Portable Data-parallel Programming in OpenCL

Berkeley Parlab

David Sheffield, Kurt Keutzer

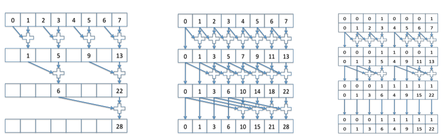
Enabling portable data-parallel programming

Motivation

- Data-parallel processors are here to stay
  - High performance
  - Huge architectural design space
    - Hierarchical data-parallelism
    - Radically different SIMD widths
  - Low cost flops enables new applications
  - OpenCL enables source compatible programming across a wide variety of architectures
    - CPU: Intel Core i7, AMD Opteron, IBM Power7
    - GPU: AMD Radeon, Nvidia GeForce
    - Accelerators: IBM Cell Broadband Engine
- Parallel code is not performance portable
  - Devil is in the architectural details
    - Dynamic versus static instruction scheduling
    - SIMD width
    - Scratchpad memories
    - Atomic operations
      - IBM Cell BE does not have atomic operations

A high-level framework for OpenCL

- We constructed a high-level data-parallel framework
  - Implemented with C++ classes and templates
    - Developer does not need to write OpenCL kernels
  - Element-wise operations
    - Arithmetic instructions
  - Reduction operators

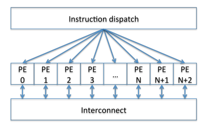


Reduction   Scan   Segmented Scan

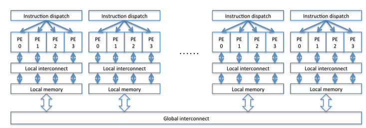
- We are not automatically flattening nested data-parallel kernels
  - NESL was built for different data-parallel processors than available today
- We generate kernels for anonymous element-wise functions
  - OpenCL uses Just-In-Time compilation
    - Just add additional sources for anonymous functions at runtime
  - Generate vectorized and scalar implementations
  - Developer does not need to deal with low-level deals
    - Strip-mining and other unpleasant deals are handled automatically
- Data-parallel primitives optimized for each platform
  - Optimized scan
  - Optimized segmented scan

Flattened data-parallelism

- Flattening worked well on old SIMD machines
  - Single-level of SIMD
  - Flat memory hierarchy



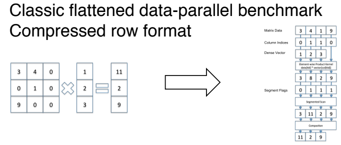
- Modern SIMD machines are hierarchical
  - Multiple-levels of SIMD
  - Deep memory hierarchy
    - Software controlled memories too
  - Thread dispatch is relatively dynamic



- Importance of flattened data-parallelism is proportional to expected segment length
  - Segments shorter than naïve SIMD width will benefit from flattening
  - Trend towards longer SIMD vectors
    - Nvidia G80: 8 wide SIMD
    - Nvidia GF104: 48 wide SIMD

Sparse Matrix-Vector Multiply

- Sparse matrix and dense vector
  - Common in natural systems
- Classic flattened data-parallel benchmark
- Compressed row format



Euclidean distance

- Euclidean distance between several vectors

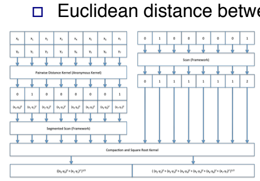


Figure 1: Anonymous function from Euclidian distance benchmark

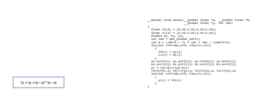
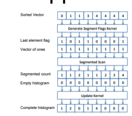


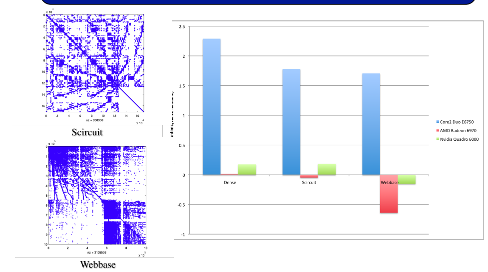
Figure 2: Vectorized anonymous kernel constructed from lambda function

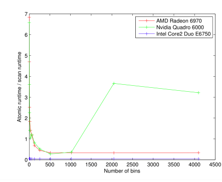
Histogram

- Not all data-parallel accelerators support atomic operations
  - Use sort and and scan instead

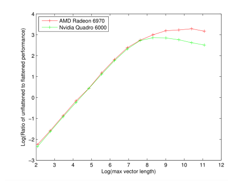


Results





Histogram results



Distance results

- Flattening is essential when segments are shorter than the native SIMD width
- Performance of atomic memory operations is non-intuitive
- Flattening hurts performance on the CPU
- Nvidia GPU handles significant load imbalance without flattening

Future work

- Performance of nested data-parallel code depends on segment lengths
  - Runtime could dispatch select implementation at runtime
- Integrate OpenCL backend into an existing programming language such as Python or Scala
- Use OpenCL as a data-parallel intermediate representation.
  - Generate kernels based on system information
  - Optimized kernel generation for different system architectures
- Optimize primitives for each class of architectures
- Construct more advanced applications using framework