Computação em Larga Escala

MPI Problems 2 – Algorithmic analysis

António Rui Borges

Summary

- *Largest and smallest value of a number sequence*
  - *Characterization*
  - *Solution decomposition*
- *Matrix multiplication*
  - *Characterization*
  - *Solution decomposition*

Largest and smallest value of a number sequence

Let $\{seqVal_n\}$ be a sequence of numbers of length N, the *largest value of seqVal*, denoted *valMax*, is the element of *seqVal* that satisfies the property

$$\exists_{valMax \in seqVal} \quad \forall_{n \in \mathbb{N}_0} \; n < N \;\Rightarrow\; seqVal_n \leq valMax$$

and the *smallest value of seqVal*, denoted *valMin*, is the element of *seqVal* that has the property

$$\exists_{valMin \in seqVal} \quad \forall_{n \in \mathbb{N}_0} \; n < N \;\Rightarrow\; seqVal_n \geq seqMin \quad .$$

It takes $N - 1$ compares and possible saves to perform each of these operations.

Solution decomposition (LSV) - 1

A parallel solution to the problem requires at least two processes and may be divided in iterative steps consisting of the following operations

- normalization of the sequence length, N_{norm}

  it should be the smallest multiple of the number of processes, *Nproc*

$$\begin{aligned} N_{norm} &= N + NProc - N \bmod NProc && \Leftarrow N \bmod NProc \neq 0 \\ &= N && \Leftarrow N \bmod NProc = 0 \end{aligned}$$

- initialization of the [possible] new elements

  the new elemeents should be made equal to the smallest and the largest values that can be stored in the sequence elements for finding, respectively, the largest and the smallest value of the sequence so that they will not affect the final result

$$\begin{aligned} seqValMax_n &= MIN \\ seqValMin_n &= MAX \quad , \qquad \text{with } N \leq n < N_{norm} \end{aligned}$$

Solution decomposition (LSV) - 2

- sequence partition

  each process receives a piece of the sequence with equal length

$$seqValParcMax_n(k) = seqValMax_{NProc*k+n}$$
$$seqValParcMin_n(k) = seqValMin_{NProc*k+n} \quad ,$$
$$\text{with } \; 0 \le n < N_{norm} / NProc \quad \wedge \quad 0 \le k < NProc$$

- sequence reduction by element by element operation

  the sequence length is reduced by performing global compare and possible save operations on corresponding elements of the subsequences

$$seqValMax_n = seqValParcMax_n(0) \; CPS \cdots CPS \; seqValParcMax_n(NProc-1)$$
$$seqValMin_n = seqValParcMin_n(0) \; CPS \cdots CPS \; seqValParcMin_n(NProc-1) \quad ,$$
$$\text{with } \; 0 \le n < N_{norm} / NProc$$

Solution decomposition (LSV) - 3

Since the normalized sequence length is decreased at each iteration step

$$1 \le N_{norm}(i+1) < N_{norm}(i) \quad , \qquad \text{with } \; i \in \mathbb{N}_0 \quad ,$$

it will eventually be of size one and, then, one has

$$\begin{aligned} valMax &= seqValMax_0 \\ valMin &= seqValMin_0 \quad . \end{aligned}$$

Implementation (LSV) - 1

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>
#include <unistd.h>

#define  N       32
int main (int argc, char *argv[])
{
   int rank, nProc;
   int *seqValMin, *seqValMax, *seqValParcMin, *seqValParcMax;
   bool goOn;
   int i, n, nNorm;
   MPI_Init (&argc, &argv);
   MPI_Comm_rank (MPI_COMM_WORLD, &rank);
   MPI_Comm_size (MPI_COMM_WORLD, &nProc);
   if (nProc <= 1)
      { if (rank == 0) printf ("Wrong number of processes! It must be greater than 1.\n");
        MPI_Finalize ();
        return EXIT_FAILURE;
      }
   nNorm = N + (((N % nProc) == 0) ? 0 : nProc - (N % nProc));
   seqValMin = malloc (nNorm * sizeof (int));
   seqValMax = malloc (nNorm * sizeof (int));
   if (rank == 0)
      { srandom (getpid ());
        printf ("Generated sequence of values\n");
        for (i = 0; i < nNorm; i++)
          if (i < N)
             { seqValMin[i] = seqValMax[i] = ((double) rand () / RAND_MAX) * 1000;
               printf ("%d ", seqValMin[i]);
             }
             else { seqValMin[i] = INT_MAX;
                    seqValMax[i] = INT_MIN;
                  }

        printf ("\n");

      }
```

Implementation (LSV) - 2

```
    n = nNorm / nProc;
    seqValParcMin = malloc (n * sizeof (int));
    seqValParcMax = malloc (n * sizeof (int));
    goOn = true;
    while (goOn)
    { if (n == 1) goOn = false;
      MPI_Scatter (seqValMin, n, MPI_INT, seqValParcMin, n, MPI_INT, 0, MPI_COMM_WORLD);
      MPI_Reduce (seqValParcMin, seqValMin, n, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
      MPI_Scatter (seqValMax, n, MPI_INT, seqValParcMax, n, MPI_INT, 0, MPI_COMM_WORLD);
      MPI_Reduce (seqValParcMax, seqValMax, n, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
      nNorm = n + (((n % nProc) == 0) ? 0 : nProc - (n % nProc));
      for (i = 0; i < nNorm; i++)
        if (i >= n)
           { seqValMin[i] = INT_MAX;
             seqValMax[i] = INT_MIN;
           }
      n = nNorm / nProc;
    }
    if (rank == 0)
       { printf ("Smallest value = %d\n", seqValMin[0]);
         printf ("Largest value = %d\n", seqValMax[0]);
       }
    MPI_Finalize ();
    return EXIT_SUCCESS;
}
```

Implementation (LSV) - 3

```
[ruib@ruib-laptop exercises-2]$ mpiexec -n 1 ./largSmallValSeq
Wrong number of processes! It must be greater than 1.
[ruib@ruib-laptop exercises-2]$

[ruib@ruib-laptop exercises-2]$ mpiexec -n 2 ./largSmallValSeq
Generated sequence of values
159 519 413 352 843 83 265 546 192 94 346 946 660 819 638 880 741 307 748 843 17 397 32
782 144 749 611 64 33 491 966 193
Smallest value = 17
Largest value = 966
[ruib@ruib-laptop exercises-2]$

[ruib@ruib-laptop exercises-2]$ mpiexec -n 3 ./largSmallValSeq
Generated sequence of values
148 907 449 812 841 782 892 675 878 574 340 20 921 54 686 231 855 805 479 407 841 13 267
938 303 635 713 916 918 605 603 66
Smallest value = 13
Largest value = 938
[ruib@ruib-laptop exercises-2]$

[ruib@ruib-laptop exercises-2]$ mpiexec -n 4 ./largSmallValSeq
Generated sequence of values
195 871 546 20 39 516 532 509 77 543 680 195 973 876 403 826 224 567 253 587 390 175 384
948 857 933 79 280 682 119 690 877
Smallest value = 20
Largest value = 973
[ruib@ruib-laptop exercises-2]$

[ruib@ruib-laptop exercises-2]$ mpiexec -n 7 ./largSmallValSeq
Generated sequence of values
597 241 943 44 664 967 529 629 576 592 521 25 921 240 603 874 263 84 713 415 999 633 687
812 131 691 635 829 17 922 879 614
Smallest value = 17
Largest value = 999
[ruib@ruib-laptop exercises-2]$
```

Implementation (LSV) - 4

```
[ruib@ruib-laptop exercises-2]$ mpiexec -n 16 ./largSmallValSeq
Generated sequence of values
87 130 977 5 162 164 658 759 759 572 515 596 682 976 147 225 379 77 943 983 319 747 426
464 288 582 232 676 407 531 11 494
Smallest value = 5
Largest value = 983
[ruib@ruib-laptop exercises-2]$

[ruib@ruib-laptop exercises-2]$ mpiexec -n 31 ./largSmallValSeq
Generated sequence of values
923 655 84 810 26 498 599 563 981 460 398 76 736 51 316 113 148 154 677 463 721 923 546
818 981 264 633 508 120 597 166 43
Smallest value = 26
Largest value = 981
[ruib@ruib-laptop exercises-2]$

[ruib@ruib-laptop exercises-2]$ mpiexec -n 32 ./largSmallValSeq
Generated sequence of values
719 4 759 619 737 354 568 239 341 967 12 102 825 916 774 995 953 336 246 984 429 713 233
945 924 925 273 239 142 501 235 861
Smallest value = 4
Largest value = 995
[ruib@ruib-laptop exercises-2]$

[ruib@ruib-laptop exercises-2]$ mpiexec -n 37 ./largSmallValSeq
Generated sequence of values
86 591 833 382 748 864 113 798 607 219 652 104 108 433 695 742 799 297 878 986 839 394 22
170 30 16 524 25 432 328 614 518
Smallest value = 16
Largest value = 986
[ruib@ruib-laptop exercises-2]$
```

Matrix multiplication

Let A and B be square matrices of order N, one calls *multiplication of A by B*, and denotes $A \times B$, the square matrix C of order N, whose elements are given by

$$C = A \times B \Leftrightarrow c_{i\,j} = \sum_{k=0}^{N-1} a_{i\,k} \cdot b_{k\,j} \quad , \quad \text{with } 0 \leq i\,,\, j < N \quad .$$

It takes N^3 scalar multiplications and $N^2\,(N\text{-}1)$ scalar additions to carry out the operation.

Solution decomposition (MP) - 1

$$\begin{Vmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{Vmatrix} \times \begin{Vmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{Vmatrix} = \begin{Vmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{Vmatrix}$$

Solution decomposition (MP) - 2

A parallel solution to the problem may consist of the following operations

- <u>allowed number of processes</u>, *NProc*

  since the problem is computationally-intensive, the allowed number of processes should be a submultiple of *NProc*, different from 1, to make it efficient
- <u>the coefficients of matrix *B* are sent to all processes</u>

  so that all processes may compute all the coefficients of matrix C belonging to groups of adjacent lines
- <u>matrix *A* is partitioned in groups of adjacent lines</u>

  each process receives a group of lines
- <u>product computation</u>

  the computation at each process is carried out in parallel
- <u>matrix C composition</u>

  the different groups of lines of matrix *C* are stored together to get the matrix in the regular format.

Implementation (MP) - 1

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <unistd.h>

#define  N      8

int main (int argc, char *argv[])
{
   int rank, nProc;
   int *storeA = NULL, **matA, *storeB, **matB, *storeC = NULL, **matC = NULL;
   int *storeAParc, *storeCParc, **matAParc, **matCParc;
   int n, i, j, k;

   MPI_Init (&argc, &argv);
   MPI_Comm_rank (MPI_COMM_WORLD, &rank);
   MPI_Comm_size (MPI_COMM_WORLD, &nProc);
   if ((nProc <= 1) || ((N % nProc) != 0))
      { if (rank == 0)
           printf ("Wrong number of processes! It must be a submultiple of %d different from 1.\n", N);
        MPI_Finalize ();
        return EXIT_FAILURE;
      }
   n = N / nProc;
   storeB = malloc (N * N * sizeof (int));
   matB = malloc (N * sizeof (int *));
   storeAParc = malloc (n * N * sizeof (int));
   matAParc = malloc (n * sizeof (int *));
   storeCParc = malloc (n * N * sizeof (int));
   matCParc = malloc (n * sizeof (int *));
   for (i = 0; i < N; i++)
   { matB[i] = ((int *) storeB + i * N);
     if (i < n)
        { matAParc[i] = ((int *) storeAParc + i * N);
          matCParc[i] = ((int *) storeCParc + i * N);
        }
   }
```

Implementation (MP) - 2

```
if (rank == 0)
   { srandom (getpid ());
     storeA = malloc (N * N * sizeof (int));
     storeC = malloc (N * N * sizeof (int));
     matA = malloc (N * sizeof (int *));
     matC = malloc (N * sizeof (int *));
     for (i = 0; i < N; i++)
     { matA[i] = ((int *) storeA + i * N);
       matC[i] = ((int *) storeC + i * N);
     }
     printf ("Generated matrix A values\n");
     for (i = 0; i < N; i++)
     { for (j = 0; j < N; j++)
       { matA[i][j] = ((double) rand () / RAND_MAX - 0.5) * 10;
         printf ("%4d ", matA[i][j]);
       }
       printf ("\n");
     }
     printf ("Generated matrix B values\n");
     for (i = 0; i < N; i++)
     { for (j = 0; j < N; j++)
       { matB[i][j] = ((double) rand () / RAND_MAX - 0.5) * 10;
         printf ("%4d ", matB[i][j]);
       }
       printf ("\n");
     }
   }
```

Implementation (MP) - 3

```
    MPI_Bcast(storeB, N * N, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Scatter (storeA, n * N, MPI_INT, storeAParc, n * N, MPI_INT, 0, MPI_COMM_WORLD);
    for (i = 0; i < n; i++)
      for (j = 0; j < N; j++)
      { matCParc[i][j] = 0;
        for (k = 0; k < N; k++)
          matCParc[i][j] += matAParc[i][k] * matB[k][j];
      }
    MPI_Gather (storeCParc, n * N, MPI_INT, storeC, n * N, MPI_INT, 0, MPI_COMM_WORLD);
    if (rank == 0)
    { printf ("Matrix c values\n");
      for (i = 0; i < N; i++)
      { for (j = 0; j < N; j++)
          printf ("%4d ", matC[i][j]);
        printf ("\n");
      }
    }
    MPI_Finalize ();

    return EXIT_SUCCESS;
}
```

Implementation (MP) - 4

```
[ruib@ruib-laptop exercises-2]$ mpiexec -n 1 ./matProd
Wrong number of processes! It must be a submultiple of 8 different from 1.
[ruib@ruib-laptop exercises-2]$

[ruib@ruib-laptop exercises-2]$ mpiexec -n 3 ./matProd
Wrong number of processes! It must be a submultiple of 8 different from 1.
[ruib@ruib-laptop exercises-2]$

[ruib@ruib-laptop exercises-2]$ mpiexec -n 4 ./matProd
Generated matrix A values
  -4   -3    4   -4   -3   -3    1    2
   2    0    0    0    1    4   -3   -2
  -1   -4    4    4   -1    2    0    0
  -2    0   -2    1    0    0   -4    0
   1    4    0    3   -3   -3    0    4
   2   -3    0    0   -4    1    1    0
   2    0    0   -4   -1    3    0    1
   0    2   -2    3   -2   -1   -1    4
Generated matrix B values
  -2    3    2    0   -4   -1   -1    2
   0    3   -3    0    0    2    4   -2
  -1    0   -1    2   -2    3   -1    2
   0    1    0    3    4    4    3   -2
   2    0    1    3    4   -4    0   -1
   4    1    4    0    0    4    1    3
  -1   -4    0    1    3    3   -1    0
   0    4   -1    0    4   -3    1    2
Matrix c values
 -15  -24  -20  -12   -9   -9  -26   12
  17   14   23    0  -21    7    3   11
   4   -9   13   17    8   33   -5   13
  10   11   -2   -5    4  -12   11  -10
 -20   31  -29    0   12    7   25  -10
  -9   -6   13  -11  -21   15  -14   17
   6    9   14  -15  -24   -5  -10   24
  -5   28  -14   -2   21   -1   23   -7
[ruib@ruib-laptop exercises-2]$
```