MASTER OF SCIENCE
IN ENGINEERING

Parallel Computing and Algorithms

Prof. Dr. C. Stamm christoph.stamm@fhnw.ch Tel.: 056 202 78 32

Exercise 2

1 Amdahl's Law

Assume a problem of size *W*, which has a non-parallelizable (serial) component of size $W_{seq} = f \cdot W$, where *f* is a fraction between 0 and 1.

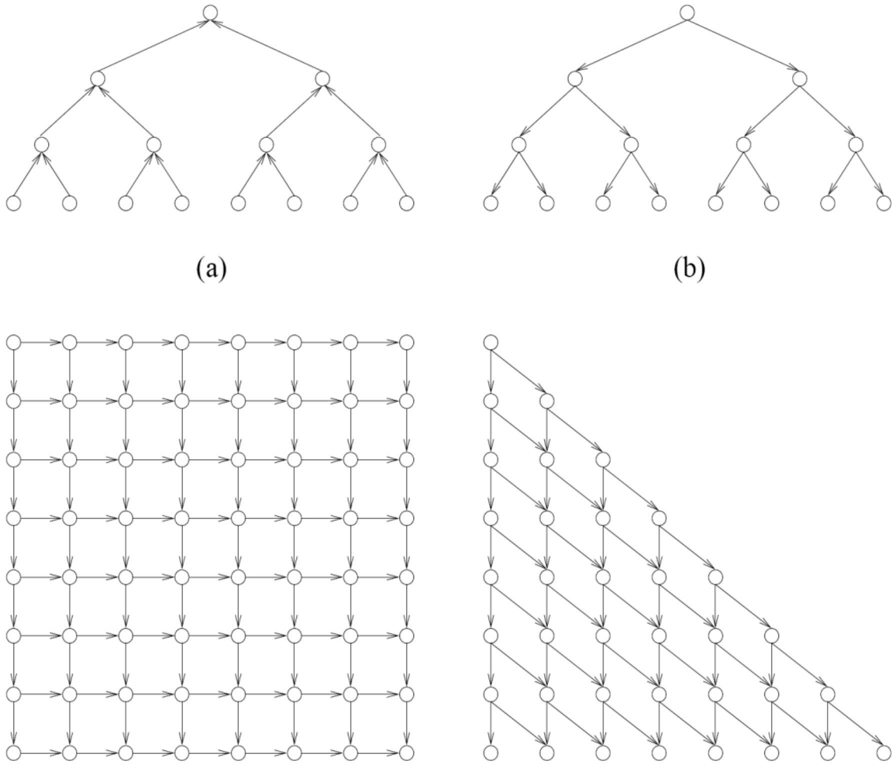
a) Express the speedup *S* in terms of *f* and *p*, where *p* is the number of processing elements.

b) Prove that W/W_{seq} is an upper bound on its speedup, no matter how many processing elements are used.

2 The DAG model of parallel computation

Parallel algorithms can often be represented by dependency graphs (see Figure): nodes represent tasks and directed edges represent the order in which the tasks must be performed. A node of the graph can be scheduled for execution as soon as the tasks at all the nodes that have incoming edges to that node have finished execution. Any deadlock-free dependency graph must be a directed, acyclic graph (DAG).

Let *N* be the number of nodes in each dependency graph.



(a)

(b)

(c)

(d)

Compute for all four dependency graphs:

a) an integer *n*, that represents the height of the tree or the mesh, respectively (measured in nodes);
b) the degree of concurrency (expressed in *n*);
c) the maximum possible speedup if an unlimited number of processing elements is available;
d) the values of speedup, efficiency, and the overhead if the number of processing elements is equal to the half of the degree of concurrency.

3 Adding Numbers

Let *A* be the efficient algorithm for adding *n* numbers on *p* processing elements introduced during the lectures.

a) Assume that it takes 10 time units to communicate a number between two processing elements, and that it takes 1 unit of time to add two numbers. Express the parallel runtime of *A* in *n* and *p*.

b) Express the standard speedup in *n* and *p* and plot the speedup curve for the problem size n = 5'000 and $p \in 2^i$, $0 \leq i \leq 10$.

Scaled speedup is defined as the speedup obtained when the problem size is increased linearly with the number of processing elements; that is, if *W* is chosen as a base problem size for a single processing element, then the $Scaled\ speedup = \frac{T_S(pW)}{T_P(pW,p)}$.

c) Now plot the scaled speedup curve for the same set of values of *p* and compare it with the standard speedup curve.

4 Image Processing on GPU

In Exercise 1 you optimized and parallelized an image processing routine for edge detection. Now, the goal is to implement the same routine for a GPU and to see what speedup is possible.

Before you start, please make sure that OpenCL version 1.1 or higher is supported on your GPU, and the necessary SDKs are installed. NVIDIA, AMD, and Intel support OpenCL on their GPUs. Depending on the GPU manufacturer you need to download and install a specific SDK:

• NVIDIA	CUDA Toolkit	http://developer.nvidia.com/cuda-downloads http://developer.nvidia.com/opencl
• Intel	SDK for OpenCL	http://software.intel.com/en-us/vcsource/tools/opencl-sdk

The OpenCL specification and manuals are developed by the Khronos group:

http://www.khronos.org/opencl/

Following OpenCL related source code files are given:

cl.hpp	OpenCL 1.2 C++ wrapper
main.h	header file imported in ocl.cpp
main.cpp	main routine and tests
ocl.cpp	OpenCL host code
edges.cl	OpenCL device code

a) Study the given source code carefully. Make sure that you can build and run it on your computer.

b) Complete the edge detection procedure `edges(…)` in OpenCL C-syntax in "edges.cl" and check the output. Why does it not produce the identical result as the parallel CPU implementation? What are the differences? What is the speedup compared to the parallel CPU implementation?

c) There are different technologies available to program GPUs and all of them produce different GPU code. Hence, it is helpful to use another technology for the same problem and to compare the results, the performance, and the implementation cost.
Windows and Visual Studio users should try to use C++ AMP (Accelerated Massive Parallelism). AMP is fully integrated in Visual Studio. A template file "amp.cpp" is already included in the given source code.
Linux users should try to use OpenACC. The OpenACC API describes a collection of compiler directives (similar to OpenMP) to specify loops and regions of code in standard C/C++ to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators. More information about this approach is on the following web site. Please note, that OpenACC compilers are only free for thirty days: http://www.openacc-standard.org. A template file "acc.cpp" is already included in the given source code.