Transforming Irregular Algorithms for Heterogeneous Computing - Case Studies in Bioinformatics

Jing Zhang
Advisor: Dr. Wu Feng
Collaborator: Hao Wang
Irregular Algorithms

• Characterized by ...  
  – Operate on **irregular data structures** - trees, graphs, linked lists, priority queues  
  – Data are processed in variable-iteration loops  
  – Both memory access and control flow are **irregular** and **unpredictable**

• Emerging data intensive applications have increasing irregularity in memory access and control flow over massive data set  
  – Bioinformatics applications  
    • Human genome: 3.2 billion letters (3.2 Gb)  
  – Social network analysis, graph algorithms  
    • Twitter (June-Dec 2009) - 17 million users, 476 million tweets (6Gb)
Heterogeneous Computing Systems

- Heterogeneous computing systems can offer massively parallel computation with high power efficiency and throughput
  - GPU, AMD APU
  - Intel Xeon Phi
  - FPGA, DSP, ...

- But, designed for regular programs, relying on data locality and regular computation to tolerate access latencies
  - Thousands of smaller (weak), more efficient (simple) cores
  - Limited cache size, i.e., short cache line lifetimes
Heterogeneous Computing Systems

• Heterogeneous computing systems can offer massively parallel computation with high power efficiency and throughput
  – GPU, AMD APU
  – Intel Xeon Phi
  – FPGA, DSP, ...

• But, designed for regular programs, relying on data locality and regular computation to tolerate access latencies
  – Thousands of smaller (weak), more efficient (simple) cores, which lack branch prediction, out-of-order execution, etc.
  – Limited cache size, i.e., short cache line lifetimes

It is very challenging to map irregular algorithms on heterogeneous computing systems!
Impacts of Irregularity on GPU Performance

• Irregular control flow
  – Branch divergence
    • Occurs when threads inside warps branches to different execution paths
    • Waste computational resource
Impacts of Irregularity on GPU Perf (cont’d)

- Irregular memory access
  - Uncoalesced memory access
  - Lower memory bus utilization

One segments
A single 128-byte transaction
**Coalesced Memory Access**

Cross N segments
N 32-byte transaction
**Uncoalesced Memory Access**
Irregular Algorithms Optimizations on GPU

• Irregular control flow
  – Kernel fission
  – Sorting work
  – Warp-centric execution
  ……

• Irregular memory access
  – Memory access reordering
  – Exploiting memory hierarchy
  – Data structure adaptation
  ……
Case Study: Mapping BLASTp on GPU*

• BLAST - Basic Local Alignment Search Tool
  – Find most similar sequences from database for a query sequence
  – Use **heuristic** algorithms, which have significant irregularities in both memory access and control flow

• P is for protein sequence; N is for nucleotide sequence

BLAST Algorithm

Four stages in BLAST
1. Hit detection
2. Ungapped extension
3. Gapped extension
4. Final extension
BLAST Algorithm

Four stages in BLAST
1. Hit detection
2. Ungapped extension
3. Gapped extension
4. Final extension
BLAST Algorithm

Four stages in BLAST
1. Hit detection
2. Ungapped extension
3. Gapped extension
4. Final extension
BLAST Algorithm

**query position, subject position**

### Lookup Table

- **ABA**: 0,0
- **ABB**: none
- **ABC**: 0,5

**Subject Sequence**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>B</th>
<th>B</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**Query Sequence**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

**Four stages in BLAST**
1. Hit detection
2. Ungapped extension
3. Gapped extension
4. Final extension
BLAST Algorithm

Four stages in BLAST
1. Hit detection
2. Ungapped extension
3. Gapped extension
4. Final extension
BLAST Algorithm

Four stages in BLAST
1. Hit detection
2. Ungapped extension
3. Gapped extension
4. Final extension
BLAST Algorithm

Four stages in BLAST
1. Hit detection
2. Ungapped extension
3. Gapped extension
4. Final extension
BLAST Algorithm

Four stages in BLAST
1. Hit detection
2. Ungapped extension
3. Gapped extension
4. Final extension
BLAST Algorithm

Four stages in BLAST
1. Hit detection
2. Ungapped extension
3. Gapped extension
4. Final extension

Knowledge: “seed-and-extend” – the most widely used alignment paradigm
State-of-the-Art GPU BLAST Approaches*

- Use intuitive parallelization - “coarse-grained parallelization”, where a thread compares the query sequence to a subject sequence

*: Design and Implementation of a CUDA-compatible GPU-based Core for Gapped BLAST Algorithm (ICCS ’10)

GPU- BLAST: Using Graphics Processors to Accelerate Protein Sequence Alignment CUDA-BLASTP (Biinformatcs)

CUDA-BLASTP: Accelerating BLASTP on CUDA-enabled Graphics Hardware (TCBB)

Accelerating Protein Sequence Search in a Heterogeneous Computing System. (IPDPS ’11)
Challenge #1: Control Flow Irregularity

- With coarse-grained parallelism, different threads may execute across different stages

![Diagram with subject sequences and hit detection](image)
Solution: Kernel Fission

- Split hit detection and ungapped extension stage into separate kernels
- Use intermediate kernels to bridge the two stages
Challenge #2: Control Flow Irregularity (2)

- Coarse-grained extension, where a thread is responsible for a diagonal, can result in divergence, since different diagonals could be extended to different lengths.
Solution: Warp-centric Extension

- Map a warp of threads to one diagonal
- Threads in a warp checks different positions concurrently

![Warp-based Extension Diagram]
Challenge #3: Memory Access Irregularity

• Use a global array, called lastHit Array, to record the previous hit in each diagonal
Challenge #2: Memory Access Irregularity

- Read LastHit Array to get last hit position in the diagonal
- Write back current position to lastHit Array

Subject Sequence:

<table>
<thead>
<tr>
<th>Query Sequence</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 A A B B A B C</td>
<td>Read 0</td>
</tr>
<tr>
<td>1 B B B A B C</td>
<td>Write 0</td>
</tr>
<tr>
<td>2 C C C A B C</td>
<td>Operation 0</td>
</tr>
<tr>
<td>3 C C C A B C</td>
<td>Position 0</td>
</tr>
<tr>
<td>4 B B B A B C</td>
<td></td>
</tr>
</tbody>
</table>
Challenge #2: Memory Access Irregularity

- Read LastHit Array to get last hit position in the diagonal
- Write back current position to lastHit Array

Subject Sequence:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

Query Sequence:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

Thread 0

Operation | Position
--- | ---
Read | 0
Write | 0
Read | -5
Write | -5
Challenge #2: Memory Access Irregularity

- Read LastHit Array to get last hit position in the diagonal
- Write back current position to lastHit Array

Subject Sequence

<table>
<thead>
<tr>
<th>Query Sequence</th>
<th>Subject Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
</tr>
</tbody>
</table>

Thread 0

<table>
<thead>
<tr>
<th>Operation</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>0</td>
</tr>
<tr>
<td>Write</td>
<td>0</td>
</tr>
<tr>
<td>Read</td>
<td>-5</td>
</tr>
<tr>
<td>Write</td>
<td>-5</td>
</tr>
<tr>
<td>Read</td>
<td>5</td>
</tr>
<tr>
<td>Write</td>
<td>5</td>
</tr>
</tbody>
</table>

Thread 0: Read LastHit Array to get last hit position in the diagonal. Write back current position to lastHit Array.
Challenge #2: Memory Access Irregularity

- Read LastHit Array to get last hit position in the diagonal
- Write back current position to lastHit Array

<table>
<thead>
<tr>
<th>Query Sequence</th>
<th>Subject Sequence</th>
<th>Operation</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>Read</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>Write</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>Read</td>
<td>-5</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>Write</td>
<td>-5</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>Read</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>Write</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>Read</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>Write</td>
<td>0</td>
</tr>
</tbody>
</table>

Thread 0

Query Sequence

Subject Sequence

Operation

Position

Read | 0
Write | 0
Read | -5
Write | -5
Read | 5
Write | 5
Read | 0
Write | 0

-5

5
Challenge #2: Memory Access Irregularity

- Irregular Read/Write Ops on LastHit array

```
Thread 0
0 1 2 3 4 5 6 7
A B C B B A B C
```

Subject Sequence

```
Query Sequence

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>
```

Operation | Position
---|---
Read | 0
Write | 0
Read | -5
Write | -5
Read | 5
Write | 5
Read | 0
Write | 0

Irregular Memory Access
Solution: Memory Access Reordering

- Collect and group hits by diagonal numbers via binning

![Diagram showing binning and out-of-order access]

Threads

0 1 2 3 4 5 6 7

A B C B B A B A

Query Sequence

0 1 2 3 4 5 6 7

A B C C A B A

Subject Sequence

Binning

Bins

0,0 0,5 5,0 5,5 Out-of-order

5,5 0,0 ...

-5 5,0
Solution: Memory Access Reordering

- Sort hits by positions in each bin
Solution: Memory Access Reordering

- Filter out non-extenable hits by distance of adjacent hits
Solution: Memory Access Reordering

- Transform an irregular algorithm (lastHit array method) into three regular phases
Performance Numbers

- Compared to original version, cuBLASTP has...
  - Less branch divergence overhead

<table>
<thead>
<tr>
<th></th>
<th>cuBLASTP - Hit Detection</th>
<th>cuBLASTP - Hit Sorting</th>
<th>cuBLASTP - Hit Filtering</th>
<th>cuBLASTP - Ungapped Extension</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch Divergence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overhead*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*: Branch Divergence Overhead: the ratio of divergent execution to total execution
Performance Numbers

- Compared to original version, cuBLASTP has ...
  - Less branch divergence overhead
  - Better occupancy

* Branch Divergence Overhead: the ratio of divergent execution to total execution
Performance Numbers

• Compared to original version, cuBLASTP has ...
  – Less branch divergence overhead
  – Better occupancy
  – Better load efficiency

* Branch Divergence Overhead: the ratio of divergent execution to total execution
Speedup

• With swissprot database, cuBLASTP achieves ...
  • Up to 2.6-fold speedup of hit detection and ungapped extension over GPU-BLASTP, which is the fastest GPU-based BLAST
  • Up to 1.6-fold overall speedup over GPU-BLASTP (cuBLASTP has parallelization of the rest of stages)
Conclusion and Future Work

Conclusion
• Mapping irregular algorithms to heterogeneous system is non-trivial, need couples of transformation and adaption on algorithms and data structures
• But these concepts of transformations are propagable
  – Kernel fission, memory access reordering, warp-centric execution, ...

Future Work
• Generalize optimizations from cuBLASTP for different algorithms and other heterogeneous systems
• Design a library for irregular algorithms with optimized irregular data structure and operations
Other Research Topics

• Optimizing BWA (Burrows Wheeler Alignment) on multicore CPU and Intel Xeon Phi
• dbblast: database index based BLAST on multicore CPU and Intel Xeon Phi
• Dynamic parallelism on AMD APU (New)
• Optimizing irregular applications for GPU/Xeon Phi clusters (Future)

• Other Research Interests
  – Cloud computing
  – MapReduce with accelerators