GPU accelerated signal processing in the Ion Proton™ whole genome sequencer

Mohit Gupta and Jakob Siegel
Why sequence DNA?
What is DNA

• Hereditary material in humans and almost all other living organisms

• Comprises of four chemical base pairs: (A,T), (C,G) in double helix structure

• Sequence of these base pairs determines how an organism develops, survives and reproduce
Next Generation DNA Sequencing

• High throughput sequencing
  – Enable rapid advances in genome level research

• Massively parallel
  – Requires advanced compute technologies
  – Big data problem in some sense

• Rapid turnaround time
  – Human Genome Project: 1990-2003 ( $3 billion)

• Need to be affordable
  – Democratize DNA sequencing
  – $1000 genome

• Form factor
Ion Proton™ sequencer in 2012

- Simple chip loading
- Intuitive graphical user interface
- State-of-the-art electronics
- Integrated reagent delivery

Innovative semiconductor sequencing chip
Scalable Sequencing Machine on a Chip

Leverages $3 Trillion Investment
Try to imagine:

a 165 Megapixel camera recording RAW images at 30 FPS for 5 seconds for each of 400 nucleotide flows

→ 3.7 TB compressed data in less than 4 hours.
Transistor as a pH meter

Electronic signal processing


Compute Intensive signal processing

The Chip is the Machine
Data Processing Pipeline

19 TB
Data acquisition and compression in FPGA

3.7 TB
Signal Processing

260 GB
BaseCalling and Alignment to reference genome

5/22/13
How to process data at the source?

- Big data on Proton™ Sequencer
- Cloud/Cluster not an option
- Dual 8-core Intel® Xeon® Sandy Bridge
- Dual Altera® Stratix® V
- NVIDIA® Tesla® C2075 (soon K20)
- 11 TB (SSD and HDD)
GPU to the rescue

- Removed main hotspot in signal processing pipeline
- **Speedups of more than 200x over a CPU core!**
Signal Processing
Signal Processing Flow

1. Reading flow data
2. Image Processing
3. Regional Parameter Estimation (Common to all wells)
4. Parameter Estimation unique to each well (LM fitting)
5. Post Fit Processing
6. Writing signal values
Mathematical model

- Sophisticated model
  - Background correction
  - Incorporation
  - Buffering
- Regional Parameters
  - Enzyme kinetics, nucleotide rise, diffusion etc.
- Well Parameters
  - Hydrogen ions generated, buffering, DNA copies etc.
Parameter Estimation

• Rich data fitting on first 20 flows
  – Custom Levenberg-Marquardt algorithm is used for fitting
  – Multiple parameters are estimated
  – Requires a full matrix solve like Cholesky decomposition
  – Required for each well

• A two parameter fit is required in rest of the flows

• Generates signal value corresponding to hydrogen ions generated in each flow
Levenberg Marquardt (LM) Algorithm

• Least squares curve fitting where sum of squared residuals between observed and predicted data is minimized

\[ S(\beta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta)]^2 \]

where

- \( S \): sum of squared residuals between observed and predicted data
- \( \beta \): set of model parameters
- \( y, x \): observed data

• Essentially Gauss Newton (GM) algorithm with a damping factor \( \lambda \) tuned in every solve iteration to progress in the direction of gradient descent
LM algorithm cont’d

• For our application
  – Minimize the residual between raw data and signal value obtained from the mathematical model for each well
  – Provides numerical solution for the parameters governing the model
  – Iterative algorithm executed repeatedly till there is no appreciable change in parameters
  – Convergence in reasonable time and iterations depends on the initial guess for parameters
Challenges with the CPU codebase

• Original code based on many small functions that limit the efficient use of the GPU resources

• Large input buffers need to be transferred to the GPU for every job
  – PCIe bandwidth becomes bottleneck for small jobs
  – Mitigated by end to end implementation on GPU to remove PCIe transfers for intermediate steps

• Data layout not optimal to perform coalesced reads and writes for GPU threading model
Execution Model

• Stream based execution allows for overlap of:
  – Kernel execution
  – PCIe transfers
  – Host side data reorganization
  – CPU pre and post processing

• Queue system for heterogeneous execution
  – Balances execution of different tasks between CPU and GPU
Execution Model cont’d

• Implementation Concept

  – Resources needed for stream execution are pre-allocated and obtained from a resource pool.

  – If resources to create a Stream Execution Unit (SEU) are available the Stream Manager will try to poll a new job from a job queue.

  – The Stream Manager can drive multiple SEUs which can be of different types.

  – Theoretically up to 16 SEUs can be spawned in one Stream Manager if enough resources are available.
GPU Code Optimizations

• Merging smaller kernels into one fat kernel
  – No kernel launch overheads
  – Removes lot of redundant global memory reads and writes
• Invariant code motion
• Instruction reordering to allow for better caching
• Loop unrolling
• Reduction in integer operation for address calculations
• Reduction in register pressure
  – Occupancy was register limited
  – Did a sweep over ‘maxrregcount’ to arrive at optimal value
GPU Memory Optimizations for C2075

• Global memory access optimization

  – Data transform from AoS to SoA.
  – Data reorganization to allow for vec4 reads
  – Use of shared memory to store some heavily accessed elements
  – Padded buffers to allow for more efficient 128byte segment reads
  – “Excessive” use of constant memory
  – Optimization for L1 cache
LM implementation on GPU

• LM for each well is done by one thread on GPU

• Millions of wells
  – Embarrassingly parallel problem
  – High on compute
  – Several iterations required to obtain a solution with a reasonable tolerance

• Thread exits once numerical solution is obtained for the parameters
Bottlenecks of GPU implementation of LM

Divergence among GPU threads in a warp

- Each well is sequencing different DNA fragment
- Need different number of iterations
  - Different number of iterations of increasing or decreasing λ for each iteration
- Towards the end of the algorithm execution, only few threads are active in a warp
- Scarce resources on GPU doesn’t allow LM kernels from two different jobs to overlap
Switch to Gauss Newton Algorithm

- Same as LM algorithm without damping parameter $\lambda$ to be tuned in every iteration
- Converges faster if initial guess is in the vicinity of the solution
- Mitigated the divergence problem caused by LM to a great extent
  - No $\lambda$ tuning iterations are involved
- Reduced code and complexity
- Limit maximum iterations to be performed in worst case scenario
Transition from C2075 to K20

• As expected the codebase worked without changes out of the box.

• Initially the heavily C2075 optimized code and kernel settings did not show a speedup. (?!)
  – Kernel settings fine tuned for occupancy, shared memory and cache usage.
    • K20: same shared/L1 size for more threads in active state
  
  – Heavy use of L1 for global memory read and write access
    • K20: no more L1 caching for global writes and reads

  – Multiple processes using the GPU occupying all it’s memory
    • K20: only ~5GB instead of the ~6GB available on C2075
K20 tweaks

• No more L1 caching for global write and reads
  – Use __ldg() to cache global reads in texture cache
  – Intermediate results moved from global to local memory to benefit from write and read caching in L1

• Reduction in available global memory size
  (no effect on GPU performance but on our CPU side execution model)
  – Dynamic memory management
  – Variable number of streams per GPU context
K20 tweaks

• Same shared/L1 size for more threads in active state
  Correct L1 setting to K20 optimal setting

Performance C2075 vs K20 for different L1 settings

1.65x

5/22/13
Performance Results

• Speedup of a GPU over a single CPU core
  – more than 200x

• Speedup achieved over a 12 core Intel Xeon X5650 @ 2.67GHz and a K20 GPU
  – 17x for algorithm
  – 4.1x whole pipeline
Conclusion and Future Work

• If divergence is the performance limiter
  – Look for alternate algorithm which suits the GPU
• Profiling alone doesn’t help
  – Need to search for optimum over the space (cache size, reg count, block size etc.)
• Timing bottleneck shifted to other parts of the pipeline
  – Alignment
    • Well known algorithm Smith waterman for detailed alignments
    • Some good GPU implementations already out there
• Prepare for PII and PIII chips
  – Exploit Kepler architecture to its maximum potential
  – Take some parts of signal processing pipeline to GPU
    • Crucial with high density of wells on these chips
Shift focus to secondary Analysis

Current Template Preparation, Sequencing, and Analysis times for PI Chip
(customer configuration: Dual 8-core Intel® Xeon® Sandy Bridge, NVIDIA® Tesla® C2075)
40 Years of Accumulated Moore’s Law

In 2 years ION has increased throughput 1,000 Fold.
Thank You

NVIDIA
specifically the DevTech Team
Sarah Tariq
Jonathan Bentz
Mark Berger
Kimberley Powell

The entire Ion Torrent R&D team
For Research Use Only; Not for use in diagnostic procedures.

All data shown in this presentation was generated by Life Technologies, except where indicated

© 2013 Life Technologies Corporation. All rights reserved. The trademarks mentioned herein are the property of Life Technologies Corporation and/or its affiliate(s) or their respective owners.
LM algorithm

- Need to solve the following equation for $\delta$

$$
\left( J^T J + \lambda \text{diag}(J^T J) \right) \delta = J^T (y - f(\beta))
$$

where

- $J$: Jacobian matrix
- $\delta$: Increment vector to be added to parameter vector $\beta$
- $y$: vector of observed values at $(x_0, \ldots, x_n)$
- $f$: vector of function values calculated from the model for given vector of $x$ and parameter vector $\beta$
- $\lambda$: damping parameter to steer the movement in the direction of decreasing gradient
LM algorithm cont’d

• Run an outer loop on some maximum number of iterations
  – Exit if no appreciable change in parameter value from previous to current iteration

• Run an inner loop
  – Worse case loop count depends on max or min value of $\lambda$
  – Increase $\lambda$ if residuals increase compared to previous outer iteration
  – Decrease $\lambda$ if residual decrease compared to previous outer iteration
Effect of cache settings on performance

Performance C2075 vs K20 for different L1 settings