MERCATOR: Supporting Irregular Behaviors in Streaming SIMD Computation

Steve Cole, Jeremy Buhler, and Roger Chamberlain
Dept. of Computer Science and Engineering, WUSTL

IRREGULAR STREAMING COMPUTATIONS

SIMD (Single-Instruction, Multiple-Data) architectures such as GPUs naturally support streaming computations that process a long stream of independent data items, items may be grouped into ensembles of size equal to the SIMD width (e.g., GPU block size), with each element processed in parallel by a single SIMD lane (one GPU thread).

Most streaming computing frameworks, such as StreamIt, primarily support regular streaming computations in which each item in an ensemble undergoes identical processing – hence every SIMD lane does the same work. In contrast, irregular computations may perform different, data-dependent work for each item. Irregularity leads to idled SIMD lanes and divergent computation, which reduce application throughput significantly.

MERCATOR framework efficiently expresses irregular streaming computations for SIMD platforms.

- Abstracts irregular behaviors using dataflow representation of application
- Supports irregular behaviors (filtering, divergence) efficiently, transparently using SIMD-parallel algorithms
- Exposes opportunities for optimization

MERCATOR targets NVIDIA GPUs using the CUDA language.

Decision cascades (image recognition, biosequence analysis, sensor data processing)

Pattern matching (e.g. network traffic)

Tree traversal (decision tree evaluation, suffix tree string matching)

Big graph algorithms (belief propagation, page rank, shortest paths, clustering)

The MERCATOR framework efficiently expresses irregular streaming computations for SIMD platforms.

- Abstracts irregular behaviors using dataflow representation of application
- Supports irregular behaviors (filtering, divergence) efficiently, transparently using SIMD-parallel algorithms
- Exposes opportunities for optimization

MERCATOR targets NVIDIA GPUs using the CUDA language.

Design principle: programmer need not explicitly code interaction among SIMD lanes!

Sample MERCATOR Application Topology Spec

```
// A module type describes one or more
// nodes implementing the same code
#pragma mtr module Filter (int(128); int(128); int(128))

// Node declarations (w/module types)
#pragma mtr node sourceNode :: SOURCE
#pragma mtr node filterNode1 :: Filter
#pragma mtr node filterNode2 :: Filter
#pragma mtr node filterNode3 :: Filter
#pragma mtr node sinkNode :: SINK<int(128), int(128), int(128)>

// Edge declarations
#pragma mtr edge sourceNode::out(filterNode1::in[0])
#pragma mtr edge filterNode1::in[0](filterNode2::in[0])
#pragma mtr edge filterNode2::in[0](filterNode3::in[0])
#pragma mtr edge filterNode3::in[0](sinkNode::out)
```

The programmer’s graph is comprised of multiple modules of the same type, each performing the same task. The modules are connected through dataflow edges, which represent the flow of data between modules. Each module is associated with a node type, which defines the input and output data types and the operations performed by the module on the data.

MERCATOR efficiently abstracts irregular streaming GPU computations. Its runtime infrastructure helps performance in the presence of irregular behavior.

References


Research and development on MERCATOR is supported by NSF CISE award 1500173.