```
> restart:
```

Aufgabe 6

Parallele Programmierung mit dem Grid Programming Model

a) Schreiben Sie mit Hilfe des Grid Programming Models (Grid) eine Prozedur `myadd(f,i,n)` zur Addition der Zahlen `f(1) + f(2) + ... + f(n)`. Hierbei sei `f(i)` ein von `i` abhängiger Ausdruck.

Hinweise: Werten Sie die Terme `f(i)` mit evalf aus. Siehe auch `mySeq(f,i,n)` in Kapitel 8.5.1 des Skripts.

```
> myadd := proc(f, i::nonnegint, n::nonnegint, logs::boolean:=
  false)
    uses Grid:
    local final, k, res, thisNode, maxNodes, div, lower, upper, j:
    description "Proc for adding up values from the given function.
  This proc uses parallel computing.":

    thisNode:=MyNode():
    maxNodes:=NumNodes():
    div:=floor(n/maxNodes):
    lower:=thisNode * div + i:
    upper:=lower + div - i:

    if maxNodes mod 2 <> 0 then
      if thisNode = maxNodes - 1 then
        upper:=lower + div:
      end if:
    end if:

    if logs then printf("Started computing the node %d with lower=
  %d and upper=%d\n", thisNode, lower, upper) end if:

    res:=add(evalf(eval(f,x=j)), j=lower..upper):

    if logs then printf("Finished computing the node %d with lower=
  %d and upper=%d\n", thisNode, lower, upper) end if:

    if thisNode > 0 then
```

```
    if logs then printf("Sending the computed result %f from node
  %d to master node 0\n", res, thisNode) end if:

    Send(0, res):

   else

    if logs then printf("Started collecting the results from %d
  nodes\n", maxNodes) end if:

    res + add(Receive(k), k=1..maxNodes-1) + add(evalf(eval(f,x=
  j)),j=div * maxNodes+1 ..n);   # Siehe Aufruf unten für n=25.

   end if:
  end proc;
```

```
myadd := proc(f, i::nonnegint, n::nonnegint, logs::boolean := false)                    (1)
  local final, k, res, thisNode, maxNodes, div, lower, upper, j;
  description
  "Proc for adding up values from the given function. This proc uses parallel computing.";
  thisNode := Grid:-MyNode( );
  maxNodes := Grid:-NumNodes( );
  div := floor(n / maxNodes);
  lower := thisNode * div + i;
  upper := lower + div - i;
  if maxNodes mod 2 <> 0 then
     if thisNode = maxNodes - 1 then upper := lower + div end if
  end if;
  if logs then
     printf("Started computing the node %d with lower=%d and upper=%d\n", thisNode, lower,
     upper)
  end if;
  res := add(evalf(eval(f, x = j)), j = lower..upper);
  if logs then
     printf("Finished computing the node %d with lower=%d and upper=%d\n", thisNode,
     lower, upper)
```

```
    end if;
    if 0 < thisNode then
        if logs then
            printf("Sending the computed result %f from node %d to master node 0\n", res,
            thisNode)
        end if;
        Grid:-Send(0, res)
    else
        if logs then printf("Started collecting the results from %d nodes\n", maxNodes) end if;
        res + add(Grid:-Receive(k), k = 1 ..maxNodes - 1) + add(evalf(eval(f, x = j))), j = div
        * maxNodes + 1 ..n)
    end if
end proc
```

b) Berechnen Sie die Summe $\sum_{i=1}^{n} \frac{1}{i^2}$ mit $n = 10^7$ und messen Sie die benötigte Laufzeit von myadd auf 4 und 8 Knoten (Threads). Messen Sie zum Vergleich die Laufzeit, die die Maple-Funktion add benötigt.

```
> f := x -> 1/x**2;
```

$$f := x \mapsto \frac{1}{x^2} \tag{2}$$

```
> with(Grid):
> result_with_my_add := Launch(myadd, f(x), 1, 10^7, true,
  numnodes=4);
```

```
Started computing the node 3 with lower=7500001 and upper=10000000
Started computing the node 2 with lower=5000001 and upper=7500000
Started computing the node 0 with lower=1 and upper=2500000
Started computing the node 1 with lower=2500001 and upper=5000000
Finished computing the node 3 with lower=7500001 and upper=10000000
Sending the computed result 0.000000 from node 3 to master node 0
Finished computing the node 1 with lower=2500001 and upper=5000000
Sending the computed result 0.000000 from node 1 to master node 0
Finished computing the node 0 with lower=1 and upper=2500000
Started collecting the results from 4 nodes
Finished computing the node 2 with lower=5000001 and upper=7500000
Sending the computed result 0.000000 from node 2 to master node 0
```

$$result\_with\_my\_add := 1.644933967 \quad (3)$$

Zur Erklärung der fehlenden Summanden in Deiner Prozedur.

```
> korrigiertes_ergebnis_n25 :=Launch(myadd, f(x), 1, 25, true,
  numnodes=4);
  result_n25 :=add(evalf(f(x)), x = 1..25);
  Rest_der_fehlt := evalf(f(25));      # Rest der mit Deiner
  Prozedur ignoriert wird.
Started computing the node 2 with lower=13 and upper=18
Started computing the node 0 with lower=1 and upper=6
Finished computing the node 0 with lower=1 and upper=6
Started collecting the results from 4 nodes
Finished computing the node 2 with lower=13 and upper=18
Sending the computed result 0.025917 from node 2 to master node 0
Started computing the node 3 with lower=19 and upper=24
Started computing the node 1 with lower=7 and upper=12
Finished computing the node 1 with lower=7 and upper=12
Sending the computed result 0.073588 from node 1 to master node 0
Finished computing the node 3 with lower=19 and upper=24
Sending the computed result 0.013230 from node 3 to master node 0
```

$$korrigiertes\_ergebnis\_n25 := 1.605723404$$

$$result\_n25 := 1.605723404$$

$$Rest\_der\_fehlt := 0.001600000000 \quad (4)$$

```
> result_with_maple_add := add(evalf(f(x)), x = 1..10^7);
```

$$result\_with\_maple\_add := 1.644933967 \quad (5)$$

```
> difference := abs(result_with_my_add - result_with_maple_add);
```

$$difference := 0. \quad (6)$$

```
> time_with_myadd_4 := time(Launch(myadd, f(x), 1, 10^7, numnodes=
  4));
```

$$time\_with\_myadd\_4 := 0.218 \quad (7)$$

```
> time_with_myadd_4 := time[real](Launch(myadd, f(x), 1, 10^7,
  numnodes=4));  # Realzeit statt CPU-Zeit.
```

$$time\_with\_myadd\_4 := 7.200 \quad (8)$$

```
> time_with_myadd_8 := time[real](Launch(myadd, f(x), 1, 10^7,
  numnodes=8));
```

$$time\_with\_myadd\_8 := 6.509 \quad (9)$$

```
> # time_with_maple_add := time(add(f, i = 1..10^7));
  add(f, i = 1..10^7);                                  # Du
```

```
summierst 10^7 mal die Variable f.

time_with_maple_add := time(add(evalf(f(i)), i = 1..10^7));  #
ohne evalf stehen im Zähler und Nenner sehr große Integerwerte,
                                                              #
die nicht dargestellt werden können.
                                                              #
Dies führt zur Meldung [Length of output exceeds limit of
1000000].
```

$$10000000\, f$$

$$time\_with\_maple\_add := 26.234 \qquad \textbf{(10)}$$

c) Bestimmen Sie den exakten Grenzwert der Summe $\sum_{i=1}^{\infty} \frac{1}{i^2}$ mit der Prozedur sum.

Wieviel Dezimalstellen Genauigkeit hat die Partilasumme $\sum_{i=1}^{10^7} \frac{1}{i^2}$?

```
> limit_exact:=sum(f(x), x=1..infinity);
```

$$limit\_exact := \frac{\pi^2}{6} \qquad \textbf{(11)}$$

```
> evalf(limit_exact) = result_with_my_add;
  difference_limit_partial_sum:=abs(limit_exact -
  result_with_my_add);
```

$$1.644934068 = 1.644933967$$

$$difference\_limit\_partial\_sum := 1.00 \times 10^{-7} \qquad \textbf{(12)}$$

Partialsumme und exaktes Ergebnis stimmen in den ersten 5 Dezimalstellen bzw. in den ersten 6 signifikanten Stellen überein.