Application: Variant Calling Workflow

- Identifies and characterizes mutations in NGS data
- Map NGS data to reference genome
- Correct for noisy data
- Differentiate strands in the presence of noise and ploidy
- First phase of the personalized medicine workflow

- Recomputedly used
- Data intensive part of NGS analytics

- Data filtering process
  - In: 100GB
  - Out: 50 MB
- Best Practices Workflow:
  - BWA for alignment
  - GATK for BOSR and realignment
  - GATK for SNP calling
- Data-parallel distributed computation

Performance Bottlenecks

- Walltime for human genome @ 50x coverage
  - 437 ± 0.01h on a single node
  - 28.5 ± 0.2h on 22 Blue-Waters nodes

- Very poor resource utilization
- 10 dependent performance limitations:
  1. File system as distributed memory
  2. Htstone: Inefficient for distributed file systems
  3. Sorting large alignments on-disk

Efficient NGS Analytics

- Genome Analytics as Data-Flow Graphs
  - “Kernels” (Vertices): Data transformations
  - “Patterns” (Edges): Data dependencies

- Implicity data parallel
- Composable and pluggable model
- Reuse of computation kernels
- Potential for system level optimizations
- Potential for accelerators

Genome Analytics as Data-Flow Graphs

Performance Enhancements:
- Distributed execution of kernel functions
- RDMA to cut down data-serialization costs
- RPC Control Transfer
- PGAS Memory: Data Transfer
- Efficient data formats
- Columnar store
- Use in-memory representation
- Memory Map IO
- High performance kernels

Preliminary Results

- Synthetic human chromosome 1 @ 50x
- IGen Aligner (vs. SNAP)
  - Single Node: 12k (35 min to 30 min)
  - Multiple Node: 120x
- IGen Variant Caller (vs. GATK HaplotypeCaller)
  - Single Node: 9x (36 min to 41 min)
  - Multiple Node: 8x

Conclusions

- Measurement driven study of performance bottlenecks in existing NGS analytics tools
- Similar performance pathologies across multiple tools
- Scope for system level optimization
- Present a data-flow based abstraction for NGS analytics
- Demonstrate preliminary results of significant performance acceleration
- Simpler to build high performance parts

Ongoing Work

- Improved Kernel Scheduling
  - Optimal task assignment under constraints of:
    - Affinity
    - Shared resource contention
    - Data Locality
- Accelerators
  - Explore the use of GPUs for computationally heavy kernels
  - Custom hardware accelerators
- Deployment Mechanisms
  - Containerized deployments using Docker
  - Integration with HDFS and Tachyon

Acknowledgements

We would like to thank Zhengping Yang, Varun Bahl, Valeria Fornesizza, Ludmila Manara, Arun P. Alireza, Zbigniew Iyer, Zhangwei Kobza, and Victor Jongeneel for their help, support and advice.