UNIVERSITÉ DU LUXEMBOURG

Project 1: Parallel Not-Equal

Lattice Theory for Parallel Programming

Goals

- ✭ Understanding a new model of computation based on lattice theory.
- ✭ Implementing two algorithms in this model on GPU.
- ✭ Send the code and report in ”name.surname.parneq.zip” before 28th of November 2024 (23h59) by email to `yi-nung.tsao@uni.lu` with the subject ”[MHPC][LTPP] Parneq Project”.
- ✭ This project accounts for 10% of the total grade.
- ✭ Don’t share your code.
- ✭ This is a solo project.

Exercise 1 – Warm-up: Parallel Minimum

In C++, implement an algorithm to find the minimum in an array:

- Implement a sequential algorithm first.
- In CUDA/C++, implement a version using a global minimum variable (decreasing integer lattice) and the fixpoint model of computation.
- In CUDA/C++, implement a parallel algorithm on GPU.
- Use a massive number of threads (eg. 1024 blocks of 1024 threads) to compare the efficiency of both approaches.
- Use the HPC of the University if you don’t have a CUDA-compatible GPU in your computer (or want more GPU power!).

Exercise 2 – Parallel Combinatorial Optimization

We are going to solve a simple combinatorial problem consisting of n variables $X = \{x_1, \ldots, x_n\}$, K upper bounds $u_1, \ldots, u_k$ and a list $C \subseteq \mathcal{P}(\mathbb{N} \times \mathbb{N})$. We must count the number of assignments $asn : X \to \mathbb{Z}$ such that:

- $\forall i \leq n,\ 0 \leq asn(x_i) \leq u_i$ (domain constraints).
- $\forall (i,j) \in C,\ asn(x_i) \neq asn(x_j)$.

To solve this problem, a very simple algorithm is to rely on a backtracking procedure where you enumerate all the values of the variables and test if they match the constraints. We provide a backtracking algorithm for reference (`nqueens.cpp`), but this version is for a slightly different problem (N-Queens problem) and only look for the first solution, not all. You can read about N-Queens on Wikipedia (`https://en.wikipedia.org/wiki/Eight_queens_puzzle`), it can be useful to create testing instances for your problem.

1. Modify `nqueens.cpp` to count all solutions.
2. Modify `nqueens.cpp` to check an arbitrary list of inequalities.
3. We can improve this algorithm by removing impossible values from the variables during backtracking:
   - Initially, we represent the *domain* of each variable x_i by a set $\{0, \dots, u_i\}$.
   - For a constraint $x_i \neq x_j$, if we have $x_i = \{2\}$, we can remove 2 from the set of x_j, because if $x_j = 2$ it will violate the constraint.
   - We perform this operation until we reach a fixpoint.
   - You can represent the set of value using an array $\texttt{dom}_i$ of Booleans for each variable such that $\texttt{dom}_i\texttt{[v] = true}$ if $v \in x_i$ and false otherwise.
4. Now that we have this sequential algorithm, we are going to parallelize it on GPU using CUDA/C++.
   The main parallelization task will be the removal of impossible values in parallel.
   Using the parallel lattice programming model seen in class, you should be able to implement it without locks or atomics, using a fixpoint loop.
5. Design and code optimizations, while maintaining correctness. Measure the efficiency on various check board size, possibly for various variants and optimizations of your algorithm, and write a short report to explain the core of your algorithm.
6. In the report, add the proof of correctness that the parallel algorithm is producing the same result than the sequential version. You must essentially prove that your algorithm is within the framework of PCCP.