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

Exercise 5

1 LU Decomposition

Consider the LU decomposition of a dense matrix shown on Slide 6.27.

a) Draw a task-dependency graph corresponding to the decomposition into 14 tasks.
b) Show an efficient mapping of the task-dependency graph onto three processors.
c) Show an efficient mapping of the task-dependency graph onto four processors.
d) If each task takes a unit amount of time, which of the two mappings (onto 3 or 4 processors) solves the problem faster.

2 Cannon's Matrix Multiplication Algorithm

"Cannon.cpp" contains an implementation of Cannon's matrix multiplication algorithm based on blocking MPI communication operations. "mainMPI.cpp" contains a test environment like the one given in task 2. The test environment also includes matrix data initialization and matrix block partitioning.

a) Study carefully the method `cannonBlocking(…)` and test the implementation of Cannon's algorithm. Make proposals where non-blocking communication operations could replace blocking operations.
b) Implement your ideas of using non-blocking communication operations in `cannonNonBlocking(…)`.

3 Matrix Multiplication Competition

This exercise is a small competition: Who develops the fastest parallel matrix multiplication implementation?

Given are the following files:

- "matrixmult.cpp" contains an already implemented caching aware sequential matrix multiplication algorithm and a template for your own parallel CPU implementation. The given implementation must not be modified, because it is the base for the speedup measurements;
- "matrixmult.cl" contains a template for your own GPU kernel implementation;
- "ocl.cpp" contains an already implemented initialization method `initOCL(…)`, which setups OpenCL and isn't part of the performance measurement, and a method `matMultGPU(…)`, which should do the data exchange between CPU and GPU and starts the GPU kernel;
- "main.cpp" contains the test environment, where you can see how the average speedup and efficiency over different matrix sizes is computed both for your CPU and OpenCL implementations.

There will be two separate competitions, one for CPUs only, and another one for GPUs. The rules are:

- Implement in `matMultCPU(…)` a correct parallel matrix multiplication algorithm using CPU cores only. Try to maximize the speedup with respect to the given sequential matrix multiplication implementation. The efficiency of this parallel CPU implementation will be finally ranked in the CPU competition.
- Implement in `matMultGPU(…)` a correct parallel matrix multiplication algorithm using a GPU. Try to maximize the speedup with respect to the given sequential matrix multiplication implementation. The efficiency of this parallel GPU implementation will be finally ranked in the GPU competition.