1.

a)

```
int * bloco;
(...)
if(myid == 0) {
   (...)
   int buffer_size = (MPI_BSEND_OVERHEAD + n * sizeof(int));
   char * buffer = malloc(buffer_size);
   MPI_Bsend(
   MPI_Buffer_attach(&buffer, &buffer_size);
   MPI_Bsend(bloco, n, MPI_Int, 1, 0, MPI_COMM_WORLD);
   (...)
   MPI_Buffer_detach(&buffer, &buffer_size);
   free(buffer);
}
```

b)

O envio buffered bloqueante obriga a que o processo ~~espera~~ que envia os dados, espere até a estes estarem copiados para um buffer que apenas é utilizado para comunicar a informação que nele está contida, sem se efetuarem alterações nos dados. Após este buffer ter todos os dados que se pretendem comunicar, então o processo pode prosseguir e executar novas instruções, desta forma a informação está segura e não é corrompida.
Por outro lado, o envio buffered não bloqueante permite que o processo que envia os dados comece logo a executar novas instruções e ao mesmo tempo que os dados são copiados para o buffer, o que significa que pode dar-se o caso em que estes dados são alterados enquanto ainda ~~se está a~~ estão a ser copiados para o buffer, o que não devia acontecer.
Posto isto, com o buffered ~~[illegible]~~ bloqueante não é preciso ter cuidados extra, no entanto com o buffered não bloqueante é preciso ter cuidados extra. Para resolver ~~[illegible]~~ este problema, bastaria usar a função MPI_Wait juntamente com o parâmetro request.

2.

a) ~~MPI_Type_vector(50,~~

```
MPI_Type_vector(50, 100, 100, MPI_INT, &atype)
MPI_Type_commit(&atype)
MPI_Send(a[0][6], 1, atype, 1, 0, MPI_COMM_WORLD)
```

Rodrigo [illegible] Gonçalves

b)

```
MPI_Type_vector(50, 100, 200, MPI_INT, &btype)
MPI_Type_commit(&btype)
MPI_Send(a[0][0], 1, btype, 1, 1, MPI_COMM_WORLD)
```

c)

```
MPI_Type_vector(5000, 2, 2, MPI_INT, &ctype)
MPI_Type_commit(&ctype)
MPI_Send(a[0][1], 1, ctype, 1, 2, MPI_COMM_WORLD)
```

d)

```
MPI_Type_vector(100, 100, 100, MPI_INT, &dtype)
MPI_Type_commit(&dtype)
MPI_Send(a[0][0], 1, dtype, 1, 3, MPI_COMM_WORLD)
```

4.

a)

```
MPI_Comm_rank(Comm1, &newid)
MPI_Cart_shift(Comm1, 0, 1, &top, &bottom)
MPI_Cart_shift(Comm1, 1, 1, &left, &right)
```

b)

```
MPI_Send(top, 1, MPI_INT, left, MPI_ANY_TAG, comm1)
```

c)

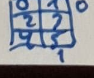
Ao inicializar o valor de identificação do vizinho no canto superior direito a MPI_PROC_NULL, está-se a garantir que antes de comunicação, todos os processos vêem o seu vizinho nesse canto como não existente. Assim, garante-se sempre que no limite esse vizinho é visto como não existente.

d)

Quando o comunicador não é periódico, essencialmente significa que não há condições fronteira periódicas relativamente aos processos, ou seja o processo mais em cima não tem como vizinho de cima o processo mais em baixo e vice-versa. Para além disto, o processo mais à direita não tem como vizinho à direita o processo mais à esquerda e vice-versa.

~~Caso o comunicador fosse periódico as respostas não seriam as mesmas, pois agora os vizinhos nas diagonais~~

Caso o comunicador fosse periódico, as respostas seriam iguais pois os vizinhos seriam achados corretamente. ~~No entanto a resposta à alínea b e c teria que ser diferente pois agora os vizinhos no canto superior direito já não seriam achados corretamente, pois podia dar-se o caso em que, por exemplo, o vizinho mais em cima continuasse~~

Exemplo: processos →  o processo 0 via como vizinho no canto superior direito como o 5.