Protokoll 2

Gruppe 2

December 20, 2022

Modul:	ETI22/23
Abschnitt:	Rechnerarchitektur

Namen:	Luis Renner
	Tianrui Zheng

1 Maximale Double Precision Leistung

Wie im letzten Praktikum berechnet wird, schafft ein Kern unseres Prozessors Intel(R) Xeon(R) E5-2680 v3:

Anzahl Kerne	= 12
AVX Frequency	= 2.1 GHz
Peak Performance	= 12 * 2.1 GHz * 16 $\frac{\text{FLOP}}{\text{Takt}}$
	= 403.2 GFLOPS

Die maximale Double Precision Leistung entspricht **Peak Performance** und beträgt also 403.2GFLOPS.

2 MPI Funktionen für Matrix Multiplikation

Zunächst wird die MPI-Umgebung durch:

MPI_Init(&argc, &argv)

initialisiert. Und wir bestimmen Rank und Size durch:

MPI_Comm_size(MPI_COMM_WORLD, &rank)
MPI_Comm_size(MPI_COMM_WORLD, &size)

Dann führen wir die Multiplikation zweier $n \times n$ Matrizen A und B durch. Wir unterteilen Matrix A(**a**) in verschiedene Chunks(**ap**) derselben Größe durch(**SUB_MPI_COUNT = n*(n/size)**):

MPI_Bcast(a, SUB_MAT_COUNT, MPI_FLOAT, ap, SUB_MAT_COUNT, MPI_FLOAT, 0, MPI_COMM_WORLD)

und teilen Matrix B(**b**) mit allen Prozessen durch(**MPI_COUNT = n*n**):

MPI_Bcast(b, MAT_COUNT, MPI_FLOAT, 0, MPI_COMM_WORLD)

Danach wird in jedem Prozess Multiplikation zwischen Chunk von A(**ap**) und der ganzen Matrix B(**b**) ausgeführt und das Teilergebnis speichern wir in einem Puffer(**cp**):

mat_mul(n/size, n, ap, b, cp);

Nachdem alle Prozesse jeweilige Berechnungen beendet haben, geben sie dem Master-Prozess(rank 0) das eigene Teilergebnis zurück:

MPI_Gather(cp, SUB_MAT_COUNT, MPI_FLOAT, c, SUB_MAT_COUNT, MPI_FLOAT, 0, MPI_COMM_WORLD)

Schließlich beenden wir MPI-Operationen durch:

MPI_Finalize()

3 MPI Versionen in "scs5" Modulumgebung

In Modulumgebung "scs5" sind folgende Versionen von OpenMPI zu finden:

```
------------------------------------------------------------------------
  openmpi:
------------------------------------------------------------------------
     Versions:
        openmpi/1.8.8-gnu
        openmpi/1.8.8-intel
        openmpi/1.10.2-pathscale6.0
        openmpi/1.10.2-pgi15.9
        openmpi/1.10.2-pgi16.1
        openmpi/1.10.2-pgi16.4
        openmpi/1.10.2-gnu-cuda
        openmpi/1.10.2-gnu
        openmpi/1.10.2-hpcx-gnu
        openmpi/1.10.2-intel
        openmpi/1.10.3-pgi16.5
        openmpi/1.10.3-intel-debug
        openmpi/1.10.4-pgi16.9
        openmpi/2.1.0-pgi17.4-cuda
        openmpi/2.1.0-pgi17.4
        openmpi/2.1.0-gnu
        openmpi/2.1.0-gnu6.3
        openmpi/2.1.0-intel
        openmpi/2.1.1-gnu-nodlopen
        openmpi/2.1.1-gnu5.3-cuda
        openmpi/3.0.0-pgi17.7-cuda
        openmpi/3.0.0-gnu5.5
        openmpi/3.0.0-gnu7.1
     Other possible modules matches:
        OpenMPI

------------------------------------------------------------------------
```

4 Verwendete MPI Version

Wir kompilieren unsere MPI-Anwendungen mit dem **mpicc** Befehl, der von Open-MPI mitgeliefert wird. Dieser Befehl ist eigentlich nur ein Wrapper für GCC. Die genutze GCC-Version kann mit dem Befehl:
mpicc –version
ermittelt werden (bei uns GCC 7.3.0) und die genutzte OpenMPI-Version kann mit:
mpicc –showme:version
ermittelt werden (bei uns OpenMPI 3.1.1).

5 Konfiguration

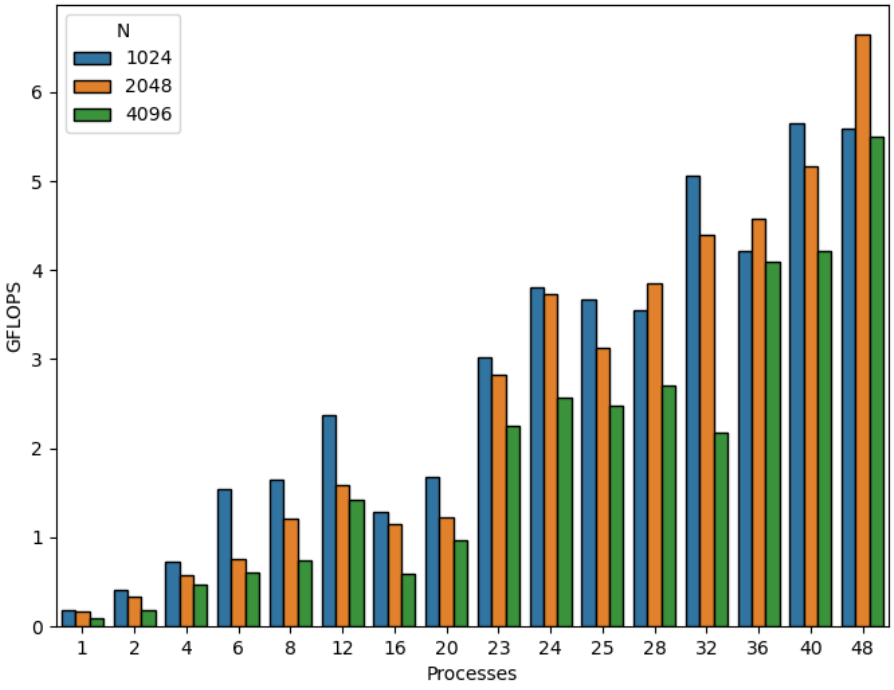
Um einen Job mit insgesamt 12 Prozessen gleichmäßig verteilt auf 2 Knoten zu starten, können wir folgende Konfiguration nutzen:

srun -n 12 -N 2 ... ./a.out

6 Ergebnisse

```
void mat_mpi(int n, int rank, int size, int argc, char **argv, int iteration)
{
    int r = n%size;
    while(r != 0)
    {
        n++;
        r = n%size;
    }
    const int N = n;
    const int M = N/size;
    const int MAT_COUNT = N*N;
    const int SUB_MAT_COUNT = M*N;
    const size_t MAT_SIZE = sizeof(float)*MAT_COUNT;
    const size_t SUB_MAT_SIZE = sizeof(float)*SUB_MAT_COUNT;

    float *a = NULL;
    float *b = malloc(MAT_SIZE);
    float *c = NULL;

    float *ap = malloc(SUB_MAT_SIZE);
    float *cp = malloc(SUB_MAT_SIZE);
    for(int i = 0; i < SUB_MAT_COUNT; i++) cp[i] = 0;

    clock_t t;

    if(rank==0)
    {
        srand(1);
        a = malloc(MAT_SIZE);
        c = malloc(MAT_SIZE);

        mat_fill(N, a);
        mat_fill(N, b);
        for(int i = 0; i < MAT_COUNT; i++) c[i] = 0;
        t = clock();
    }

    MPI_Scatter(a, SUB_MAT_COUNT, MPI_FLOAT, ap, SUB_MAT_COUNT, MPI_FLOAT, 0, MPI_COMM_WORLD);
    MPI_Bcast(b, MAT_COUNT, MPI_FLOAT, 0, MPI_COMM_WORLD);

    mat_mul(M, N, ap, b, cp);
    MPI_Gather(cp, SUB_MAT_COUNT, MPI_FLOAT, c, SUB_MAT_COUNT, MPI_FLOAT, 0, MPI_COMM_WORLD);

    if(rank == 0)
    {
        t = clock() - t;
        const double TIME_TAKEN = ((double)t)/CLOCKS_PER_SEC;
        printf("n:%d Time:%fs\n", N, TIME_TAKEN);

        free(c);
        free(a);
    }

    free(cp);
    free(ap);
    free(b);

    return;
}
int main(int argc, char **argv)
{
    int rank, size;
    MPI_Init(&argc,&argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Rank %d of %d\n", rank+1, size);
    MPI_Barrier(MPI_COMM_WORLD);

    int n[3] = {1024, 2048, 4096};
    for(int i = 0; i < 3; i++)
    {
        mat_mpi(n[i], rank, size, argc, argv, j);
    }

    MPI_Finalize();

    return 0;
}
```

Alle Werte wurden 10 mal gemessen und es ist jeweils der Median, das Minimum und das Maximum in $\frac{GFLOPS}{s}$ gegeben.
Gemessen wurde für Anzahl der Prozesse $\epsilon [1, 2, 4, 6, 8, 12, 16, 20, 23, 24, 25, 28, 32, 36, 40, 48]$.
Die Leistungen wurden mit $\frac{2*n^3}{Zeit}$ berechnet.



Nodes	Processes	N	GFLOPS	Max	Min
1	1	1024	0.1872	0.1953	0.1633
1	1	2048	0.1652	0.1684	0.1532
1	1	4096	0.0982	0.1091	0.0964
1	2	1024	0.4138	0.6010	0.3763
1	2	2048	0.3367	0.3852	0.2977
1	2	4096	0.1859	0.2165	0.1638
1	4	1024	0.7206	0.7650	0.6983
1	4	2048	0.5804	0.5931	0.5437
1	4	4096	0.4723	0.4882	0.4622
1	6	1024	1.5339	1.6434	1.3451
1	6	2048	0.7558	0.9052	0.7457
1	6	4096	0.6045	0.8382	0.5724
1	8	1024	1.6470	1.6935	1.5339
1	8	2048	1.2014	1.2896	1.1345
1	8	4096	0.7445	0.9014	0.6382
1	12	1024	2.3650	2.3890	2.3341
1	12	2048	1.5890	1.6086	1.5618
1	12	4096	1.4165	1.4341	1.4061
1	16	1024	1.2859	1.3772	1.2342
1	16	2048	1.1543	1.1791	1.1476
1	16	4096	0.5953	0.7773	0.4900
1	20	1024	1.6725	1.8046	1.3422
1	20	2048	1.2249	1.2315	1.2081
1	20	4096	0.9636	0.9659	0.9625
1	23	1024	3.0246	3.5791	2.0452
1	23	2048	2.8330	2.8538	2.7844
1	23	4096	2.2446	2.2542	2.2366
1	24	1024	3.8076	4.7722	2.3598
1	24	2048	3.7267	3.7592	3.6475
1	24	4096	2.5701	2.5767	2.5613
2	25	1024	3.6737	4.7439	2.3593
2	25	2048	3.1258	3.1523	3.0789
2	25	4096	2.4716	2.4741	2.4656
2	28	1024	3.5437	4.2949	2.2605
2	28	2048	3.8519	3.9134	3.6788
2	28	4096	2.6978	2.7087	2.6922
2	32	1024	5.0648	5.2378	4.6684
2	32	2048	4.4028	4.4973	4.2734
2	32	4096	2.1739	2.3965	2.1041
2	36	1024	4.2108	5.5067	2.5265
2	36	2048	4.5789	4.7722	4.4392
2	36	4096	4.0961	4.1686	4.0566
2	40	1024	5.6512	8.5899	2.9826
2	40	2048	5.1684	5.3024	4.8945
2	40	4096	74.2105	4.2367	4.1979
2	48	1024	5.5924	8.5899	2.9020
2	48	2048	6.6486	7.4695	5.3855
2	48	4096	5.4945	5.5374	5.4757