Informatics Parallel Computing

Research Group for Parallel Computing
Faculty of Informatics
TU Wien

Parallel Computing

2023S
Assignment 1

Issue date: 2023-03-22
Due date: 2023-03-29 (23:59)

The exercises can be done in groups of at most three. The exercises are optional, and will be graded if handed in, but will not count towards the final grade for the lecture. Solutions should be short and concise, and with brief explanations. Please also give the calculations leading to your results.

Hand-in is via TUWEL only. Advance group registration in TUWEL is needed (regardless of group size, $1, 2, 3$). Preferably, use the LaTeX-template for the solution, and mark hand-in clearly with name and matriculation number for all group members. Handwritten solutions will neither be accepted nor graded. Only one hand-in per group is needed, the last submitted solution for a group is graded.

There will be no extension to the deadline.

Important: For all O-calculations, you can assume that constants have been normalized to one.

Exercise 1 (3 [3x1] points)

The dot (inner) product of two n-element vectors `x` and `y` can be computed as

$$\texttt{pr} = \sum_{i=0}^{n-1} \texttt{dot}[i] = \sum_{i=0}^{n-1} \texttt{x}[i] \cdot \texttt{y}[i] \quad .$$

The following piece of PRAM pseudo-code is part of a program for computing the final dot product `pr` of the two vectors `x` and `y`.

```
par (0<=i<n) {
  dot[i] = x[i] * y[i];
}
```

1. What is the number of time steps taken by the algorithm?
2. What is the asymptotic number of operations?
3. Which PRAM model is required for the program to execute correctly?

Exercise 2 (4 [4x1] points)

In order to complete the dot product computation of the previous exercise, you need to compute the sum of the element products pr as $\texttt{pr} = \sum_{i=0}^{n-1} \texttt{dot}[i]$. The task is to do this fast, in $O(\log n)$ time steps with $O(n)$ work. This can easily be accomplished for instance by modifying the maximum finding algorithm from the lecture:

```
int nn = n;
while (nn>1) {
  k = (nn>>1) + (nn&0x1); // ceil(nn/2)
  par (0<=i<k) {
    if (i+k<nn) a[i] = max(a[i],a[i+k]);
  }
  nn = k;
}
// a[0] now contains maximum
```

1. Modify the maximum finding algorithm above (taken from the lecture) to compute the sum pr.
2. Which PRAM model is needed for your algorithm/program to be executed correctly? Why this model?
3. Does your algorithm exploit commutativity of the addition operator $+$ used in the sum computation?
4. Give a reduction algorithm that works for arbitrary associative, but possibly non-commutative binary operations $\odot$.

Exercise 3 (3 [3x1] points)

A program works on square matrices of order n and performs a large number of matrix-matrix multiplications one after the other. The number of such iterations is some (possibly large) constant K. Each iteration takes at most cn^3 time steps for some medium to large constant c (that is $O(n^3)$), and can be perfectly parallelized (up to some maximum number $p, p < n^3$ of processors). Before the iterations can start, a sequential preprocessing of the input matrices is necessary and takes at most $3cn^3$ operations for the same constant c.

1. What is the maximum theoretical speed-up that this program can achieve as a function of c, K, n (allow n to grow with p, since there is no benefit from using more than cn^3 processors)?
2. Calculate the concrete speed-up with $c = 100, K = 100, n = 1000$ for $p = 10$ and $p = 1000$ processors, respectively.
3. What is the maximum speed-up with $c = 100, K = 1000, n = 10000$?

Exercise 4 (2 [1+1] points)

Two problems have best sequential algorithms running in $O(n \log n)$ and $O(n^2)$, respectively. Both algorithms can be parallelized, but have a constant sequential, non-parallelizable fraction. With a multi-core system with 64 processor-cores, we want to achieve a speed-up of 50.

1. How large can the sequential (non-parallelizable) fraction of the work be in the two cases?
2. Are there differences between the two cases?

Exercise 5 (4 [2+2] points)

A simple, parallel, work-optimal matrix-vector multiplication algorithm is running in $O(n^2/p + n)$ time steps on input of size $n^2 + n$ (matrix and vector), whereas sequential matrix-vector multiplication can be done (optimally) in $O(n^2)$ time steps.

1. Give a PRAM algorithm for matrix-vector multiplication (for $p = n$) that runs with the claimed bound. Which PRAM model is needed?
2. How large must n be in order to achieve a speed-up of 75 on 100 processor-cores?

Exercise 6 (6 [1+1+2+2] points)

We have three different parallel algorithms for matrix-vector multiplication at our disposal, running in time $O(n^2/p+\log n)$, $O(n^2/p+\sqrt{n})$, and $O(n^2/p+n)$, respectively. The best sequential algorithm is running in $O(n^2)$ time.

1. For each of the three cases, state $T\infty(n)$ and state the parallelism.
2. For each of the three cases, assuming the asymptotic constants have been normalized to 1, state the maximum number of processors for which a linear speedup can be achieved.
3. State the iso-efficiency functions for all three cases, that is input size n as a function of p at least required to achieve a given, fixed efficiency e. It may not be possible to give a closed-form formula in each case; if not state that: n must be at least...
4. Compute a required (integer!) n for efficiencies $e = 0.5$ and $e = 0.95$ for $p = 10,100,1000$ for the parallel algorithm with running time $O(n^2/p + n)$.