

## Aufgabe 1 - Das erste MPI-Programm

Bei ausführung des Programmes ist aufgefallen, dass für weniger als 5 Prozessen die Reihenfolge nicht immer stimmt. Trotz der nutzung von *MPI\_Barrier*, ergibt die Ausgabe bei weniger Prozessen teilweise folgendes (**Warum?**).

```
wege@west1:~$ mpirun -np=6 ~/Uebungen/6/timempi.x
[0] west1 // 2025-11-21 11:09:13.747368
[1] west1 // 2025-11-21 11:09:13.747367
[2] west1 // 2025-11-21 11:09:13.747367
[3] west1 // 2025-11-21 11:09:13.747367
[4] west1 // 2025-11-21 11:09:13.747367
[0] beendet jetzt!
[1] beendet jetzt!
[2] beendet jetzt!
[4] beendet jetzt!
[5] beendet jetzt!
[3] beendet jetzt!
wege@west1:~$ █
```

Figure 1: Ausgabe nach Ausführung

```
wege@west1:~$ mpirun -np=4 ~/Uebungen/6/timempi.x
[0] beendet jetzt!
[1] beendet jetzt!
[2] beendet jetzt!
[0] west1 // 2025-11-21 11:14:46.888693
[1] west1 // 2025-11-21 11:14:46.888691
[2] west1 // 2025-11-21 11:14:46.888690
[3] beendet jetzt!
wege@west1:~$ █
```

Figure 2: Fehlerhafte Ausgabe nach Ausführung

Der Sonderfall mit nur einem Prozess funktioniert soweit wie erwartet.

```
wege@west1:~$ mpirun -np=1 ~/Uebungen/6/timempi.x
[0] west1 // 2025-11-21 11:21:16.830836
[0] beendet jetzt!
wege@west1:~$ █
```

Figure 3: Ausgabe mit einem Prozess

## Aufgabe 2 - Ergebnisse sammeln im MPI-Programm

Hier tritt wieder das selbe Problem wie in 1) auf. Vermutung: Es liegt am im Hinweis beschriebenen buffern vom Terminal-Output.

```
weg@west1:~$ mpirun -np=6 ~/Uebungen/6/timempi.x
[0] west1 // 2025-11-21 11:26:35.286803
[1] west1 // 2025-11-21 11:26:35.286800
[2] west1 // 2025-11-21 11:26:35.286810
[3] west1 // 2025-11-21 11:26:35.286804
[4] west1 // 2025-11-21 11:26:35.286801
[5] Kleinstes MS-Anteil: 286800
[5] Größte Differenz:: 10
[0] beendet jetzt!
[1] beendet jetzt!
[2] beendet jetzt!
[3] beendet jetzt!
[4] beendet jetzt!
[5] beendet jetzt!
weg@west1:~$ █
```

Figure 4: Ausgabe mit einem Prozess

## Aufgabe 3 - Paralleles Debugging mit DDT

1. Die Programmparameter für DDT können im DDT-Fenster bei Ausführung von Run (queue submission mode) oder direkt in der Comandozeile beim Befehl `ddt ./timempi.x <argc, argv>` angegeben werden.
2. Auch in DDT gibt es die üblichen Step-Möglichkeiten *Step Into*, *Step Over* und *Step Out* (sowie Pause/Continue funktionen).  
*Step Into* gehe in die Funktion/Befehl hinein.  
*Step Over* gehe über die Funktion/Befehl zur nächsten Funktion/Befehl.  
*Step out* gehe aus der Funktion/Befehl oder Loop hinaus.
3. Die linien in der Darstellung zeigen an, wie die Prozesse zueinander stehen. So sind sie die meiste Zeit Parallel zueinander (übereinander) dargestellt wenn sie Parallel laufen. Ab dem Zeitpunkt wo nur noch ein Prozess arbeitet (schreibt) und die anderen wegen der Barriere auf ihn warten, wird auch nur noch der eine arbeitende Prozess dargestellt.
4. Im Evaluate-Fensters können Variablen angegeben werden, sodass zu demem Zeitpunkt der entsprechende Wert bzw. Zustand dargestellt wird. Anders als in *Current Line*, wo nur werte für die aktuelle Zeile oder *Locals* wo nur locale Werte betrachtet werden können.
5. Das Evaluate-Fenster ändert sich entsprechend des jeweils ausgewählten Prozesses, sodass die Werte der Variablen vom jeweiligen Prozess im Evaluate-Fenster angezeigt werden.
6. Arrays werden als eine Liste dargestellt, wobei diese Ausgeklappt werden kann um einzelne Werte anzusehen (ähnlich wie auch structures).

{[0] = 716574, [1] = 0}
716574
0
<optimized out>
<optimized out>
''
*[2] west1 // 2025-11-21 12:30
— 2
— 3
*2025-11-21 12:30:05*
{tv_sec = 1763724605, tv_usec = 716574}
1763724605
716574

Figure 5: Local Variables in DDT