Single Pass, BLAST-like, Approximate String Matching on FPGAs*

Martin Herbordt  Josh Model  Yongfeng Gu
Bharat Sukhwani  Tom VanCourt

Computer Architecture and Automated Design Laboratory
Department of Electrical and Computer Engineering
Boston University
http://www.bu.edu/caadlab

*This work supported, in part by the U.S. NIH, the Naval Research Lab, and by Lincoln Labs

the talk, one minute version …

- Approximate string matching (ASM) is a fundamental operation of bioinformatics
- High performance ASM is important not only for everyday queries, but also for integration as a subroutine into larger applications
- HW BLAST has been challenging because it depends on random access into GB scale databases
- We side-step this problem by streaming the (entire) databases through the FPGA and processing queries at streaming rate
- The naïve structure to do this is a 2D systolic array w/ O(M^2) HW
- The HW can be reduced to O(M) by recasting the alignment algorithm
- Implementations show that this works as described on a PC w/ COTS board (running NCBI blastp)
- We have various other results/proofs that we don’t have time for in this talk
Outline

1. Introduction to Approximate String Matching (ASM)
2. Overview of HW implementations of ASM
3. Our FPGA algorithms
4. Hardware, validation, limitations, results
5. Summary and extensions

Bioinformatics – The Fundamental Abstraction

*Genes and proteins can be represented as strings:*

Nucleic Acids ➔ A C T G
Amino Acids ➔ A C D E F G H I K L M N P Q R S T V W Y
Bioinformatics – The Fundamental Insight

Observation:

*Similar sequences (after some transformations or folds)* create *similar proteins with similar function*

Observation:

*Distant organisms have similar proteins that provide similar functions*

Therefore →

*Given the sequence of an unknown gene, we can predict its function by finding known genes with similar sequences*

Bioinformatics – Everyday Operation

Queries to NCBI web site consist of the …

– Query string
– Database to be queried against
– Type of query (DNA, protein)
– Sensitivity parameters
  • Trade off missing potential matches versus compute time

From NCBI point of view – types of queries received*

(for proteins, nucleotides are analogous)

– > 90% of queries are < 1000 residues
– > 80% of database selections are for *nr* ("non-redundant")
  ~millions of proteins, ~1.8 billion residues

(convention → query length = M, database size = N)

*Couloris2005*
Example: GCGATCT versus GCATTTTA

- Each character-character match (G-G, C-C, G-A ...) is scored independently with a scoring matrix
  - Identity (or close match) = positive score
  - Mismatches (change of type) = negative score

**Gonnet Pam250 Matrix**

|     | A | C | D | E | F | G | H | I | K | L | M | N | P | Q | R | S | T | V | W | Y |
| A   | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| C   | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| D   | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| E   | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| F   | 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| G   | -1| 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| H   | -2| -1| 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| I   | -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| K   | -2| -1| 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| L   | -1| 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| M   | 0 | -1| 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| N   | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| P   | 2 | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| Q   | 3 | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| R   | 4 | 3 | 2 | 1 | 0 | -1| -2| -3| -2| -1| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|
| S   | 5 | 6 | 7 | 8 | 9 | 10|
| T   | 6 | 7 | 8 | 9 | 10|
| V   | 7 | 8 | 9 | 10|
| W   | 8 | 9 | 10|
| Y   | 9 | 10|

- An alignment is a possible way for sequences to match (char-char)
  - some possible ungapped alignments:

  | GCGATCT | GCGATCT | GCGATCT | GCGATCT | GCGATCT |
  | GCATTTA | GCATTTA | GCATTTA | GCATTTA | GCATTTA |

  - each alignment generates a **ScoreSequence**

<table>
<thead>
<tr>
<th>query:</th>
<th>G</th>
<th>C</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>database:</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>A</td>
</tr>
<tr>
<td><strong>ScoreSequence:</strong></td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Approximate String Matching – Basics

Example: Evaluate the ScoreSequence 1 1 -1 -1 1 -1 -1

• Each character-character match (G-G, C-G, etc.) is scored independently with a scoring matrix
• An alignment is a possible way for sequences to match (char-char)
• To score an alignment (i.e., a single ScoreSequence of length M)
  – global alignment: sum scores in the ScoreSequence
  – local alignment: find score of highest scoring subsequence of ScoreSequence

```
# Find maximal local alignment in a single alignment
# \rightarrow Find max cumulative score with cut-off = 0
# \rightarrow Complexity = O(M)

SimpleScoring
Traverse ScoreSequence
Add NextScore to CurrentScore
If CurrentTotal > MaxScore, update MaxScore
If CurrentTotal < 0, set CurrentTotal to 0
```

Approximate String Matching – Basics

Example: GCGATCT versus an entire database

• Each character-character match (G-G, C-G, etc.) is scored independently with a scoring matrix
• An alignment is a possible way for sequences to match (char-char)
• To score an alignment (evaluate a single ScoreSequence) …
• Simple algorithm to find maximal ungapped local alignment of all possible ungapped alignment (i.e. N ScoreSequences) …

```
# Find maximal local alignment of all ungapped alignments
# \rightarrow Find max cumulative score with cut-off = 0
# \rightarrow Complexity = O(MN)

Traverse Database – Foreach Alignment
  Generate ScoreSequence
  Do SimpleScoring
```

• Complexity of gapped alignments is potentially unbounded
ASM – Gapped alignments (DP-based methods)

- Create query/database tableau:

```
<table>
<thead>
<tr>
<th>G</th>
<th>C</th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

- Traverse the tableau with a Dynamic Programming algorithm →
  Score each grid cell \((i,j)\) \(S_{ij}\) is computed using the following recurrence:

\[
S_{ij} = \begin{cases} 
  0 & \text{if } (i,j) = 0,0 \\
  S_{i-1,j} - S_{pap} & \text{if } i > 0 \\
  S_{i,j-1} - S_{pap} & \text{if } j > 0 \\
  \text{else} & \text{if } S_{i,j} > \max \left( S_{i-1,j} - S_{pap}, S_{i,j-1} - S_{pap}, S_{i-1,j-1} + s(q_i,r_j) \right) \\
  \text{else} & \text{if } S_{i,j} = \max \left( S_{i-1,j} - S_{pap}, S_{i,j-1} - S_{pap} \right)
\end{cases}
\]

- Complexity: \(O(MN)\)

ASM – An even better way (BLAST)

The BLAST heuristic …

1. Look for small clusters of matches on main diagonal
2. Try to extend those (and only those) clusters
3. Try to merge those extended clusters
   - e.g. using DP methods on regions of interest

- Complexity: \(O(N) + O(M^2)\) with \(M << N\)
Outline

1. Introduction to Approximate String Matching (ASM)
2. Overview of HW implementations of ASM
3. Our FPGA algorithms
4. Hardware, validation, limitations, results
5. Summary and extensions

Systolic HW Implementation of DP/ASM

DP processing follows main diagonal … leading to a wavefront dependency (A), which is easily computed with a linear array (B).

- Complexity with M cells: $O(N)$
What’s hard about HW BLAST?

- Random access into database for extensions

- The serial version is already $O(N)$

- DP is already $O(N)$ and handles gaps!

Outline

1. Introduction to Approximate String Matching (ASM)
2. Overview of HW implementations of ASM
3. Our FPGA algorithms
4. Hardware, validation, limitations, results
5. Summary and extensions
DP and BLAST are duals of each other …

DP \rightarrow \text{processes } M \text{ alignments simultaneously;}
\rightarrow \text{processing is perpendicular to main diagonal}

BLAST \rightarrow \text{processes 1 extension at a time}
\rightarrow \text{processing is parallel to main diagonal}

DP HW \rightarrow \text{advances one db character per cycle}
BLAST HW \rightarrow \text{advances one db character per cycle??}

---

**A HW \textbf{Intensive} Implementation of BLAST**

- For phase 3 (gapped alignment) use a systolic implementation of DP
  \textit{complexity} = \( O(M) \)

- For phases 1 & 2 (ungapped alignment) create streaming hardware
  \textit{complexity} = \( O(N) \)

  \textbf{Q:} Why bother if already asymptotically optimal?
  \textbf{A:} May still be able to run substantially faster (i.e., smaller constant within the } \( O() \)

- Idea: Use a 2D systolic structure

---

**2DSystolicBLAST**

\# In a single cycle \( \rightarrow \)
Dimension 1: Foreach alignment, generate ScoreSequence

\# In M cycles for each ScoreSequence
\# process M ScoreSequences simultaneously \( \rightarrow \)
Dimension 2: Foreach ScoreSequence
use SimpleScoring to generate local alignment

\# Time Complexity = \( O(N) \)
\# Area Complexity = \( O(M^2) \)
TreeBLAST – Optimize the HW (v1)

Problem w/ “HW-intensive” BLAST: \( M \approx 1000 \rightarrow M^2 \approx 1,000,000 = \text{too big!} \)

Solution: Optimize 2nd dimension
- Redo SimpleScoring to use \textit{iterative merging}
- \( \log_2(M) \) depth tree \( \rightarrow \) HW = \( O(M) \)

Operation:
- Query string held in place, database streams over it
- On each cycle (alignment), one ScoreSequence generated
- ScoreSequences evaluated systolically by the tree structure

\[ \begin{array}{c}
\text{TreeBLAST} \\
\# \text{In a single cycle } \rightarrow \\
\text{Dimension 1: Foreach Alignment } \\
\text{generate ScoreSequence} \\
\# \text{In } \log_2(M) \text{ cycles for each ScoreSequence} \\
\# \text{process } \log_2(M) \text{ ScoreSequences} \\
\# \text{simultaneously } \rightarrow \\
\text{Dimension 2: Foreach ScoreSequence } \\
\text{use tree structure to generate local alignment} \\
\# \text{Time Complexity = } O(N) \\
\# \text{Area Complexity = } O(M)
\end{array} \]
TreeBLAST – What the 1D array looks like

To generate **ScoreSequences** →

- Preload 1D look-up tables in Block RAMs (columns of scoring matrix)
  - one entry per pair of match pairs (~400 entries)
  - i.e., indices are doublets of database characters

- Each Block RAM provides match scores for four characters per cycle
  - 300 BRAMS → ScoreSequence of length 1200 in a single cycle

TreeBLAST – What the nodes look like

Each node requires only
- four words storage
- small amount of logic
TreeBlast – Why this is interesting …

Induction:
• Characterize any sequence of numbers with four quantities. Throw away the sequence.
• Those four quantities can be combined simply to generate the same quantities for what would be the concatenation (in either direction) of the original sequences.

Combine parameters of black box string 1 ➔

with those of black box string 2 (in either direction) ➔

and obtain the maximal subsequence with cut-off

TreeBLAST – Why this works

Proof:

(at 8:45 in the morning???)
TreeBLAST is highly sensitive

ServerBLAST – Optimize the HW (v2)

**Solution:** (again) optimize 2nd dimension of 2D array

- Keep original SimpleScoring algorithm
- Reduce number of SimpleScoring servers from M to a **small number k**
- Only process ScoringSequences that have a least one seed
- Determine seed in parallel in a single cycle during ScoreSequence generation
**ServerBLAST Behavior**

Question: What’s faster →
- the rate at which alignments of interest (AOIs) are generated, or
- the rate at which alignments (ScoreSequences) can be processed?

Answer 1: rate of AOI generation > processing rate
Overall processing time = |AOIs| x T_{AOI\_processing} / k

Answer 2: rate of AOI generation << processing rate
Overall processing time = database size x cycle time

---

**Experiment:**
- LIM2 protein (200 amino acids) versus *e. coli* (1.7M amino acids)
- Medium sensitivity
- k = 20 sufficient to achieve near-streaming rate.
- k = 10 is sufficient to obtain ½ streaming rate
- k = 20 is achievable on current FPGAs for small-medium sized queries

**Extension (not yet implemented):**
- Look for multiple seeds (collinearity) to drastically reduce frequency of AOIs
- Requires 3-5 cycles, rather than 1
Outline

1. Introduction to Approximate String Matching (ASM)
2. Overview of HW implementations of ASM
3. Our FPGA algorithms
4. Hardware, validation, limitations, results
5. Summary and extensions

System Implementation

- Board: Annapolis Microsystems Wildstar II Pro for PCI
- 2 Xilinx Virtex II Pro VP70 -5 FPGAs (only one is used)
  Xilinx, Synplicity, and ModelSim design tools
- 2.8 GHz Xeon Dell PC running Windows XP
- Database stored on 150GB IDE-connected NTFS drive
- DMA drivers from Annapolis Microsystems
- Streaming disk I/O routines from Microsoft

Also,
- TreeBLAST implemented through post place-and-route
  only on a Xilinx XC4VLX160
Validation

- Validated against NCBI BLAST (blastp) for Win32 version 2.2.13
- Source and query strings as suggested by bioperf.org
  - 10 sequences chosen at random from *e coli* and queried versus *drosophila* database
- Parameters
  - Neighborhood (word) size = 3
  - Threshold for seed extension = 11
  - Drop-off for ungapped extension = 7
  - Reporting cutoff = 66 (raw score)
- Ungapped BLAST was run
- Collinear merges ignored *(moot if DP run as follow-on)*

*Identical pair-wise alignments and scores obtained on multiple runs*

Performance

**FPGA Performance – TreeBLAST**

Max query size
- VP70 (direct) = 600
- VP70 (folded) = 1200

Throughput
- VP70 -5 (direct) = 110Maa/sec
- VP70 -5 (folded) = 55Maa/sec
- XC4VLX160 (direct) = 178Maa/sec
  - Post place-and-route only

Sensitivity
- word size = unbounded
- threshold = 0
- drop-off = 100%

**System Performance**

- Disk to FPGA streaming rate = .5 MB/sec
- Memory to FPGA streaming rate = 320 MB/sec
Performance Comparisons

Difficulties in making BLAST comparisons – performance highly dependent on:
• Size of query, size of database, actual query/database combination
• Sensitivity

Queries of *e. coli* proteins versus *nr* database (1.8GB), nominal sensitivity for NCBI

<table>
<thead>
<tr>
<th>Query Size</th>
<th>NCBI TreeBLAST</th>
<th>NCBI BLAST on same PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 600</td>
<td>17 seconds</td>
<td>preprocessing 457 seconds</td>
</tr>
<tr>
<td>&lt; 1200</td>
<td>34 seconds</td>
<td>query processing 563 seconds</td>
</tr>
</tbody>
</table>

NCBI BLAST on NCBI server farm* 20-30 seconds
- Several hundred CPUs
- Databases preprocessed, in memory, and distributed among CPUs
- Query serviced by multiple CPUs

*NCBI2006

Outline

1. Introduction to Approximate String Matching (ASM)
2. Overview of HW implementations of ASM
3. Our FPGA algorithms
4. Hardware, validation, limitations, results
5. Summary and extensions
System Integration Scenarios (1)

- FPGA/DP versus FPGA/BLAST → TreeBLAST 12x faster than FPGA/DP

- Production version of HW BLAST requires FPGA/DP pass –
  - Pipe highest scoring sequences to a 2nd FPGA
  - Even better: reconfigure FPGA for final pass!
  - Time is still negligible
    \[(\text{number of sequences of interest} \times O(M)) \approx \frac{1}{100,000} \text{ of BLAST pass}\]

- Applicability to multi-FPGA systems – trivial parallelization
  - Partition databases and stream each part to a different FPGA

- Likely limiting factor: FPGA operating frequency

- Added resources for FPGA/DP for many-FPGA systems …
  - Large number of BLAST FPGAs → small number of DP FPGAs

System Integration Scenarios (2)

Optimization: for small-medium queries, reconfigure FPGA(s) to have multiple trees

Dealing with large queries: partition query into sequences that fit
  - Bioinformaticists approve

**Scenario:**
- Component: FPGA + 256MB memory
- 200+ MHz TreeBLAST
- Two trees per FPGA on average
- Throughput versus \(nr\) would be \((\text{#-of-FPGAs} / 5)\) queries per sec
- 1000 “components” could service 200 queries/second, ~20 million per day, at high sensitivity
Summary

Questions?