We implemented characteristics of a cluster of GPUs in Nemerle using NUDA [4], a set of high-level language extensions for GPU programming. Internally, NUDA uses OpenCL for GPU code generation. MPI .NET is used across cluster nodes.

**Main problem** is divergence of control flow and memory accesses across neighbouring search stacks. This results in only 15% efficiency. However, the code scales perfectly across 108 of GPUs, so we concentrate on improving single-GPU efficiency.

**Optimization 1**: Concentrate fixed bits in middle search rounds. By concentrating fixed bits in middle search rounds, we decrease divergence of search path between neighbouring stacks. As we can choose input data, we choose ones causing least divergence.

**Optimization 2**: Sort search stacks after each pass. To decrease divergence, we “coherentize” search by sorting stacks after each path. We compared search algorithms (stable vs quicksort), and search criteria, such as searched value, round number, round change (delta), iterations to round change (njd). We also experiment with “snapping”, that is, requiring the thread to reach round switch before termination.

**Optimization 3**: Triple-loop backward searching. In triple-loop, outermost switches rounds and checks for termination, middle loop iterates over complete messages, and inner loop checks the characteristic. This reduces number of instructions on hotspot path, and reduces divergence.

**Other optimizations**. Contrary to expectations, using on-GPU shared memory improves overall performance by 2.5% only. In addition, 10% improvement is given by removing message characteristic check during message stage.

The best option is stable sort by value + snapping, implemented using on-GPU radix sorting. Sorting takes less than 1% of search time, and improves performance by 85% over baseline.

This work was supported by T-Platforms and RFBR research grant 11-07, 93960-SAR-a. We are also thankful to Moscow State University for providing access to “Lomonosov” supercomputing facility.