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

Aula 30: Comunicação Unilateral - Extensões

Celso L. Mendes, Stephan Stephany

LAC / INPE

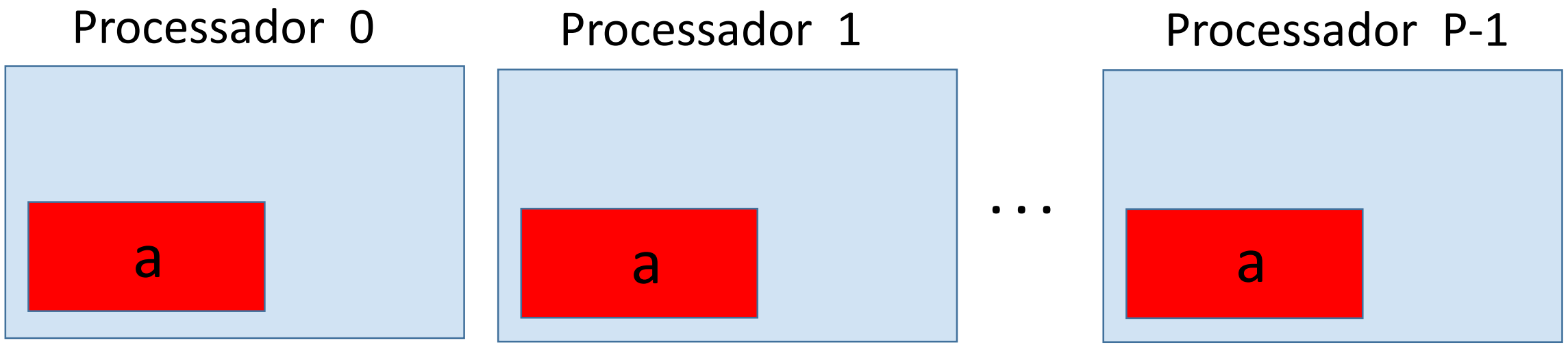
Emails: celso.mendes@inpe.br, stephan.stephany@inpe.br

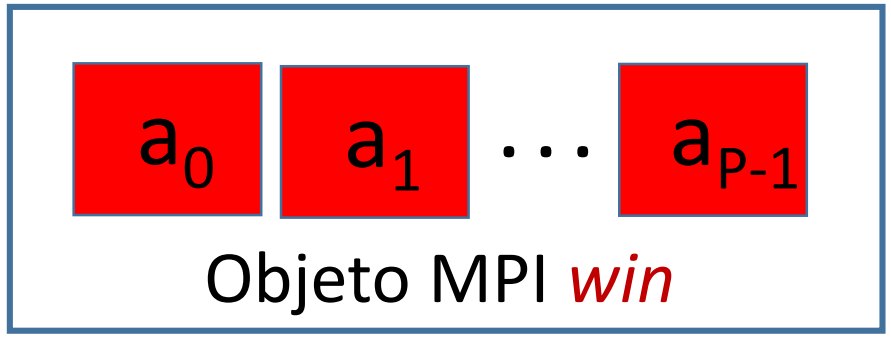
Modo Unilateral - Outro Exemplo

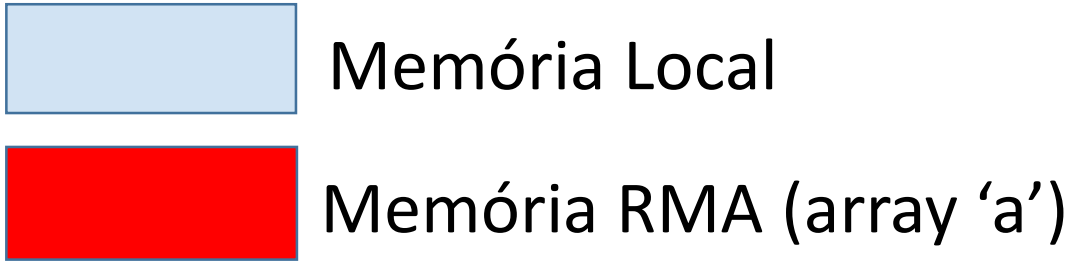
```
int main(int argc, char ** argv)
{
        int *a; MPI_Win win;
        MPI_Init(&argc, &argv);
        /* Aloca memória local */
        MPI_Alloc_mem(1000*sizeof(int), MPI_INFO_NULL, &a);
        /* Inicializa memória local */
        a[0] = 1; a[1] = 2;
        /* Expõe memória para uso com RMA */
        MPI_Win_create(a, 1000*sizeof(int), sizeof(int),
                MPI_INFO_NULL, MPI_COMM_WORLD, &win);
        /* Array 'a' passa a ser accessível a qualquer rank em MPI_COMM_WORLD */
        /* Qualquer processador pode fazer put/get com o array A remoto */
        < … >
        MPI_Win_free(&win);
        MPI_Free_mem(a);
        MPI_Finalize(); return 0;
}
```

→

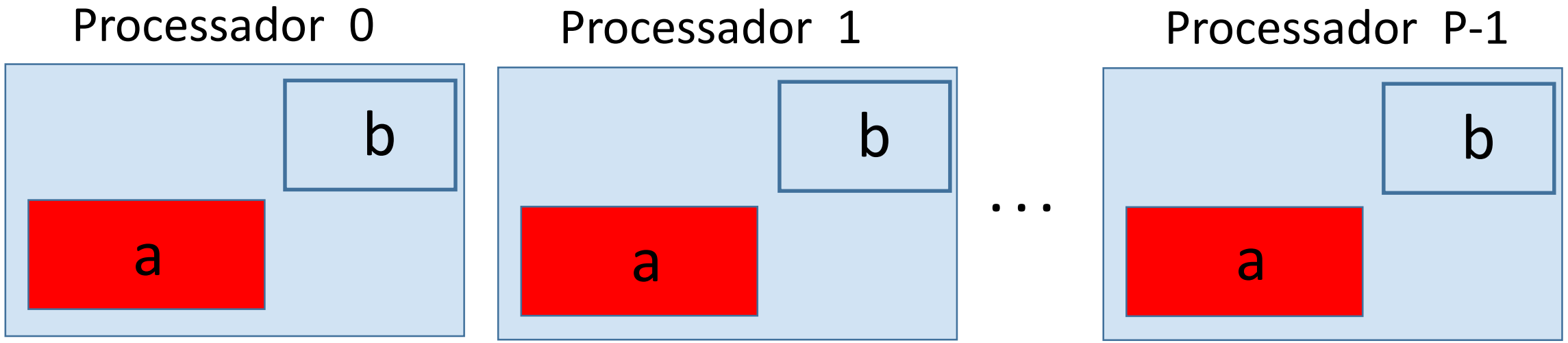
MPI_Win_create







Após MPI_Win_create



Operações locais de Proc.0:

a[5]=b[5];

b[6]=a[6];

Operações remotas de Proc.0:

MPI_Put($b_0[7] \rightarrow a_1[7]$);

MPI_Get($b_0[8] \leftarrow a_1[8]$);

Proc.0 **não** tem como acessar b_1, b_2, …, b_{P-1}

Criação de Janelas

- MPI_Win_create:
  - Expõe parte da memória local já existente/alocada
- Outras formas de criação possíveis (MPI-3):
  - MPI_Win_allocate
    - Aloca memória localmente e a expõe para RMA
  - MPI_Win_create_dynamic
    - Declara janela a ser usada com memória que será alocada
    - Após ser alocada localmente, memória é associada à janela
  - MPI_Win_create_shared
    - Permite compartilhamento entre processos num mesmo nó

Criação de Janela com Alocação

```
int main(int argc, char ** argv)
{
        int *a; MPI_Win win;
        MPI_Init(&argc, &argv);
        /* Cria memória RMA junto com a alocação */
        MPI_Win_allocate(1000*sizeof(int), sizeof(int),
                MPI_INFO_NULL, MPI_COMM_WORLD, &a, &win);
        /* Array 'a' passa a ser accessível a qualquer rank em MPI_COMM_WORLD */
        /* Qualquer processador pode fazer put/get com o array A remoto */
        < … >
        MPI_Win_free(&win);

        MPI_Finalize(); return 0;
}
```

Criação com Alocação Futura

```
int main(int argc, char ** argv)
{
        int *a; MPI_Win win;
        MPI_Init(&argc, &argv);
        /* Cria janela para RMA */
        MPI_Win_create_dynamic(MPI_INFO_NULL, MPI_COMM_WORLD, &win);
        /* Aloca memória localmente */
        a = (int *) malloc(1000 * sizeof(int));
        a[0]=1; a[1]=2;
        /* Declara memória como RMA */
        MPI_Win_attach(win, a, 1000*sizeof(int));
        /* Array 'a' passa a ser accessível a qualquer rank em MPI_COMM_WORLD */
        /* Qualquer processador pode fazer put/get com o array A remoto */
        < … >
        MPI_Win_detach(win,a); free (a);
        MPI_Win_free(&win);

        MPI_Finalize(); return 0;
}
```

Get & Accumulate

- Movimentação combinada com operação e busca

  *MPI_Get_accumulate(void *origin_addr, int origin_count, MPI_Datatype origin_type, void *result_addr, int result_count, MPI_Datatype result_type, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_type,* ***MPI_Op op****,MPI_Win win)*

  - Similar a *MPI_Accumulate*, também atômica
  - Combina dado de *origin* com o de *target* usando ***op***, e deixa o resultado combinado em *target*
  - Busca conteúdo inicial de *target* e o deixa em *result*

Fetch & Operate

- Movimentação combinada com operação e busca, aplicada a um único ítem

  *MPI_Fetch_and_op(void *origin_addr, void *result_addr, MPI_Datatype dtype, int target_rank, MPI_Aint target_disp, **MPI_Op op**, MPI_Win win)*

  - Versão simplificada de *MPI_Get_accumulate*, também atômica
  - Combina um dado de *origin* com um de *target* usando ***op***, e deixa o resultado combinado em *target*
  - Busca conteúdo inicial de *target* e o deixa em *result*
  - Tipo de dado *dtype* é o mesmo para todos os operandos
  - Permite otimizações de hw em algumas plataformas

Ordem de Operações em RMA

- Put/Get/Accumulate são sem bloqueio
- Ordem de término das operações
  - Nenhuma ordem garantida para *Puts/Gets* entre duas epochs
  - Resultados de *Puts/Gets* concorrentes à mesma janela são indefinidos
  - Resultados de *Accumulates* concorrentes à mesma janela são definidos de acordo com a ordem em que ocorrem
    - *Accumulates* do mesmo proc são ordenados

Exemplo: Comunicação de Bordas

```
subroutine exchng1(a, nx, s, e, win, &
                   bottom_nbr, top_nbr)
use mpi
integer nx, s, e, win, bottom_nbr, top_nbr
double precision a(0:nx+1,s-1:e+1)
integer ierr
integer(kind=MPI_ADDRESS_KIND) bottom_ghost_disp, top_ghost_disp

call MPI_WIN_FENCE(0, win, ierr)
! Put bottom edge into bottom neighbor's top ghost cells
! See text about top_ghost_disp
top_ghost_disp = 1 + (nx+2)*(e-s+2)
call MPI_PUT(a(1,s), nx, MPI_DOUBLE_PRECISION, &
             bottom_nbr, top_ghost_disp, nx, &
             MPI_DOUBLE_PRECISION, win, ierr)
! Put top edge into top neighbor's bottom ghost cells
bottom_ghost_disp = 1
call MPI_PUT(a(1,e), nx, MPI_DOUBLE_PRECISION, &
             top_nbr, bottom_ghost_disp, nx, &
             MPI_DOUBLE_PRECISION, win, ierr)
call MPI_WIN_FENCE(0, win, ierr)
return
end
```

Fonte: Gropp et al
Using Advanced MPI