Travaux Pratique 11

Ce sujet est en lien avec le cinquième chapitre du cours, et concerne la programmation répartie. Les mêmes commentaires que ceux des derniers TP s'appliquent ici aussi.

Dans ce sujet, vous poursuivez votre découverte enchantée de MPI en utilisant une topologie en **grille de $p \times p$ processeurs** au total, numérotés depuis 0. Ainsi, chaque processeur de numéro q peut envoyer et recevoir (mode bidirectionnel) un message aux processeurs de numéro $q-1$, $q+1$, $q-p$ et $q+p$ (modulo p).

Toute communication vers un processeur qui n'est pas sur la même colonne ou la même ligne est strictement interdite …

Votre travail est à rendre (il le sera à chaque fois) sous la forme du code (et uniquement la partie « student ») accompagnée d'un rapport au format PDF, le tout dans une archive compressée au format ZIP. Ne respecter pas ces contraintes, et votre note en sera diminuée de quelques points.

Vous ne devez modifier que ce qui se trouve dans le répertoire « `./student/` » !

Notez que chaque exercice vient avec un squelette, qui s'occupe de la ligne de commande, du lancement de votre code (défini dans une classe particulière), et d'une vérification sommaire du résultat (lorsque c'est possible). Utilisez l'option « `-h` » ou « `--help` » pour connaître le fonctionnement de la ligne de commande …

Exercice 1 : Découverte de la topologie en grille

Commencez ici par étudier la documentation de la fonction `MPI_Comm_split`.

Ensuite, étudiez le code fourni, afin de bien comprendre comment cette fonction est utilisée pour construire une topologie logique en grille, en "découpant" un groupe de $n \times n$ processeurs en lignes et colonnes. Ce court programme affiche, pour chaque processeur, ses deux rangs (un dans chaque groupe, donc l'un pour sa ligne et l'autre pour sa colonne).

Ce mécanisme permet ainsi de communiquer par ligne de processeurs et par colonne de processeurs.

Exercice 2 : Transposition de matrice

En utilisant une topologie logique en grille, vous allez maintenant écrire un algorithme qui utilise des diffusions sur les lignes et les colonnes. En pratique, cet algorithme calcule la transposée d'une matrice. N'essayez pas de faire une version parallèle au sein de chaque processus, mais d'écrire un algorithme fonctionnel.

Attention : suivant la solution vue en TD, vous ne devez utiliser que des `SEND` et des `RECV` sur vos anneaux formant la grille ...

La classe matrice est nouvelle : elle propose une distribution par bloc de cellules et non plus par bloc de lignes. En clair, au lieu d'avoir $m \times n$ éléments par processeur de la matrice, vous avez maintenant $b_h \times b_w$ éléments.

Voici un petit exemple de résultat attendu :

```
Windows PowerShell
PS D:\aveneau\Enseignement\Master\M1\APR\TP\TP11\correction> MPIEXEC -np 4 .\win32\Release\Transposition.exe -e 5
Build matrix A(32,32) ...
Build matrix B(32,32) ...
Call student function ...
Check the result ...
Well done, your software seems to be correct!
000 032 064 096 128 160 192 224 256 288 320 352 384 416 448 480 512 544 576 608 640 672 704 736 768 800 832 864 896 928 960 992
001 033 065 097 129 161 193 225 257 289 321 353 385 417 449 481 513 545 577 609 641 673 705 737 769 801 833 865 897 929 961 993
002 034 066 098 130 162 194 226 258 290 322 354 386 418 450 482 514 546 578 610 642 674 706 738 770 802 834 866 898 930 962 994
003 035 067 099 131 163 195 227 259 291 323 355 387 419 451 483 515 547 579 611 643 675 707 739 771 803 835 867 899 931 963 995
004 036 068 100 132 164 196 228 260 292 324 356 388 420 452 484 516 548 580 612 644 676 708 740 772 804 836 868 900 932 964 996
005 037 069 101 133 165 197 229 261 293 325 357 389 421 453 485 517 549 581 613 645 677 709 741 773 805 837 869 901 933 965 997
006 038 070 102 134 166 198 230 262 294 326 358 390 422 454 486 518 550 582 614 646 678 710 742 774 806 838 870 902 934 966 998
007 039 071 103 135 167 199 231 263 295 327 359 391 423 455 487 519 551 583 615 647 679 711 743 775 807 839 871 903 935 967 999
008 040 072 104 136 168 200 232 264 296 328 360 392 424 456 488 520 552 584 616 648 680 712 744 776 808 840 872 904 936 968 1000
009 041 073 105 137 169 201 233 265 297 329 361 393 425 457 489 521 553 585 617 649 681 713 745 777 809 841 873 905 937 969 1001
010 042 074 106 138 170 202 234 266 298 330 362 394 426 458 490 522 554 586 618 650 682 714 746 778 810 842 874 906 938 970 1002
011 043 075 107 139 171 203 235 267 299 331 363 395 427 459 491 523 555 587 619 651 683 715 747 779 811 843 875 907 939 971 1003
012 044 076 108 140 172 204 236 268 300 332 364 396 428 460 492 524 556 588 620 652 684 716 748 780 812 844 876 908 940 972 1004
013 045 077 109 141 173 205 237 269 301 333 365 397 429 461 493 525 557 589 621 653 685 717 749 781 813 845 877 909 941 973 1005
014 046 078 110 142 174 206 238 270 302 334 366 398 430 462 494 526 558 590 622 654 686 718 750 782 814 846 878 910 942 974 1006
015 047 079 111 143 175 207 239 271 303 335 367 399 431 463 495 527 559 591 623 655 687 719 751 783 815 847 879 911 943 975 1007
016 048 080 112 144 176 208 240 272 304 336 368 400 432 464 496 528 560 592 624 656 688 720 752 784 816 848 880 912 944 976 1008
017 049 081 113 145 177 209 241 273 305 337 369 401 433 465 497 529 561 593 625 657 689 721 753 785 817 849 881 913 945 977 1009
018 050 082 114 146 178 210 242 274 306 338 370 402 434 466 498 530 562 594 626 658 690 722 754 786 818 850 882 914 946 978 1010
019 051 083 115 147 179 211 243 275 307 339 371 403 435 467 499 531 563 595 627 659 691 723 755 787 819 851 883 915 947 979 1011
020 052 084 116 148 180 212 244 276 308 340 372 404 436 468 500 532 564 596 628 660 692 724 756 788 820 852 884 916 948 980 1012
021 053 085 117 149 181 213 245 277 309 341 373 405 437 469 501 533 565 597 629 661 693 725 757 789 821 853 885 917 949 981 1013
022 054 086 118 150 182 214 246 278 310 342 374 406 438 470 502 534 566 598 630 662 694 726 758 790 822 854 886 918 950 982 1014
023 055 087 119 151 183 215 247 279 311 343 375 407 439 471 503 535 567 599 631 663 695 727 759 791 823 855 887 919 951 983 1015
024 056 088 120 152 184 216 248 280 312 344 376 408 440 472 504 536 568 600 632 664 696 728 760 792 824 856 888 920 952 984 1016
025 057 089 121 153 185 217 249 281 313 345 377 409 441 473 505 537 569 601 633 665 697 729 761 793 825 857 889 921 953 985 1017
026 058 090 122 154 186 218 250 282 314 346 378 410 442 474 506 538 570 602 634 666 698 730 762 794 826 858 890 922 954 986 1018
027 059 091 123 155 187 219 251 283 315 347 379 411 443 475 507 539 571 603 635 667 699 731 763 795 827 859 891 923 955 987 1019
028 060 092 124 156 188 220 252 284 316 348 380 412 444 476 508 540 572 604 636 668 700 732 764 796 828 860 892 924 956 988 1020
029 061 093 125 157 189 221 253 285 317 349 381 413 445 477 509 541 573 605 637 669 701 733 765 797 829 861 893 925 957 989 1021
030 062 094 126 158 190 222 254 286 318 350 382 414 446 478 510 542 574 606 638 670 702 734 766 798 830 862 894 926 958 990 1022
031 063 095 127 159 191 223 255 287 319 351 383 415 447 479 511 543 575 607 639 671 703 735 767 799 831 863 895 927 959 991 1023
PS D:\aveneau\Enseignement\Master\M1\APR\TP\TP11\correction>
```

Exercice 3 : Produit de deux matrices : méthode de base

Dans cet exercice, vous devez implanter la première fonction « basique » vue en cours permettant de multiplier deux matrices. Testez différentes tailles de matrices et de grille, et notez bien les temps de calcul.

Exercice 4 : Méthode de Cannon

Dans cet antépénultième exercice, vous devez implanter le produit de deux matrices en utilisant l'algorithme de Cannon. Testez différentes tailles de matrices et de grille, et notez bien les temps de calcul.

Exercice 5 : Méthode de Fox

Dans ce pénultième exercice, vous devez implanter le produit de deux matrices en utilisant l'algorithme de Fox. Testez différentes tailles de matrices et de grille, et notez bien les temps de calcul.

Exercice 6 : Méthode de Snyder

Dans ce dernier exercice, vous devez implanter le produit de deux matrices en utilisant l'algorithme de Snyder. Testez différentes tailles de matrices et de grille, et notez bien les temps de calcul.