The State of Parabix: A Research Agenda

Robert D. Cameron

School of Computing Science
Simon Fraser University

January 15, 2018

Outline

Outline

Parabix Technology

Parabix Concept

- Parabix employs *bitwise data parallelism* to achieve high-performance text processing.
- XML parsing [HPCA 2012].
- Regular expression matching [PACT 2014].
- Process 128 bytes at a time using 128-bit SSE2 registers on all Intel/AMD 64-bit processors.
- Process 256 bytes at a time using 256-bit AVX2 technology.

Parabix Technology

Parabix Concept

- Parabix employs *bitwise data parallelism* to achieve high-performance text processing.
- XML parsing [HPCA 2012].
- Regular expression matching [PACT 2014].
- Process 128 bytes at a time using 128-bit SSE2 registers on all Intel/AMD 64-bit processors.
- Process 256 bytes at a time using 256-bit AVX2 technology.

Parabix Regular Expression Software

- `icgrep 1.0` employs Parabix methods in a full Unicode Level 1 "grep" search tool [IUC39, ICAPP2015].
- Gigabyte/sec regular expression search.

Recent Advances

Parabix Toolchain

- 100% dynamic compilation to LLVM IR.
- Dynamic processor detection for AVX2.
- Can target NVPTX back end (Nvidia GPUs).
- Application construction using stream-processing kernels.
- Multicore processing using segmented pipeline parallelism.

Recent Advances

Parabix Toolchain

- 100% dynamic compilation to LLVM IR.
- Dynamic processor detection for AVX2.
- Can target NVPTX back end (Nvidia GPUs).
- Application construction using stream-processing kernels.
- Multicore processing using segmented pipeline parallelism.

Regular Expression Improvements

- Star-Normal Form
- Log2 Bounded Repetitions

Technology Roadmap: Parabix OS

Parabix Shell plus Core Utilities

- Parabix versions of grep, sed, awk, cut, wc, head, tail, join, ...
- Parabix shell: dynamic pipelining using pipeline parallelism.
- Goal: high performance OS for big data applications.
- Compression, transcoding, etc., built-in.
- Design for use with Linux or Darwin kernel.

Technology Roadmap: Parabix OS

Parabix Shell plus Core Utilities

- Parabix versions of grep, sed, awk, cut, wc, head, tail, join, ...
- Parabix shell: dynamic pipelining using pipeline parallelism.
- Goal: high performance OS for big data applications.
- Compression, transcoding, etc., built-in.
- Design for use with Linux or Darwin kernel.

Parabix Components for Unicode

- Integrate high-level Unicode awareness into all core utilities.
- Unicode properties and regular expression support throughout.

Parabix Languages and Compilers

Languages: Current Status

- Grammars: regexps, character classes, Unicode properties.
- Pablo stream language: operations on arbitrary-length bit streams.
- LLVM IR: high-level assembly language for stream processing *kernels*.
- Pipeline protolanguage: mapping stream sets to buffers, composing kernels, scheduling computations.

Parabix Languages and Compilers

Languages: Current Status

- Grammars: regexps, character classes, Unicode properties.
- Pablo stream language: operations on arbitrary-length bit streams.
- LLVM IR: high-level assembly language for stream processing *kernels*.
- Pipeline protolanguage: mapping stream sets to buffers, composing kernels, scheduling computations.

Compilers

- Character class compiler: generate Pablo code.
- Unicode property compiler: Pablo code for any Unicode property.
- Regexp compiler: produce Pablo code for any regular expression.
- Pablo compiler: produce Pablo Kernels in LLVM IR.
- Kernel Pipeline Compilers: produce IR from a chain of kernels.

Outline

1. Parabix Overview
2. Kernels
3. Pipeline Implementation and Compilation
4. More Kernels

Kernels

Kernel Structure

- Kernels are computational abstractions for text stream processing.
- Kernels process input stream sets, producing output stream sets.

Kernels

Kernel Structure

- Kernels are computational abstractions for text stream processing.
- Kernels process input stream sets, producing output stream sets.

Transposition Kernel

- Input: $1 \times$ `i8`: a single stream of 8-bit code units (e.g., UTF-8).
- Output: $8 \times$ `i1`: a set 8 of parallel bit streams (basis bit streams).

Kernels

Kernel Structure

- Kernels are computational abstractions for text stream processing.
- Kernels process input stream sets, producing output stream sets.

Transposition Kernel

- Input: $1 \times$ `i8`: a single stream of 8-bit code units (e.g., UTF-8).
- Output: $8 \times$ `i1`: a set 8 of parallel bit streams (basis bit streams).

Transposition Subkernels

- Transposition can actually be divided into 3 stages.
- Stage 1: $1 \times$ `i8`: to $2 \times$ `i4` (2 streams of nybbles).
- Stage 2: $2 \times$ `i4`: to $4 \times$ `i2` (4 streams of bit-pairs).
- Stage 3: $4 \times$ `i2`: to $8 \times$ `i1` (basis bit streams).

Regular Expression Kernels

Character Class Kernels

- Kernel for the character classes of a regexp: e.g., `a[0-9]*[z9]`
- Input: $8 \times$ `i1`: the 8 basis bit streams.
- Output: $3 \times$ `i1`: 3 bit streams for `[a]`, `[0-9]`, `[z9]`
- Dynamically generated by the Parabix Character Class compiler.

Regular Expression Kernels

Character Class Kernels

- Kernel for the character classes of a regexp: e.g., `a[0-9]*[z9]`
- Input: $8 \times$ `i1`: the 8 basis bit streams.
- Output: $3 \times$ `i1`: 3 bit streams for `[a]`, `[0-9]`, `[z9]`
- Dynamically generated by the Parabix Character Class compiler.

Matching Logic Kernels

- Kernel for the matching logic: e.g., `a[0-9]*[z9]`
- Input: $3 \times$ `i1`: character class streams
- Output: $1 \times$ `i1`: a bit stream of matches found.
- Dynamically generated by the Parabix Regular Expression compiler.
- Future: generate begin/end pairs for substring capture.

Modular icgrep Kernels

Line Break Kernel

- Kernel for Unicode line breaks
- Input: $8 \times$ `i1`: the 8 basis bit streams.
- Output: $1 \times$ `i1`: line breaks for any of LF, CR, CRLF, LS, PS, $\cdots$
- Currently implemented within regexp compiler: should factor out.

Modular icgrep Kernels

Line Break Kernel

- Kernel for Unicode line breaks
- Input: $8 \times$ `i1`: the 8 basis bit streams.
- Output: $1 \times$ `i1`: line breaks for any of LF, CR, CRLF, LS, PS, $\cdots$
- Currently implemented within regexp compiler: should factor out.

Match Scanning Kernel

- Kernel to generate matched lines.
- Three inputs:
  - $1 \times$ `i8`: source byte stream
  - $1 \times$ `i1`: matches bit stream
  - $1 \times$ `i1`: line break bit stream
- Output: $1 \times$ `i8` matched line output stream.

Kernel Composition: Pipelines

Kernels + StreamSets = Programs

- Name the stream sets used as inputs and outputs to each kernel.
- Compose a program as a sequence of kernels.

Kernel Composition: Pipelines

Kernels + StreamSets = Programs

- Name the stream sets used as inputs and outputs to each kernel.
- Compose a program as a sequence of kernels.

A 7-Stage icgrep Program

```
         ByteData = MMapSource(FileName)
        BasisBits = Transpose(ByteData)
         LineEnds = UnicodeLineBreaks(BasiBits)
CharacterClasses = CC_compiler<regexp>(BasisBits)
          Matches = RE_compiler<regexp>(CharacterClasses)
     MatchedLines = MatchScanner(ByteData, LineEnds, Matches)
                    StdoutSink(MatchedLines)
```

Outline

Stream Sets and Buffers

Stream Sets

- A stream set type is of the form $N \times \texttt{iK}$
- N streams of items, each item of width K bits
- All streams in a set are of the same length L (may be unknown).

Stream Sets and Buffers

Stream Sets

- A stream set type is of the form $N \times \texttt{iK}$
- N streams of items, each item of width K bits
- All streams in a set are of the same length L (may be unknown).

Buffers

- Buffers are storage for *segments* of stream sets.
- All of the streams of a set are stored in a single buffer.
- Stream sets are stored block-at-a-time (significant for $N > 1$)
- Different buffering strategies.
  - Full stream length (mmap)
  - Fixed length circular buffer.
  - Fixed length buffer with copyback.
  - Expanding buffer (expands as needed).

Pipeline Compilation

Pipeline Requirements

- Buffers are allocated for all streams.
- Internal states allocated for all kernels.
- Kernels are compiled to process data in defined buffers.

Pipeline Compilation

Pipeline Requirements

- Buffers are allocated for all streams.
- Internal states allocated for all kernels.
- Kernels are compiled to process data in defined buffers.

Pipeline Compiler Status

- Four basic compilers have been built:
  - Sequential single-core pipeline.
  - Multithread pipeline (pure pipeline parallelism).
  - Segmented pipeline parallel (threads execute alternating segments).
  - GPU-CPU hybrid pipeline.
- All compilers need work!!

Experimental Pipeline Compilers

Pipeline Parallel Compiler

- Each kernel is compiled to a separate thread function.
- Lock-free synchronization through monotonic positions.
- Balance between pipeline stages is problematic.
- Retired.

Experimental Pipeline Compilers

Pipeline Parallel Compiler

- Each kernel is compiled to a separate thread function.
- Lock-free synchronization through monotonic positions.
- Balance between pipeline stages is problematic.
- Retired.

NVPTX Pipeline Compiler

- Kernels compiled to PTX code to run on NVidia GPUs.
- Can now compile first 4 icgrep stages to GPU.
- Currently only a single workgroup of 64 threads: 4096 position SIMT.
- MatchedLineScanner compiles to CPU.
- Further work: combined GPU/CPU compilers.

Segmented Pipeline Parallelism

Combined Data and Pipeline Parallelism

- Input divided into logical segments.
- Allocate segments to P cores in round robin fashion.
- Core i responsible for all segments n such that $n \mod P = i$.
- Each core executes a full pipeline for its segment.
- For any pipeline stage s and segment $i+1$, core $(i+1) \mod P$ can proceed as soon as core $i \mod P$ completes stage i.
- Workload balanced between cores as long as no stage requires more than $1/P$ of the total time to process a segment.
- Now the default for all applications.

Compiler Issues

Buffer Allocation and Management

- Current compilers too naive: assume a common segment size across kernels.
- Workable for some applications, e.g., icgrep.
- In general, buffer sizing and discipline depends on kernel properties.
- Kernels may use lookahead on a input stream set.
  - Input buffer must have additional room for lookahead blocks.
  - Preceding kernels must process ahead of their lookahead-dependent kernels.
  - Circular buffering must be used.
- Kernels may have an expansion factor, e.g. 4/3 expansion for radix64 kernels.
- Kernels may have variable-length output, e.g., u8u16.

Compiler Issues

Kernel Contracts

- Kernels must be implemented to respect contractual requirements of the pipeline compilers.
- Report number of produced items in each output stream set.
- Report consumed positions for each input stream set.
- Declare and adhere to stream set attributes.
  - FixedRate attribute: - automate processing rate
  - BoundedRate: sets limit on buffer size
  - Lookahead attribute

Compiler Issues

Kernel Contracts

- Kernels must be implemented to respect contractual requirements of the pipeline compilers.
- Report number of produced items in each output stream set.
- Report consumed positions for each input stream set.
- Declare and adhere to stream set attributes.
    - FixedRate attribute: - automate processing rate
    - BoundedRate: sets limit on buffer size
    - Lookahead attribute

Synchronization

- Multithreading requires appropriate synchronization.
- T-thread segment parallel compiler currently has a race condition.
    - Buffer output struct may be modified by producer in thread $t+1$ before all consumers in thread t have accessed it.

Outline

Parallel Deletion Kernels

Bit Stream Compression Kernel

- Two inputs:
  - `Nxi1`: bit streams to compress
  - $1 \times$ `i1`: deletion mask stream
- Output: `Nxi1`: compressed output streams
- Example:

input[1]	10101000101010101011
input[2]	11100111100000110001
deletion mask	00111000001101000110
output[1]	100001011011
output[2]	111111001101

- Provides a general approach to stream filtering.

Transcoding Kernels

UTF-8 to UTF-16 Logic Kernel

- Input: $8 \times \texttt{i1}$: the 8 basis bit streams.
- Three outputs:
  - $16 \times \texttt{i1}$: UTF-16 parallel bit streams
  - $1 \times \texttt{i1}$: deletion mask stream
  - $1 \times \texttt{i1}$: UTF-8 error stream
- Only one logical output code unit position for 2 or 3 byte UTF-8 sequence, 2 positions for 4-byte sequences.
- Deletion mask marks positions to be removed from output stream.