Design and Evaluation of a Scalable and Portable, Reconfigurable Computing Implementation of BLAST

Parag Beeraka
parag@ittc.ku.edu

Computer Systems Design Lab (CSDL)
Information and Telecommunication Technology Center (ITTC)
University Of Kansas
03/13/2006
Outline

✧ Introduction/Motivation
✧ Background: Reconfigurable Computing (RC) and BLAST
✧ Analysis
✧ RC Implementation
✧ Results
✧ Conclusion & Future Work
Motivation
BLAST

- similarity search for biological databases
- based on heuristics
- successful for several reasons
  - results are statistically characterized
  - source is available
  - runs on many machines (desktops and up)
- molecular biologists use it as an *abstract tool*

* BLAST — Basic Local Alignment Search Tool
BLAST Similarity Search

one way BLAST is commonly used:

- researcher sequences the genome of a new organism
- want to know the function of a particular gene, but wet lab tests are expensive (in time and money)
- search against known gene functions does preliminary screening
Problem Size vs Processor and I/O Performance

- uP 59% CAGR
- I/O Perf. 10% CAGR
- (10 year) CAGR 77% db size

Number of nucleotides vs year:
- #bases(genbank)
- mp(x)
- bio(x)
- io(x)

(10 year) CAGR 77% db size

uP 59% CAGR

I/O Perf. 10% CAGR

Parag Beeraka
Problem and Possible Solutions

same query will take longer on a workstation next year — how much longer will compound annually

Possible Solutions

- Use a cluster of machines which can run in parallel
  - What about cost in terms of power, I/O subsystems and networking?

- Use Reconfigurable Computing
  - How difficult it is to port BLAST to this method of computing
  - Is the solution scalable, portable and cost-effective
Thesis Question

Is a scalable and cost-effective Reconfigurable Implementation of BLAST feasible which can aid scientists and biologists in a more productive way?
Background
Reconfigurable Computing

- Reconfigurable Computing
  - A form of computing where the hardware has the ability to re-configure based on various functions
  - Usually use FPGA’s as the hardware

- Our goal is to partition BLAST
  - Software
  - Hardware as a FPGA core

* FPGA — Field Programmable Gate Array
Target Hardware

- Xilinx Virtex II Pro FPGA
  - Two PowerPC 405 processors
  - 13,696 freely available logic slices
- 256 MB DDR SDRAM
- Compact FLASH
- Ethernet
- Serial Port

Xilinx ML - 310
Standard Linux Virtex II Pro FPGA Configuration

Internals of the Virtex II Pro FPGA

Parag Beeraka
Basic Idea of Similarity Search

- **subject** database — long string of annotated nucleotide sequences

- **query** — an acquired nucleotide sequence that may have some similarity to part of a sequence in the subject

- task:
  find all positions in subject where the query is *almost* the same; by almost we mean the bases match or there is a tolerable number of ...
  
  insertions  substitutions
  deletions  transpositions

Parag Beeraka
BLAST Heuristic

- use query to build a data structure for lookup
- stream subject in sequence-by-sequence
- if the base pairs match the query try to expand to the left and right until tolerance is exceeded
- if tolerance is met, call it a match and output its position
BLAST Example

- subject seq: `ATGCATGTTTTTCGTATGATGAGCTTTAT`
- query seq: `ATGCATGTTTTTCTATGAAGAGCTTT`

Score = 26.3 bits (13), Expect = 0.002
Identities = 23/24 (95%), Gaps = 1/24 (4%)
Strand = Plus / Plus

Query: 1 atgcatgtttttcgtatgaagagcttt 24

Sbjct: 1 atgcagtttttc-tatgatgagcttt 23
Previous RC and BLAST Work

- K. Logue [Honors Thesis 2000] — simulation of one unit
  - simulated a network of cores to improve computational
    - latency
    - throughput
  - implemented one core on PCI-based FPGA card
    5 $\times$ slower!
Initial Experiments with BLAST

- Ran BLAST on various high-end and low-end machines with various queries and databases

<table>
<thead>
<tr>
<th>Mount type</th>
<th>500Mhz &amp; 512MB RAM</th>
<th>1000Mhz &amp; 1024MB RAM</th>
<th>2400Mhz &amp; 512MB RAM</th>
<th>3200Mhz &amp; 2048MB RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFS</td>
<td>152m</td>
<td>48m</td>
<td>77m</td>
<td>17m</td>
</tr>
<tr>
<td>local drive</td>
<td>125m</td>
<td>39m</td>
<td>48m</td>
<td>16m</td>
</tr>
</tbody>
</table>
Initial Experiments with BLAST (cont’d)

Ran BLAST on various high-end and low-end machines with various queries and databases

![BLAST Performance vs. Processor Clock Freq.](chart)

- BLAST over NFS
- BLAST over IDE
- Ideal (x)

Execution Time (m) vs. Clock Frequency (MHz)

- 500 MHz 512MB RAM
- 1024 MB RAM
- 1100 MHz 1024MB RAM
- 2048MB RAM
- 2400 MHz 512MB RAM
- 3200 MHz 2048MB RAM
Profiling BLAST

So, need to figure out the most time consuming segment in BLAST.

- Added the option ‘-pg’ to the gcc compiler when compiling BLAST
- Used the tool ‘gprof’ to determine the most time consuming segment by running it on the executable
- The sub-routine **BlastNtWordFinder** took 85% of the time
- **BlastNtWordFinder** was further analyzed to determine the lines of code that were the most time consuming
- Converted the most time consuming part as a different function called **critical_code** (83%)
RC Implementation
RC Implementation of BLAST

Since ML-310 has ...

- Virtex II Pro FPGA’s with PowerPC 405’s can run Linux
- Cross compiled BLAST runs on Linux on the PowerPC 405’s
- Use Xilinx EDK software to build a complete system
- User IP cores can be added to any of the Buses

* IP — Intellectual Property
* EDK — Embedded Development Kit
RC Implementation of BLAST

Thought....

- Most time consuming segment of BLAST can run on slices on the FPGA as an user IP Core
- Access to the user IP core can be achieved through a Linux Device Driver
- Patch BLAST to add access to the Linux Device Driver

* IP — Intellectual Property
* RC — Reconfigurable Computing
RC-BLAST Base System

- Designed an IP Core and a Lookup Table to replace the **critical code** function
- RC-BLAST IP Core was added as a Master - Slave Peripheral to the On-Chip Peripheral Bus
- Lookup table was designed such that it can be loaded onto External Memory

![Diagram of RC-BLAST IP Core and Bus Connections]

Parag Beeraka 24
RC-BLAST IP Core

- Interfaced to the OPB IPIF interface which interfaces to the OPB Bus
- Input FIFO of length 1024 and width of 32 for writing databases
- Output FIFO of length 1024 and width of 32 for writing match information
- Registers to keep track of FIFO counts
- Registers to keep track of Database count
- OPB Slave State Machine
- OPB Master State Machine to read data from external memory
RC-BLAST Lookup Table

- Table is $2^{16} \times 64$ bits
- Column 1 - flag bit = 1 if hits are $\geq 3$
- Column 2 - pointer if hits $> 3$
- Column 3 - Query Offset for Hit1
- Column 4 - Query Offset for Hit2
- Column 5 - Query Offset for Hit3
- Existing design has no support for hits $> 3$
RC-BLAST Software

素晴 Device Driver

- rcblast_open() – Initializes the hardware registers
- rcblast_write_lut() – Writes the Lookup table to the external memory
- rcblast_write_fifo() – Writes databases to the FIFO’s
- rcblast_read_fifo() – Reads back hit information from the FIFO’s

素晴 Patch for BLAST source

- init_hw()
- invoke_hw()
For each sequence:

```bash
blastall -p blastn -i query -d database -o results
```

- **critical_code**
- **BlastNtWordFinder**

**If**
- **First_trip**
  - **false**
    - Load sequences into Input FIFO
  - **true**
    - Load "lookup_table" & Initialize hardware

**Obtain hit information from Output FIFO**

**Similarity matching against query**

**FPGA Slices**

**Linux Device Driver**

**Xilinx Vitex II Pro FPGA**
(1) On-Chip Caching of Partial Look-Up

- Since Lookup table is in Off-chip memory, IP cores take lot of clock cycles to access
- Used Available BlockRAM’s in the FPGA to cache the first column of the lookup table
- Made appropriate changes to the IP core to add BRAM’s and appropriate lookup
- Made appropriate changes to the Device Driver to load the BRAM’s with the first column of the Lookup table
(2) Parallel Design

RC-BLAST IP Core with support for partial lookup

RC-BLAST Core with support for on-chip caching of partial lookup consumes

- 800 logic slices which is 4% of the FPGA slices
- 36 BRAM’s were consumed which is 26% of available

Could replicate only 2 RC-BLAST core units with support for partial lookup

RC-BLAST IP Core with support for Partial Lookup doesn’t achieve much parallelism
(2) Parallel Design (cont’d)

RC-BLAST IP Core

- RC-BLAST Core consumes 800 logic slices which is 4% of the FPGA slices
- Rest of the Base System consumes 48% of the FPGA slices
- Replicated up to 8 RC-BLAST core units

**RC-BLAST IP Core is more efficient**
Results: Cost Analysis

- Compared cost of number of units that can fit in FPGA's vs Cluster of processors

- Considered an average cluster node to be 450$

- v2p30 - Present FPGA in the Target Hardware

- v2p100 - One of the largest FPGA's

Clearly FPGA's give more units in terms of cost

* Results are based on RC-BLAST IP core without support for Partial Lookup
(Expected) Results: Power Dissipation

- Power is definitely a big concern nowadays
- Compared power dissipation of existing processors with RC-BLAST units running in the FPGA

Clearly, we would expect FPGA's dissipate lot lesser power than any of the existing processors available.
Results: Speedup

- **Number of Lookup’s / second** considered as a measure to calculate speedup

- RC-BLAST runs $10 \times$ faster than just the Software version of BLAST on the FPGA

- RC-BLAST almost same as conventional processors

- With more RC-BLAST units, we can definitely achieve more speedup
Results: Portability

- Portability
  - Added RC-BLAST IP Core as a User IP core to ML-403 Base system which hosts a Xilinx Virtex IV FPGA
  - Synthesized to a bit stream without any additional effort
  - As technology progresses abiding Moore’s Law gives more transistors in a die-
    - For FPGA’s it is more logic
    - For processor’s it is more cache
Conclusion

✔ Achieved all the design objectives

❑ A design on Reconfigurable Computing resources
❑ Achieved scalability of the design with regards to low cost, low power and speedup
❑ Developed a portable design
Conclusions

Is a scalable and cost-effective Reconfigurable Implementation of BLAST feasible which can aid scientists and biologists in a more productive way? **Yes**
Contributions

Along with answering the thesis question ..... 

- Identified the I/O bound problem of BLAST by performing various experiments
- Ported Linux to a Platform FPGA
  - Developed Cross compiler for PPC405 platform
  - Developed Device Drivers to access hardware from Linux
- Design Extensions
  - Developed a system where parallelism can be obtained in BLAST
Future Work

- Change the existing OPB Bus Interface to a PLB Bus interface or an APU interface for the RC-BLAST IP Core
- Port RC-BLAST to a Cray XD-1 or a SGI - Altix
- Port RC-BLAST to a FPGA based Cluster
- Develop software to handle multiple units in an FPGA
- Develop RC-BLAST with multiple units without the need for Bus-Arbitration
Thank You

Questions !!!!
List of Slides

Outline .................................................. 2
Motivation .................................................. 3
BLAST ..................................................... 4
BLAST Similarity Search ......................... 5
Problem Size vs Processor and I/O Performance 6
Problem and Possible Solutions ............. 7
Thesis Question ....................................... 8
Background .............................................. 9
Reconfigurable Computing .................... 10
Target Hardware ................................. 11
Standard Linux Virtex II Pro FPGA Configuration 12
Basic Idea of Similarity Search ............ 13
BLAST Heuristic ....................................... 14
BLAST Example ........................................ 15
Previous RC and BLAST Work ................ 16
Analysis ..................................................... 17
Initial Experiments with BLAST .......... 18
Initial Experiments with BLAST (cont’d) ...... 19
Profiling BLAST ................................. 20
RC Implementation ........................... 21