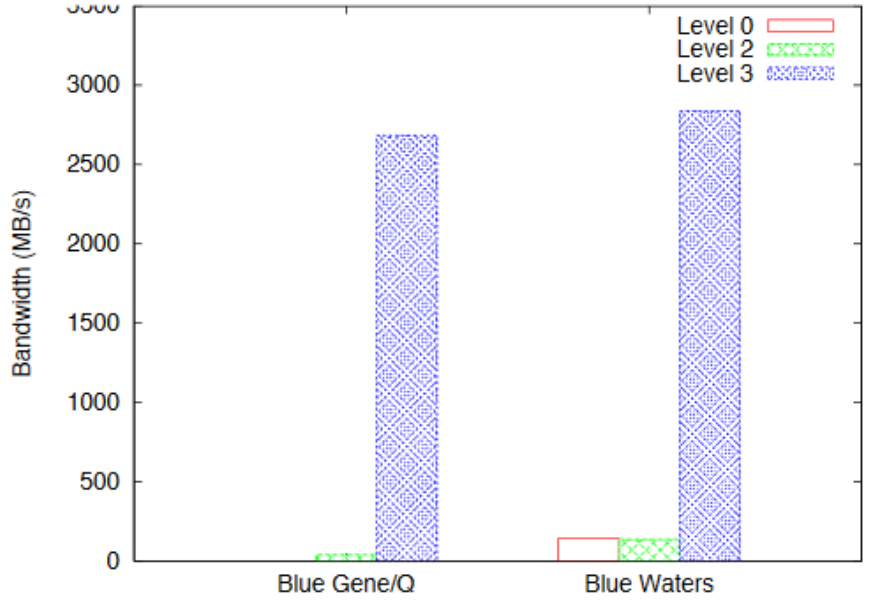
MPI_File_close(&fh);

Quatro Níveis de Acesso



- Níveis 0 e 1 pegam uma sub-linha por vez
  - Nível-1 otimiza usando coletiva
- Níveis 2 e 3 pegam todo o sub-array, usando novo tipo (dados não-contíguos)
  - Nível-3 otimiza usando coletiva
- Para cada aumento de nível, maior é o volume de dados lidos por chamada MPI
  - Aumenta a chance de otimização para a biblioteca MPI

Desempenho em Sistemas Reais



	Nível 0	Nível 2	Nível 3
BlueGene/Q	-	39	2.685
Blue Waters	142	137	2.839

- Teste: array de tamanho 1024^3
- Número de processos: 256
  - BG/Q: 32 nós, 8 procs/nó
  - BW: 16 nós, 16 procs/nó
- Sist.arquivos distintos
  - BG/Q: GPFS
  - BW: Lustre
- Resultados em MB/s
  - BG/Q: aumento de 68× !
  - BW: aumento de 20,7× !

MPI IO: Sumário

- Dispositivos de I/O têm latência alta
- Formas de obter melhor desempenho:
  - Otimizar acesso a dados não contíguos
    - Criação de tipos derivados em MPI
      - Grande vantagem de MPI IO sobre Posix I/O
    - Biblioteca MPI pode ler blocos maiores (contíguos) e extrair partes de interesse de cada processo
  - Combinar vários acessos em acessos maiores
    - Operações de I/O coletivo em MPI
    - Permite otimizações pela biblioteca MPI