Time left 0:26:27

Question **3**
Not yet answered
Marked out of 4.00
Flag question

Bewerten Sie folgende allgemeine Aussagen zu MPI (dem "Message-Passing Interface").

Richtig	Falsch	
○	✗	MPI schreibt vor, dass ausschließlich gemeinsamer Speicher ("Shared Memory") benutzt werden darf.
✗	●	MPI definitiert die Semantik (Bedeutung, Wirkung) von Kommunikationsoperationen.
○	✗	Der MPI-Standard schreibt vor, welche Algorithmen für kollektive Operationen verwendet werden müssen.
✗	○	MPI kann zusammen mit OpenMP verwendet werden.

Message-passing parallel programming model

- Finite set of "processes" with local memory (and program) that
- communicate explicitly by sending and receiving messages, all processes explicitly involved (exception: one-sided communication)
- No implicit synchronization between processes
- No other means of communication (shared nothing)

Message Passing Interface (**MPI**) ist ein Standard, der den Nachrichtenaustausch bei parallelen Berechnungen auf verteilten Computersystemen beschreibt. Er legt dabei eine Sammlung von Operationen und ihre Semantik, also eine Programmierschnittstelle fest, aber keine Implementierung.

MPI design principles, imperatives and goals

- High-performance: Communication functions close to typical "hardware" functionality, low protocol stack overhead
- Portability and Scalability
- Memory efficient: little dynamic memory (O(1)?) needed by MPI functions, memory (communication buffers) in user-space
- Coexist with other parallel interfaces (OpenMP, threads, …)
- Support (not hinder) construction of tools
- Support heterogeneous systems (data representation)
- Support SPMD or MIMD paradigm
- Support library building, application specific libraries

Question **5**
Not yet answered
Marked out of 4.00
Flag question

Time left 0:21:53

Bewerten Sie folgende Aussagen zu OpenMP.

Richtig	Falsch	
✗	●	Ein Programmfragment, das durch die OpenMP-Direktive `#pragma omp master` gekennzeichnet ist, wird von einem bestimmten Thread ausgeführt.
○	✗	Bei der Ausführung von einem OpenMP-Programm ist die Anzahl der Threads in den verschiedenen parallelen Regionen immer gleich.
✗	●	Auf einem System mit 8 Prozessor-Kernen können in einem `parallel` Block mehr als 8 Threads erzeugt werden.
✗	●	Anstatt der Direktive `#pragma omp barrier` kann auch der folgende Code `#pragma omp single {}` zur Synchronisierung der Threads benutzt werden.

#pragma omp master: Block wird immer nur vom Master Thread ausgeführt

Anzahl der verwendeten Threads wir während der Runtime pro Parallel-Block entschieden. & Anzhal kann on the fly umgestellt werden.

#pragma omp single hat am ende eine Implizite Barriere

Bewerten Sie die folgenden Aussagen zu Kommunikationsnetzwerken.

Richtig	Falsch	
✗	○	Der Durchmesser ("Diameter") eines Netzwerkes ist eine untere Schranke der Anzahl der benötigten Kommunikationsrunden, um Daten von einem Prozessor an alle anderen zu übermitteln ("Broadcast").
○	✗	In einem vollständig verbundenen Kommunikationsnetzwerk ("Fully Connected Network") mit nur einem Kommunikationskanal ("1-Ported") kann ein Prozessor Daten an alle anderen Prozessoren in $O(1)$ Kommunikationsrunden übermitteln. (2)
✗	○	Die Anzahl der Kommunikationsrunden der "Broadcast"-Operation ist in Gitternetzwerken ("Mesh" oder "Torus") mindestens so groß wie in einem vollständig verbundenen Netzwerk ("Fully-connected Network").
✗	○	In einem Mehrkern-Rechencluster ("Compute Cluster of Multicore Compute Nodes") muss ggf. die Kommunikation über das Netzwerk seriell stattfinden, wenn mehrere Prozesse auf einem Rechenknoten gleichzeitig Daten ins Netzwerk schicken möchten.

diameter(G): max(|shortest path(u,v)| over all u,v in V)

Lower bounds number of communication rounds for collective communication operations

Both: Diameter determines broadcast complexity

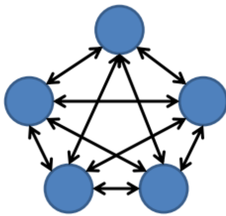
```
Fully-Connected Network hat
diameter = 1. Mesh/Torus hat
diameter ≥ 1.
Der Diameter bestimmt die
Broadcast Complexity
```

(3)

```
Help pls!!
```

G =(V,E) is the complete graph, each processor is directly connecte to each other processor

(2)



diameter = 1

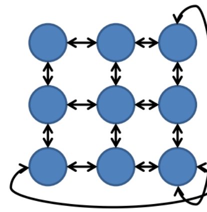
bisection width = $(p/2)^2$

Expensive: p^2-p links (cables, switch-ports, …), degree = p-1

1-ported communication system: In each communication operation/step, a processor can send/receive to/from at most one other processor

Mesh, torus (regular)

(3)



"wrap-around" for tori

diameter(mesh) = d (d√p-1)

diameter(torus) = d floor(d√p/2)

d'th root

Mesh/Torus: Diameter determines broadcast complexity

Bewerten Sie die folgenden Aussagen zum Programmieren mit Threads auf Rechnern mit gemeinsamem Speicher ("Shared Memory System").

Richtig	Falsch	
●	✗	"False Sharing" kann mit atomischen Operationen immer umgangen werden.
✗	○	Wenn Aktualisierungen auf benachbarten Einträgen eines Feldes ("Array") in einem Programm vorkommen, kann "False Sharing" vermieden werden, indem die Größe der Cache-Zeile ("Cache Line") explizit als Abstand der Einträge im Feld benutzt wird.
○	✗	"False Sharing" tritt bei der Ausführung von OpenMP-Programmen nicht auf, weil OpenMP die Cache-Zugriffe serialisiert. (3)
✗	○	In einem direkt abgebildeten Cache ("Directly Mapped Cache") wird jede Adresse im Hauptspeicher nur auf eine bestimmte Cache-Zeile ("Cache Line") abgebildet. (4)

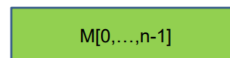
Bitte beantworten Sie alle Teile der Frage.

```
Prozessor hat nur Elemente mit denen er
arbeit in seinem Cache-Block (definiert durch
Cache-Line). Andere Prozesse müssen sich auf die
Elemente des anderen Prozesses zugriefen
(wie zuvor definiert durch die Cache-Linie)
```

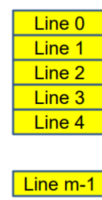
```
#pragma parallel for schedule(static,1)
for (i=0; i<m; i++) {
  y[i] = 0;
  for (j=0;j<n; j++) {
    y[i] += X[i][j]*A[j];
  }
}
```

(3) Chunks consisting of a single iteration.

Typical false sharing case



(4)



Cache: m«n, block size s

Directly mapped:

M[i] cached in line (i/s) mod m

M[0],…,M[s-1] go to line 0, M[s],…,M[2s-1] to line 1, … , M[m],…,M[m+s-1] again to line 0, …

Normally $m=2^r$ for some r>0, $s=2^{r'}$ (powers of two: mod and div translate into mask and shift)

.VA in TISS

board / Kurse / E180 - Fakultät für Informatik / E191 - Institut für Computer Engineering / 184.710-2022S / Übungen und Quizzes

Verbleibende Zeit 0:12:52

e 2
ort
eichert
ichbare
kte: 4,00
rage
ieren

Test
1
10
Versu

Gegeben ist ein MPI Kommunikator mit 5 Prozessen

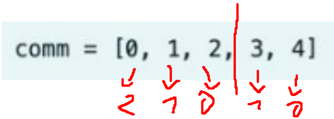
```
comm = [0, 1, 2, 3, 4]
```

Es wird nun folgendes MPI-Programmfragment von den 5 Prozessen in `comm` ausgeführt:

```
int rank, size;
MPI_Comm_rank(comm,&rank);
MPI_Comm_size(comm,&size);
color = ...;
key = ...;
MPI_Comm_split(comm,color,key,&nucomm);
```

Bewerten Sie folgende Aussagen zu dem neuen Kommunikator `nucomm`.

Richtig	Falsch	
	X	Kein `nucomm`-Kommunikator kann weiter durch `MPI_Comm_split(...)` partitioniert werden.
X		Wenn in jedem Prozess `color=rank` und `key=0` gesetzt wird, ist jeder Prozess in einem `nucomm` mit nur einem Prozess (dem Prozess selber).
X		Wenn der "Split" mit `color=(rank > size/2) ? 0 : 1` und `key=-rank` ausgeführt wird, hat der Prozess mit `rank=3` im Ursprungskommunikator danach im neuen Kommunikator `nucomm` den lokalen Rang 1.
	X	Wenn von allen Prozessen `key=0` gesetzt wird, weist MPI jedem Prozess einen zufälligen Rang im neuen Kommunikator `nucomm` zu.



Nein, da mit key=0 das alte ranking beibehalten wird

herige Seite

Nächste Seite

Bewerten Sie folgende Aussagen zu parallelen Programmen.

Richtig	Falsch	
X		Die Isoeffizienz-Funktion von einem Algorithmus sagt aus, wie die Problemgröße (Eingabegröße) in Abhängigkeit der Anzahl der Prozessoren gewählt werden muss, um eine bestimmte konstante Effizienz beizubehalten.
	X	Die Arbeit eines parallelen Programmes P wird berechnet, indem die Anzahl der ausgeführten Operationen in P durch die Anzahl der Prozessoren geteilt wird.
X		Die inklusive Präfix-Summe kann in $O(n/p + \log p)$ Zeitschritten berechnet werden und ist somit arbeitsoptimal für bis zu $\frac{n}{\log n}$ Prozessoren.
X		Der Speedup ist eine untere Schranke für die parallele Effizienz eines Programmes.

Bitte beantworten Sie alle Teile der Frage.

Definition:
A parallel algorithm/implementation is weakly scaling if there is a slowly growing function f(p), such that for n = Ω(f(p)), E(p,n) remains constant. The function f is called the iso-efficiency function

Theorem:
The (inclusive/exclusive) prefix-sums problem for an array of n elements with an associative binary operator "+" can be solved on an EREW PRAM with p processors in O(n/p+log p) time steps.

Definition:
The efficiency of parallel algorithm Par is the ratio of best possible parallel time to actual parallel time for given p and n:

$E(p,n) = (Tseq(n)/p) / Tpar(p,n)$
$= S_p(n)/p = Tseq(n) / (p\ Tpar(p,n))$

Three theoretical solutions to the parallel prefix-sums probler

1. Recursive: Fast, work-optimal
2. Iterative: Fast, work-optimal
3. Doubling: Fast(er), not work-optimal (but still useful)

Questions:
- How fast can these algorithms really solve the prefix-sum problem?
- How many operations do they require (work)?

Frage 1

Bisher nicht beantwortet

Erreichbare Punkte: 4,00

Frage markieren

Bewerten Sie folgende allgemeine Aussagen zu kollektiven Operationen in MPI.

`MPI_Comm_dup`:
Collective operation, MUST be called by all processes in `comm`

Richtig	Falsch	
X	○	Alle kollektiven Operationen in MPI müssen von allen Prozessen in demselben Kommunikator ausgeführt werden, d.h. ruft ein Prozess eine kollektive Operation mit einem Kommunikator auf, so müssen alle anderen Prozesse in diesem Kommunikator die gleiche Operation aufrufen und keine andere kollektive Operation.
○	X	Alle kollektiven Operationen haben für gleich große Eingabe-Puffer ("Buffer") die gleiche asymptotische Laufzeit. (2)
○	X	Wenn jeder Prozess im gleichen Kommunikator (`comm`) den eigenen Rank als root-Argument in einem `MPI_Bcast(...,root,comm)`-Aufruf angibt, werden alle Prozesse ihre Daten gleichzeitig an alle anderen Prozesse übermitteln (eine sogenannte "Multi-Broadcast"-Operation). (3)
○	X	`MPI_Bcast(...,root,comm)` kann statt `MPI_Barrier(comm)` zur Synchronisierung des Programmablaufs genutzt werden. (4)

`MPI_Bcast hat keinen impliziten barrier`

Complexity of collective operations (general)

(2)

MPI Collective	Complexity
MPI_Barrier	O(d)
MPI_Bcast	O(n+d)
MPI_Gather/Scatter	O(n+d)
MPI_Allgather	O(n+d)
MPI_Alltoall	O(n+p) (*)
MPI_Reduce	O(n+d)
MPI_Allreduce	O(n+d)
MPI_Scan/Exscan	O(n+d)

Important to find algorithms with small constants: See HPC lecture

(*) interesting tradeoffs possible

- p MPI processes (on p processors), n is total amount of data per process
- d = max(log p, diameter of network), determines latency

Requirement:

Collective operations must be called with consistent arguments: same root, same op, exactly matching amounts of data (see individual functions)

Re: Does mpi_bcast have an implied barrier?
von Träff Jesper Larsson - Mittwoch, 14. Juni 2023, 13:47

(4)

Dear Mr. Bayatmoghadam,

no collectives - except of course MPI_Barrier - have an implied barrier, at least you should think of them as not implying any form of barrier synchronization. These very important semantics points (blocking/non-blocking, local/non-local completion) were discussed at lot during the lectures on MPI

```
double a = 0.0;
double b = 0.0;

if (rank==0) {

  a = 5.0;
  MPI_Send(&a,1,MPI_DOUBLE,2,tag,MPI_COMM_WORLD);

} else if (rank==1) {

  MPI_Request req;
  MPI_Irecv(&b,1,MPI_DOUBLE,2,tag,MPI_COMM_WORLD,&req);
  MPI_Wait(&req,MPI_STATUS_IGNORE);

} else if (rank==2) {

  MPI_Sendrecv(&b,1,MPI_DOUBLE,1,tag,
               &a,1,MPI_DOUBLE,0,tag,
               MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  b = a;

}
```

Time left 0:06

Bewerten Sie die folgenden Aussagen.

Richtig	Falsch	
○	●	Das Programm kann in eine "Deadlock"-Situation kommen.

Nein, MPI_Sendrecv sendet b = 0 an Rank 1. b = 0, da der Behfel b=a erst nach dem senden von Rank 2 erfolgt

Richtig	Falsch	
●	✗	Das Programm überträgt den Wert der Variable a vom Prozess mit Rang 0 in die Variable b vom Prozess mit Rang 1, d.h. nach jeder Ausführung gilt b `== 5.0` für Prozess 1.
○	✗	Unmittelbar nach der `MPI_Irecv`- und vor der `MPI_Wait`-Operation von dem Prozess mit Rang 1 gilt `b==5.0`. Nein, da MPI_Irecv nicht blockiert
○	✗	Das Programm kann in eine "Deadlock"-Situation kommen. Nein, da logisch kein Kreis vorhanden ist
✗	○	Nach der `MPI_Wait`-Operation gilt für Prozess mit Rang 1: `b==0.0`.

MPI_Irecv(recvbuf,…,rank,tag,comm,request):

Initiates receive operation from specific rank, or ANY
- Operation is non-blocking: Call returns immediately, message received after completion

If flag==1 operation has completed, information in status

```
MPI_Wait(&request,&status);
```

Wait: Returns when operation has completed, information in status

Question 6

Not yet answered

Marked out of 4.00

Flag question

Time left 0:02:41

Folgendes Programm wird mit 6 MPI-Prozessen ausgeführt.

```
int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);

int A[2] = {rank, size};
int B[2] = {-1, -1};

MPI_Exscan(A,B,2,MPI_INT,MPI_SUM,MPI_COMM_WORLD);
```

Bewerten Sie folgende Aussagen zu den Ergebnissen der verschiedenen Prozesse.

Richtig	Falsch	
○	☒	Wird `MPI_Exscan(...)` durch `MPI_Allreduce(...)` und `MPI_SUM` durch `MPI_MAX` ersetzt, gilt für alle Prozesse `B[0]==6`.
☒	○	Wird `MPI_Exscan(...)` durch `MPI_Scan(...)` ersetzt, gilt für Prozess 4 (`rank==4`) `B[1]==30`.
○	☒	Für Prozess 0 (`rank==0`) gilt immer `B[1]==0`.
☒	○	Für Prozess 5 (`rank==5`) gilt immer `B[0]==10`.

Der Maximale Rank (B[0]) ist 5

				MPI_ExScan	
p	A	B		A	B
0	{0,6}	{-1,-1}		{0,6}	undefined
1	{1,6}	{-1,-1}		{1,6}	{0,6}
2	{2,6}	{-1,-1}		{2,6}	{1,12}
3	{3,6}	{-1,-1}		{3,6}	{3,18}
4	{4,6}	{-1,-1}		{4,6}	{6,24}
5	{5,6}	{-1,-1}		{5,6}	{10,30}

Bewerten Sie folgende Aussagen zu Send- und Receive-Operationen in MPI.

Richtig **Falsch**

- `MPI_Waitall()` ist eine kollektive Operation und muss von allen Prozessen im Kommunikator ausgeführt werden.

- Der Puffer in einem `MPI_Send(...)` kann nach dem Aufruf wieder benutzt werden, auch wenn die entsprechende Recv-Operation noch nicht aufgerufen wurde.

- Im folgenden Programmfragment kommt es bei großen Nachrichten zu einem Deadlock, da beide Prozesse gleichzeitig versuchen zu senden und empfangen:

```
int rank;
MPI_Comm_rank(MPI_COLMM_WORLD,&rank);

int n = ...;


float *a = (float*)malloc(n*sizeof(float));
float *b = (float*)malloc(n*sizeof(float));

if (rank==0) {
  MPI_Sendrecv(a,n,MPI_FLOAT,1,1111,
               b,n,MPI_FLOAT,1,2222,
               MPI_COMM_WORLD,
               MPI_STATUS_IGNORE);
} else if (rank==1) {
  MPI_Sendrecv(b,n,MPI_FLOAT,0,2222,
               a,n,MPI_FLOAT,0,1111,
               MPI_COMM_WORLD,
               MPI_STATUS_IGNORE);
}
```

Safe(r) programming

Process 0	Process 1
`MPI_Send`	`MPI_Send`
`MPI_Recv`	`MPI_Recv`

Unsafe. Can be made safe safe by combined send-receive

Process 0		Process 1
`MPI_Sendrecv`		`MPI_Sendrecv`

- Wird `MPI_Send(...,comm)` von einem Prozess ausgeführt, wartet der Prozess bis sein Kommunikationpartner die Nachricht empfangen hat.