EXAM: Parallel Programming

May 26, 2015, 14:00 – 17:00

The duration of this exam is 3 hours. The number of (itemized) questions is 6 with a total of 20 items. Behind each item the number of points to be earned is depicted between square brackets (total number of points is 100). This exam is "**open book**", which means you can use your own notes and textbook. Internet browsing is allowed, but not recommended (permission should be explicitly asked for). Otherwise the use of electronic devices is not allowed. Give arguments with each answer: an answer without arguments will not be graded [0].

Question 1 [15]

1. Explain the differences/similarities between the programming models of SIMD, MIMD, Vector Processing and SPMD. How do the different programming models relate to NUMA and UMA, i.e. how do they operate on NUMA and UMA architectures. [3]
2. Explain why a single crossbar switch is too expensive for building a multiprocessor network. Compare the costs of a single crossbar switch implementation with a multistage switching network implementation. [3]
3. What are the advantages/disadvantages of a multistage interconnection network compared to a mesh interconnection network. [3]
4. In general mesh connected parallel computers are the network of choice for parallel computers. Explain from an application point of view why this is the case. [3]
5. For some applications, specific multistage networks have been build. Give two examples of such networks. [3]

Question 2 [15]

1. Clearly explain why tasks interaction graphs represent input- and output-dependencies and task dependencies graphs represent flow- and anti-dependencies. [3]
2. Explain how the task interaction graph and the task dependency graph each play their own role in mapping task to processes (processors). [4]
3. Consider the parallel computation of all pairs shortest path problem. Describe (with the help of pseudo code) two parallel programs to compute this problem: one program should be based on input data decomposition and the other one should be based on output data decomposition. [8]

Question 3 [15]

1. A major problem in optimizing the performance of parallel programs is caused by the unpredictability of runtime task size. This leads in many cases to load unbalance reducing the potential parallel speedup. Give two explicit examples of such cases. [3]
2. Most unpredictable runtime task sizes are caused by **conditional** control flow dependencies for which the evaluation of the conditions is input data dependent. Explain this, by giving an example. [3]
3. Describe how these load unbalances can be prevented in a parallel program. [9]

Question 4 [15]

1. Describe how dense matrix times dense matrix multiplication can be solved through BLAS3 routines. Describe how this BLAS3 routine consists out of BLAS2 routines and each of these BLAS2 routines consists out of BLAS1 routines. [3]
2. So essentially a BLAS3 routine consists out of BLAS1 routines. Give the main reason why specific implementations were developed for BLAS3 routines next to BLAS1/BLAS2 routines. [4]
3. Describe in pseudo code the implementation of a parallel **Sparse** BLAS3 routine for Sparse Matrix times Sparse Matrix multiplication on a distributed memory computer. [8]

Question 5 [20]

Tearing techniques are used for sparse matrix computations to split the matrix into different components each of which can be operated on in parallel. In tearing techniques based on nested dissection the sparse matrix is considered an adjacency matrix of a digraph G=(V, E). For this graph the nodes are split into sets B, C, and separator node set S, such that V = B ∪ S ∪ C, and such that there no edges from nodes of B to nodes of C and vise versa. Set S is minimized by moving nodes u from S which have only incoming edges from nodes of B to set C and nodes u from S, which have only outgoing edge to nodes of C are moved to set B. By doing so all edges introduced between B and C go into one direction from B to C.

1. Describe how these Separator sets S can be even reduced further by moving nodes directly from B to C and from C to B. [13]
2. Describe (in detail) how this technique is used to parallelize LU factorization of sparse linear systems with complete pivoting. [7]

Question 6 [20]

On the next page a parallel implementation is given for the MaxFlow problem expressed in **tUPL**.

1. Explain why this is a parallel implementation. [2]
2. Explain why **tUPL** profit chains can only consists between (conditional) serial codes, which have a data dependence. [3]
3. Give all possible data dependencies between the (conditional) serial codes of the **tUPL** MaxFlow implementation. [3]
4. Describe how "Augmented Paths" from the Ford Fulkerson algorithm are automatically generated by **tUPL**. [12]

SEE NEXT PAGE

MaxFlow expressed in tUPL

```
T={<u,v,w>|  (u,v)and(v,w)edges of G (including
             back edges) and w != u }

whilelem ( t; t ε T )
  { if(Delta[t.u,t.v]>0&&Remainder[t.v,t.w] > 0)
    {
       delta_change=
          min(Remainder[t.v,t.w],Delta[t.u,t.v]);
          Delta[t.v,t.w]+= delta_change;
          Remainder[t.v,t.w] -= delta_change;
          Remainder[t.w,t.v] += delta_change;
          F[t.u,t.v] += delta_change;
          Delta[t.u,t.v] -= delta_change
    }
    if (Delta[t.u,t.v]>0&&Remainder[t.v,t.w]== 0)
    {  if (t.v == ‘s’ || t.v == ‘t’)
       {
          F[t.u,t.v] += Delta[t.u,t.v];
          Delta[t.u,t.v] = 0
       }
       else
       {# Reverse Flow
          Delta[t.v,t.u] += Delta[t.u,t.v];
          Remainder[t.v,t.u]-= Delta[t.u,t.v];
          Delta[t.u,t.v] = 0
       }
    }
  }
```