SFU SIMON FRASER UNIVERSITY
THINKING OF THE WORLD

Parabix : Boosting the Efficiency of Text Processing on Commodity Processors

Dan Lin, Nigel Medforth, Kenneth S. Herdy, Arrvindh Shriraman, Rob Cameron

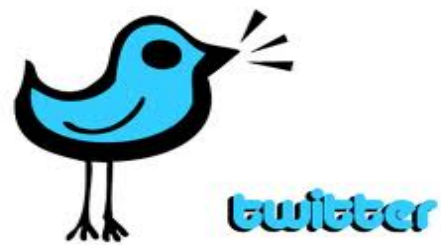
School of Computing Science, Simon Fraser University

Text Processing is Important

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<current_observation>
    <credit>NOAA's National Weather Service</credit>
    <location>New Orleans, Naval Air Station, LA</location>
    <latitude>29.833</latitude>
    <longitude>-90.017</longitude>
    <weather>Overcast</weather>
    <temperature_string>74.0 F (23.3 C)</temperature_string>
    <temp_f>74.0</temp_f>
    <temp_c>23.3</temp_c>
    <relative_humidity>85</relative_humidity>
    <wind_string>Southwest at 17.3 MPH (15 KT)</wind_string>
    <wind_dir>Southwest</wind_dir>
    <visibility_mi>9.00</visibility_mi>
</current_observation>
```

Text Processing is Important

250 million messages per day



Tweets

Text Processing is Important

Text Processing is Important

Huge amount of text every second !

Mobile Text

Emails

Tweets

Text Processing is Hard

- The "thirteenth dwarf" [illegible] nines)
  - Hardest dwarf to parallelize [illegible]

  [Berkeley "Parallel Computing Landscape" report]

"Nothing helps!"

- Large state machines for in[illegible]
  - Irregular memory access.

Cache Misses

- Parsing variable length str[illegible] r data
  - Branches in the code.

Branch Mispredictions

Simple Example of Parsing

Text input **<name> txt <error] <err)**

1. Locate “<“
2. Scan through alphabet from “<“ to match “>”.
3. Report error positions for mismatching.

Traditional method to do step 1
- ask each byte: Are you “<“?

Conventional XML Parser (Xerces)

```
while (gotData)
{ MLSize orgReader;
  co                    oken = senseNextToken(orgReader);
                        CharData) {
                        aBuf);
    co                  ; }
  else if (curToken == Token_EOF) {
    if (!fElemStack.isEmpty()) {
    continue;
  }
  ...
switch(curToken)
case                    DSection(); break;
case                    mment(); break;
case                    EndTag(gotData); break;
                        ); break;
case Token_Char_Tag: isNoNamespaces ?
scanStartTagNS(gotData) : scanStartTag(gotData); break;
default: fReaderMgr.skipToChar(chOpenAngle);
}
```

Byte-at-a-time

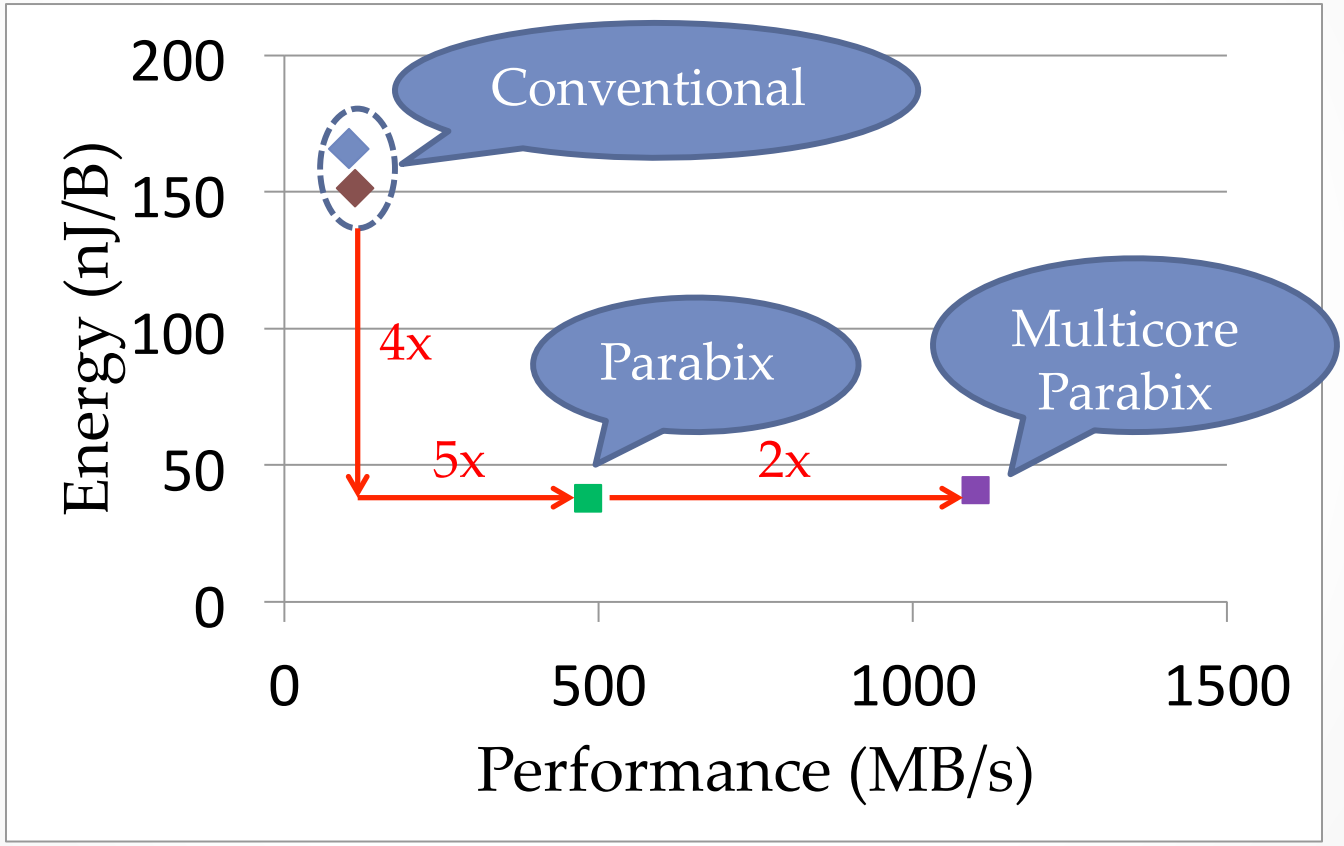
Highly Inefficient!

13 branches per input byte

Our Technology : Parabix

- Highly parallel using bitwise SIMD
  - Byte streams restructured to parallel bit streams.
  - 128 bytes at a time with 128-bit SIMD (SSE)
    - Or more, depending on architecture.
  - Almost branch free.
  - Streaming, cache-friendly model.
- Programming support
  - Character Class Compiler (CCC)
    - Mark occurrence of character classes (e.g. [<]).
  - Parallel Block Compiler (Pablo)
    - Convert Python (unbounded bitstreams) to C++ using SIMD
  - a portable SIMD library
    - Supported many architectures

Our Results: XML Parsing



Outline

- Parabix Framework:
  - Parallel Bitstream Technology
  - Parabix toolkit:
    - CCC : Character Class Compiler
    - Pablo : Parallel Block Compiler
    - Portable SIMD library
- XML Parsing with Parabix
- Performance and Energy Evaluation
- Multithreaded/Multicore Parabix

Outline

- Parabix Framework:
  - ***Parallel Bitstream Technology***
  - Parabix toolkit:
    - CCC : Character Class Compiler
    - Pablo : Parallel Block Compiler
    - Portable SIMD library
- XML Parsing with Parabix
- Performance and Energy Evaluation
- Multithreaded/Multicore Parabix

What are Parallel Bitstreams?

```
     b         7         <         A         <
01100010 00110111 00111100 01000001 00111100
```

b_0
b_1
b_2
b_3
b_4
b_5
b_6
b_7

Variables that can hold unbounded bitstreams

Given a byte-oriented character stream T, e.g., "`b7<A<`".
Transpose to 8 parallel bit streams b_0, b_1, ..., b_7.

What are Parallel Bitstreams?

	b	7	<	A	<
	01100010	00110111	00111100	01000001	00111100
b_0	0	0	0	0	0
b_1					
b_2					
b_3					
b_4					
b_5					
b_6					
b_7					

Given a byte-oriented character stream T, e.g., "`b7<A<`".
Transpose to 8 parallel bit streams b_0, b_1, ..., b_7.

What are Parallel Bitstreams?

	b	7	<	A	<
	01100010	00110111	00111100	01000001	00111100
b_0	0	0	0	0	0
b_1	1	0	0	1	0
b_2					
b_3					
b_4					
b_5					
b_6					
b_7					

Given a byte-oriented character stream T, e.g., "`b7<A<`".
Transpose to 8 parallel bit streams b_0, b_1, ..., b_7.

What are Parallel Bitstreams?

	b	7	<	A	<
	01100010	00110111	00111100	01000001	00111100
b_0	0	0	0	0	0
b_1	1	0	0	1	0
b_2	1	1	1	0	1
b_3	0	1	1	0	1
b_4	0	0	1	0	1
b_5	0	1	1	0	1
b_6	1	1	0	0	0
b_7	0	1	0	1	1

Given a byte-oriented character stream T, e.g., "`b7<A<`".

Transpose to 8 parallel bit streams b_0, b_1, ..., b_7.

Character Bitstream Classification

	b	7	<	A	<
	01100010	00110111	00111100	01000001	00111100
b_0	0	0	0	0	0
b_1	1	0	0	1	0
b_2	1	1	1	0	1
b_3	0	1	1	0	1
b_4	0	0	1	0	1
b_5	0	1	1	0	1
b_6	1	1	0	0	0
b_7	0	1	0	1	1

Now calculate the LAngle bitstream in parallel.

$[<] = \neg b_0 \wedge \neg b_1 \wedge b_2 \wedge b_3 \wedge b_4 \wedge b_5 \wedge \neg b_6 \wedge \neg b_7$

<	0	0	**1**	0	**1**

Character Bitstream Classification

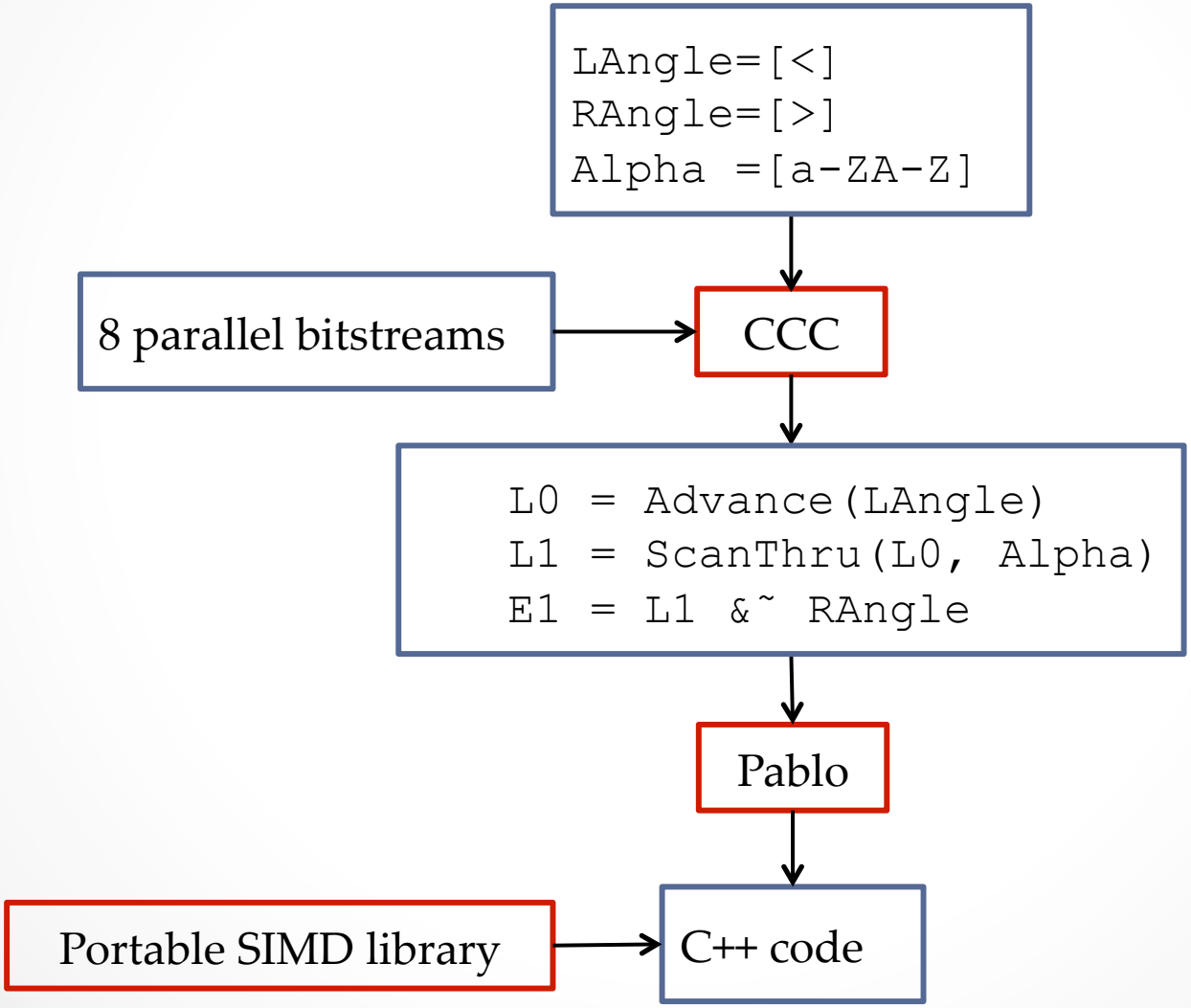
- Minimum number of operations?
  - [<] : 7 ops ← Easy
  - [<] + [>] : 10 ops ← Not so hard.
  - [<] + [>] + [a-zA-Z] : 21 ops ← Well…I can handle.

- Larger set of character classes?
  - e.g. XML parsing : about 30 character classes.

Outline

- Parabix Framework:
  - Parallel Bitstream Technology
  - ***Parabix toolkit:***
    - ***CCC : Character Class Compiler***
    - ***Pablo : Parallel Block Compiler***
    - ***Portable SIMD library***
- XML Parsing with Parabix
- Performance and Energy Evaluation
- Multithreaded/Multicore Parabix

Parabix Tool Chain



Character Class Compiler

```
LAngle = [<]
RAngle = [>]
Alpha  = [a-ZA-Z]
```

Programmer defined

```
temp1  = (bit0 | bit1)
temp2  = (bit2 & bit3)
temp3  = (temp2 &~ temp1)
temp4  = (bit4 & bit5)
temp5  = (bit6 | bit7)
temp6  = (temp4 &~ temp5)
LAngle = (temp3 & temp6)
temp7  = (bit6 &~ bit7)
temp8  = (temp4 & temp7)
RAngle = (temp3 & temp8)
temp9  = (bit6 & bit7)
temp10 = (bit5 | temp9)
temp11 = (bit4 & temp10)
temp12 = (~temp11)
temp13 = (bit4 | bit5)
temp14 = (temp13 | temp5)
temp15 = ((bit3 & temp12)|
          (~(bit3) & temp14))
temp16 = (bit1 &~ bit0)
Alpha  = (temp15 & temp16)
```

Generated by CCC !

Simple Parsing

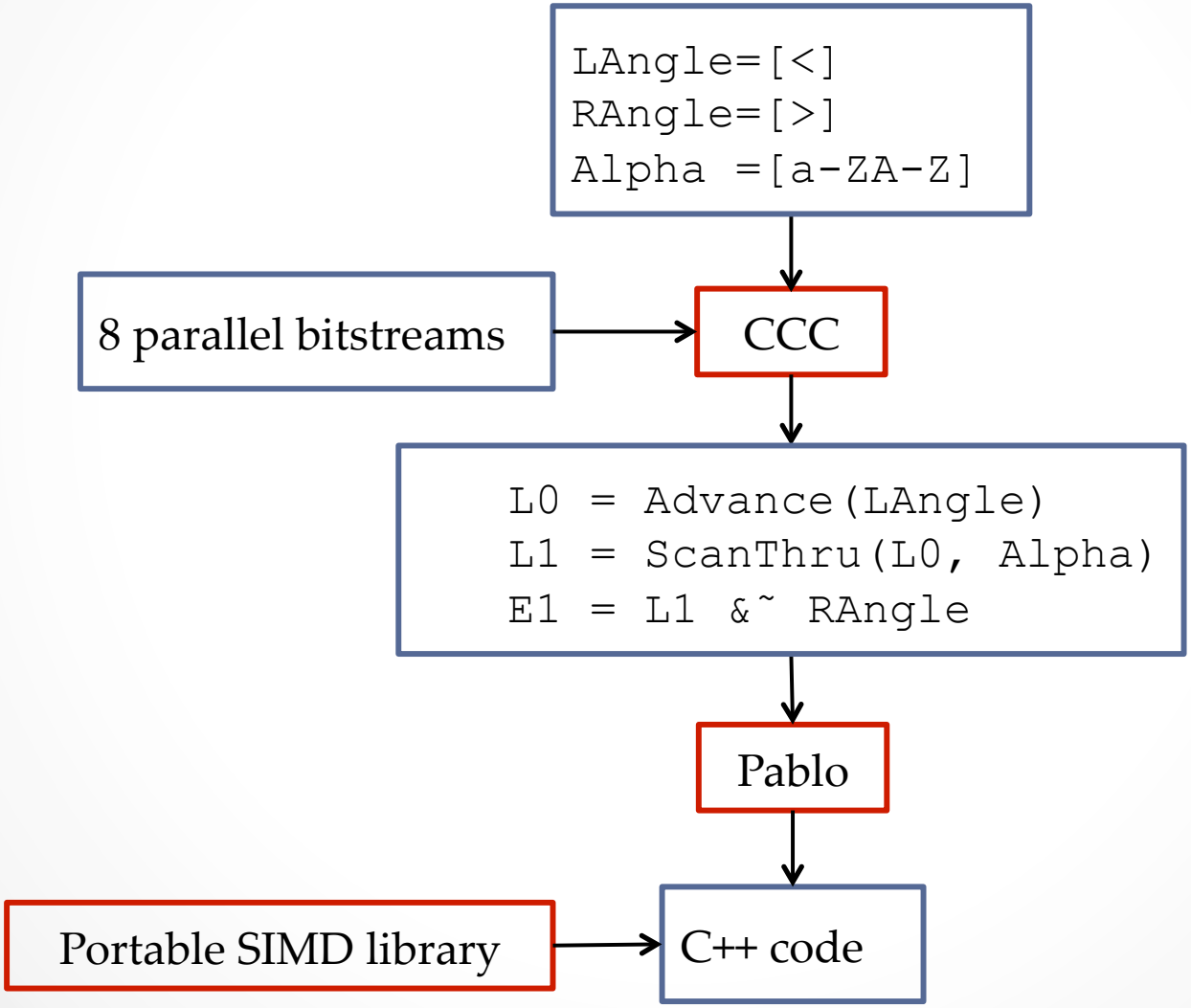
16-bit register

```
Source Text:                <name> txt <error] <err)

Alpha                       .1111..111..11111...111.
Rangle                      .....1..................
Langle                      1.........1.......1....

L0 = Advance(LAngle)        .1.........1.......1...
L1 = ScanThru(L0,Alpha)     .....1..........1.....1
E1 = L1 ∧¬Rangle            ................1.....1
```

Advance(LAngle) : LAngle>>1

ScanThru(L0, Alpha) : (L0+Alpha) ∧¬ Alpha

Parabix Tool Chain



Parallel Block Compiler (Pablo)

```
L0 = Advance(LAngle)
L1 = ScanThru(L0, Alpha)
E1 = L1 &~ RAngle
```

Programmers write in Python

```
CarryInit(carryQ, 2); }
void do_block(Lex & lex){
  BitBlock L0, L1;
  L0 = Advance_ci_co(C2, carryQ, 0);
  L1 = ScanThru_ci_co(L0, C0, carryQ, 1);
  E1 = simd_andc(L1, C1);
  CarryQ_Adjust(carryQ, 2);
}
CarryDeclare(carryQ, 2);
```

C++ Generated by Pablo !

Parallel Block Compiler (Pablo)

```
L0 = Advance(LAngle)
L1 = ScanThru(L0, Alpha)
E1 = L1 &~ RAngle
```

Programmers write in Python

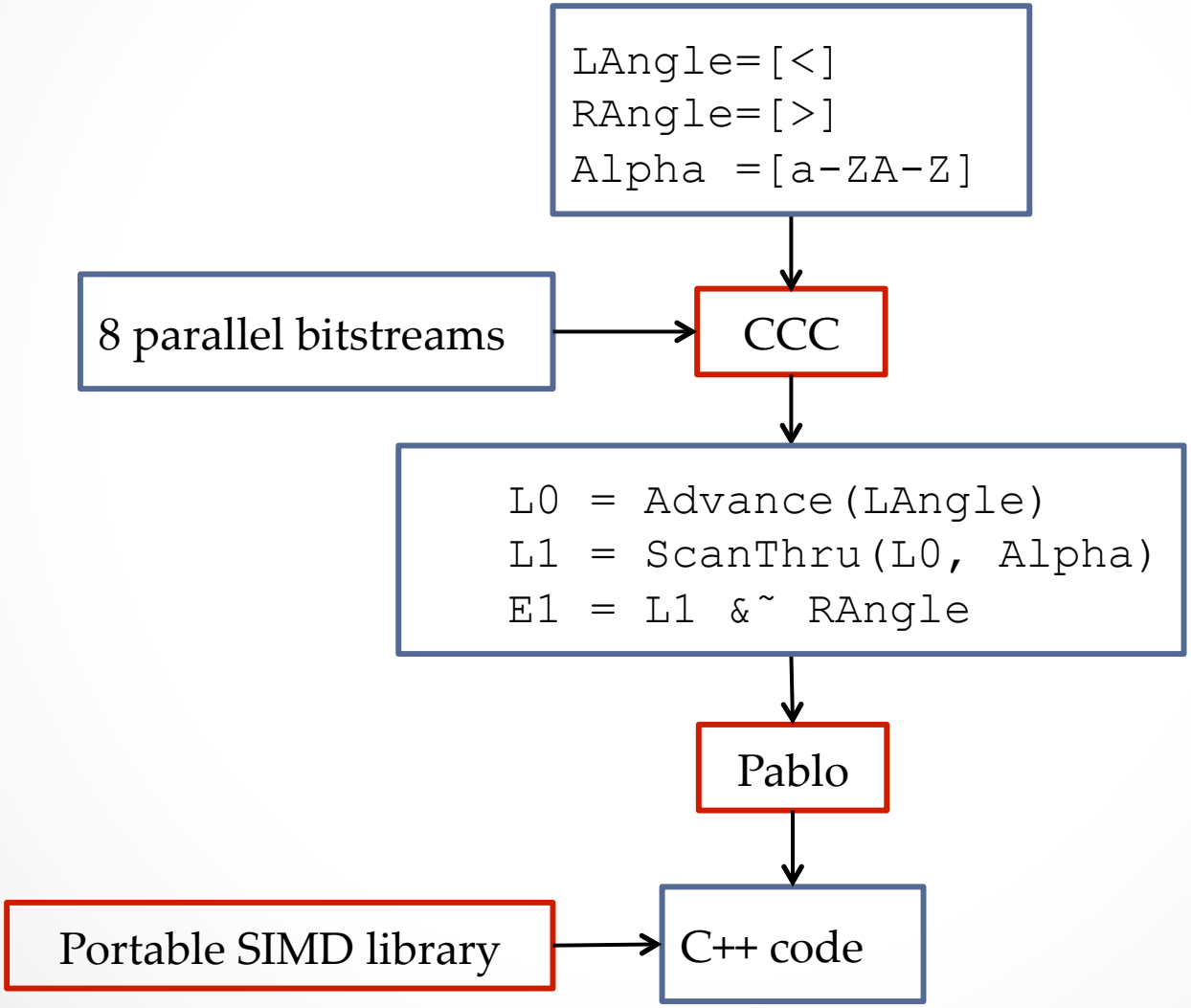
```
CarryInit(car
void do_block(Lex
  BitBlock L0, L1;
  L0 = Advance_ci_co(C2, carryQ, 0);
  L1 = ScanThru_ci_co(L0, C0, carryQ, 1);
  E1 = simd_andc( 1, C1);
  CarryQ_Adjust(ca
}
CarryDeclare
```

ci : carry in from the previous block
co: carry out to the next block

ISA-specific

C++ Generated by Pablo !

Parabix Tool Chain



Portable SIMD Library

- Our SIMD Library supports all power-of-2 field widths up to the full SIMD register width on a target machine.

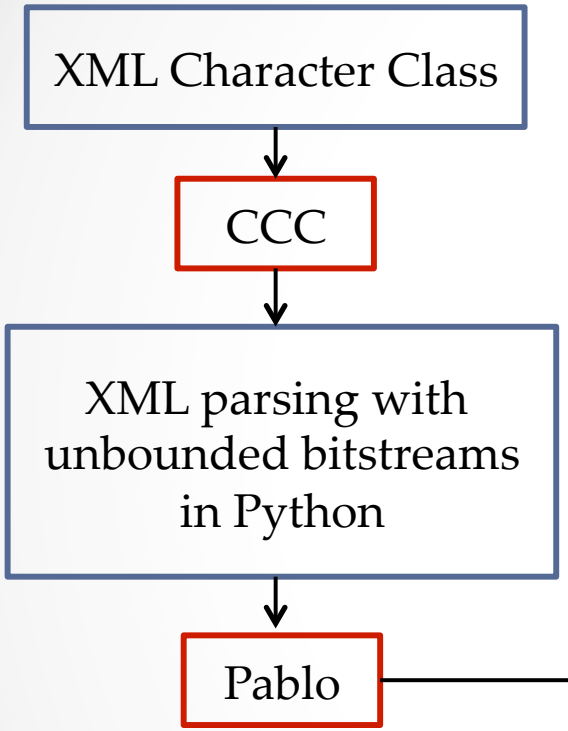
- Instruction sets supported are:
  - 128-bit Altivec
  - 128-bit SSE
  - 256-bit AVX
  - 128-bit Neon (ARM)
  - 128-bit SPU (Cell)

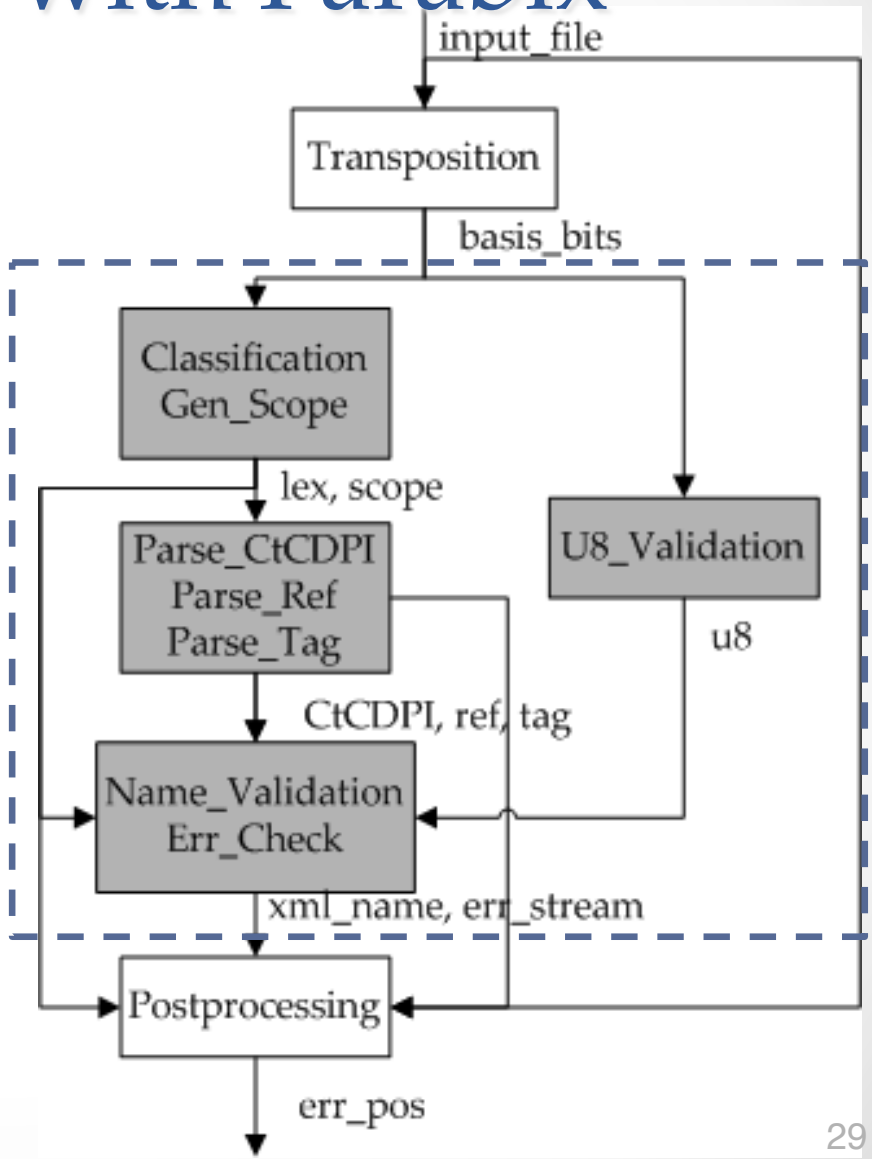
Programmers don't need to know any details of the ISAs !

Outline

- Parabix Framework:
  - Parallel Bitstream Technology – Novel application of SIMD
  - Parabix toolkit:
    - CCC : Character Class Compiler
    - Pablo : Parallel Block Compiler
    - Portable SIMD library
- ***XML Parsing with Parabix***
- ***Performance and Energy Evaluation***
- ***Multithreaded/Multicore Parabix***

XML Parsing with Parabix





Performance Study: Benchmark Files

File Name	dew	jaw	roads	po	soap
File Type	doc	doc	data	data	data
File Size (kB)	66240	7343	11584	76450	2717
Markup Density	0.07	0.13	0.57	0.76	0.87

Input Document Characteristics

Document-oriented instances often contain information intended for publication.

Data-oriented instances are typically used for the exchange of database records.

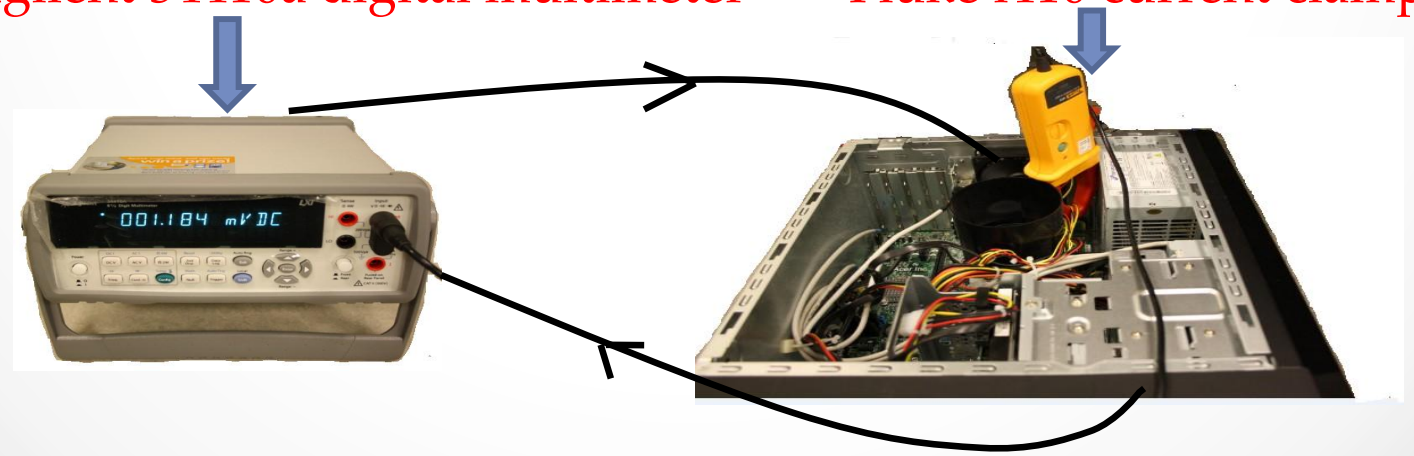
Markup Density = markup bytes / the total document size.

Experimental Set up

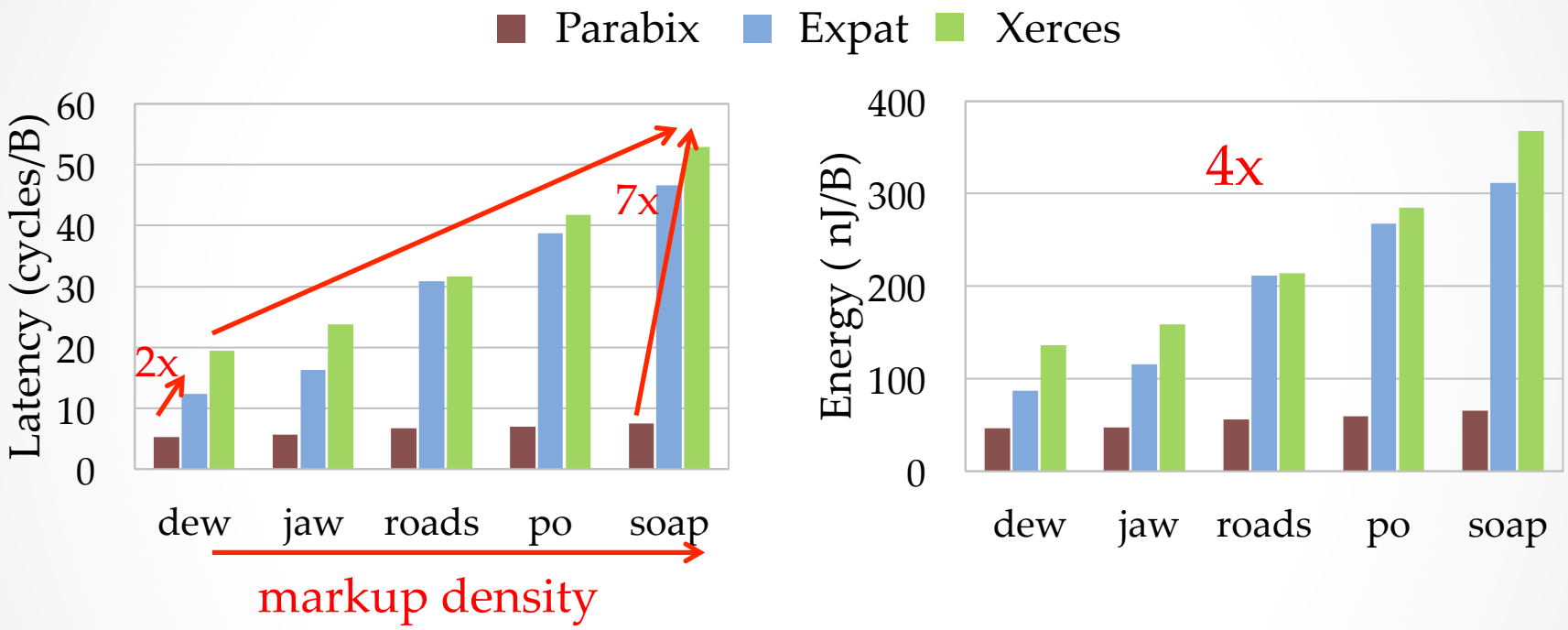
- 3 Parsers :
  - Parabix, Xerces (IBM,Apache) and EXPAT (sourceforge)
- Platforms :
  - Core2Duo, Core i3 (Baseline), SandyBridge
  - ARM A8 1Ghz. Neon ISA (Samsung Tablet)
- Metrics : Cycles / Byte. nSecond / Byte. nJoules / Byte.
  - Performance counters
- Power measurement

Agilent 34410a digital multimeter

Fluke i410 current clamp



Performance Results: Latency & Energy



Core i3

Markup density has substantial influence on the performance of traditional parsers.

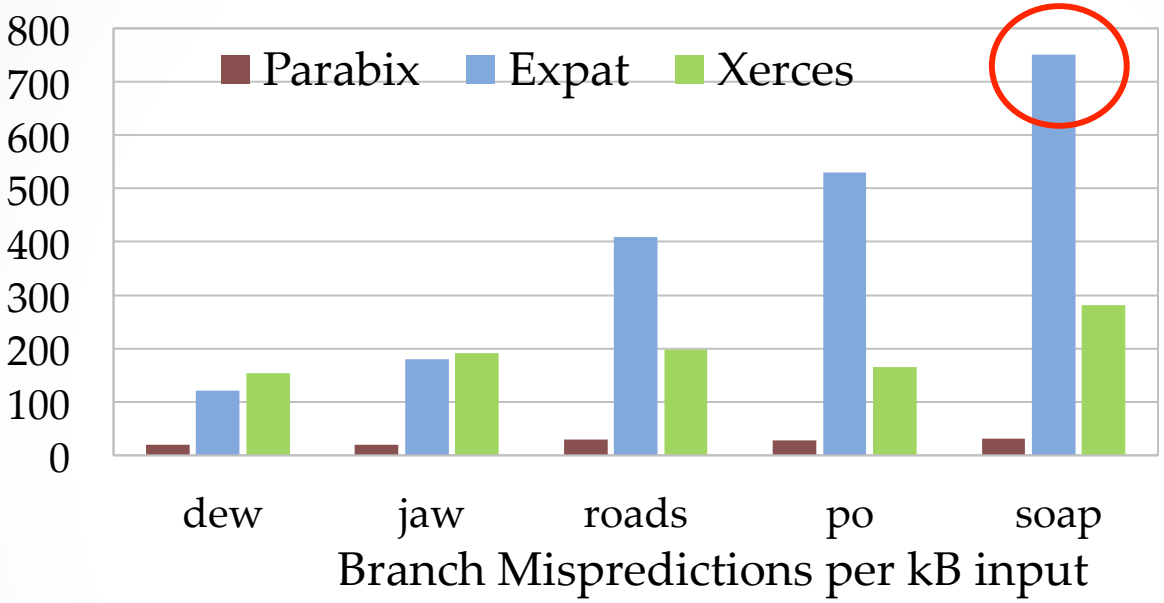
Parabix achieves more speedup on higher markup density inputs.

Parabix saves 4x energy on average.

Performance Factors

Misprediction penalty

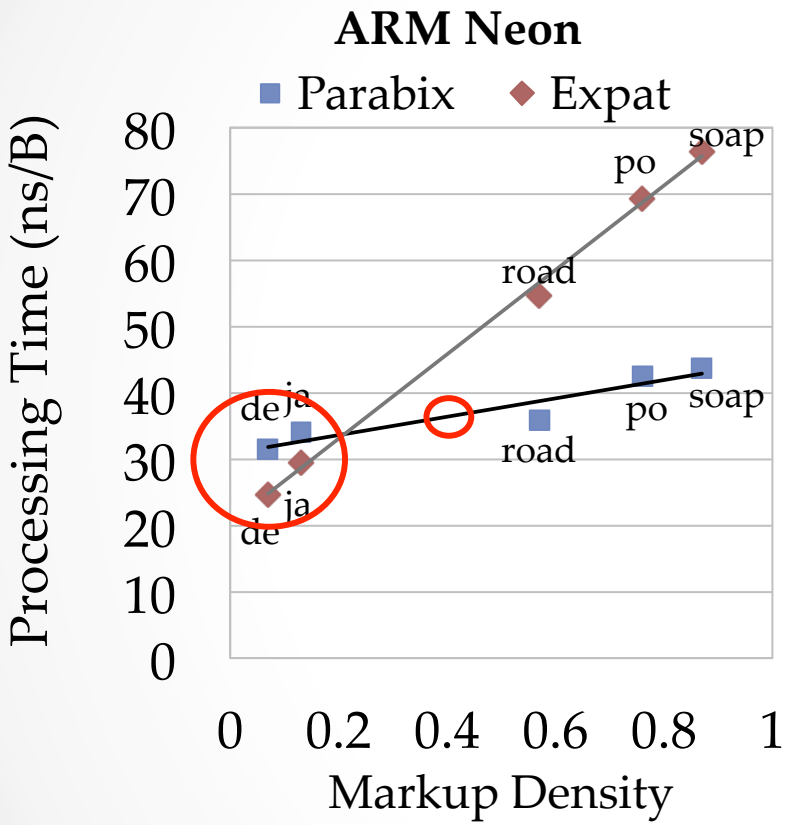
700 x 10 =
7000 cycles/kB
= processing
time of Parabx

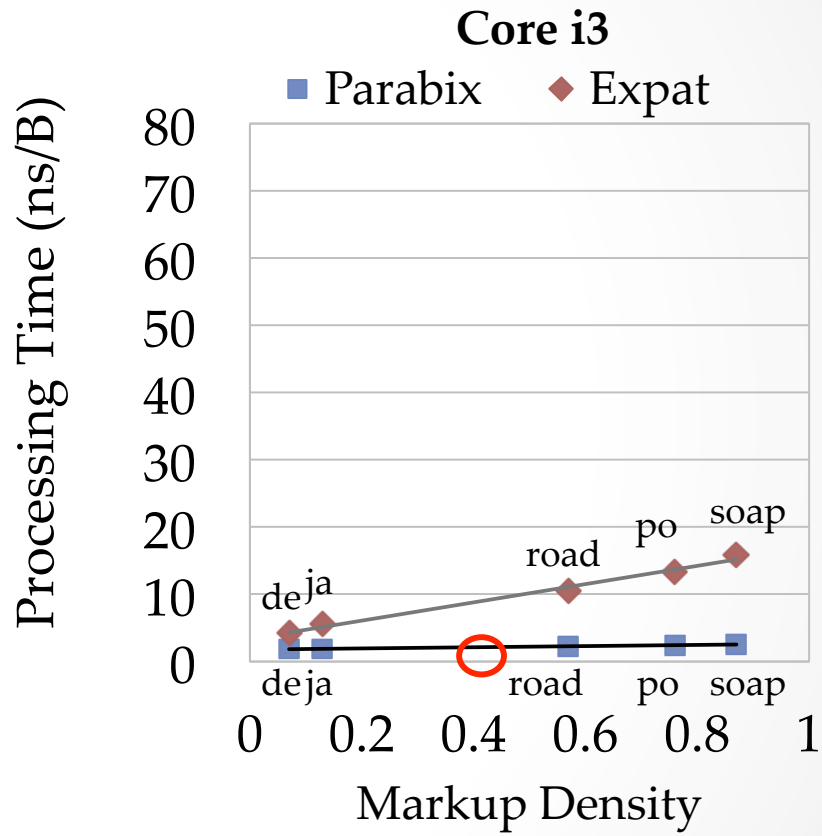


	Parabix	Expat	Xerces
L1	4.1	31.7 (↑ 8x)	104.2 (↑ 26x)
L2	0.1	12.0 (↑ 120x)	1.7 (↑ 17x)

Cache Misses per kB of input data

Parabix on Mobile (ARM Neon)



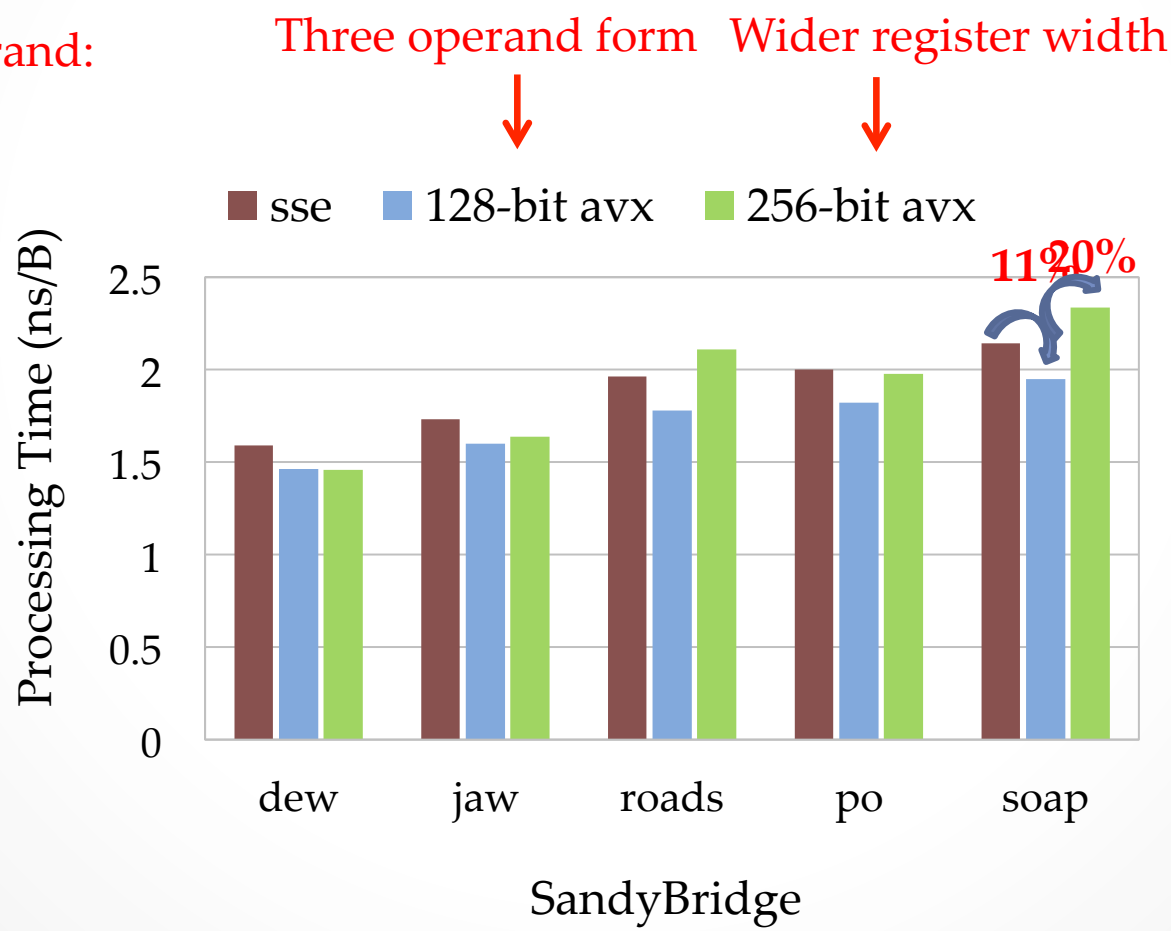


ARM Neon instructions have higher latency than Core i3's SSE

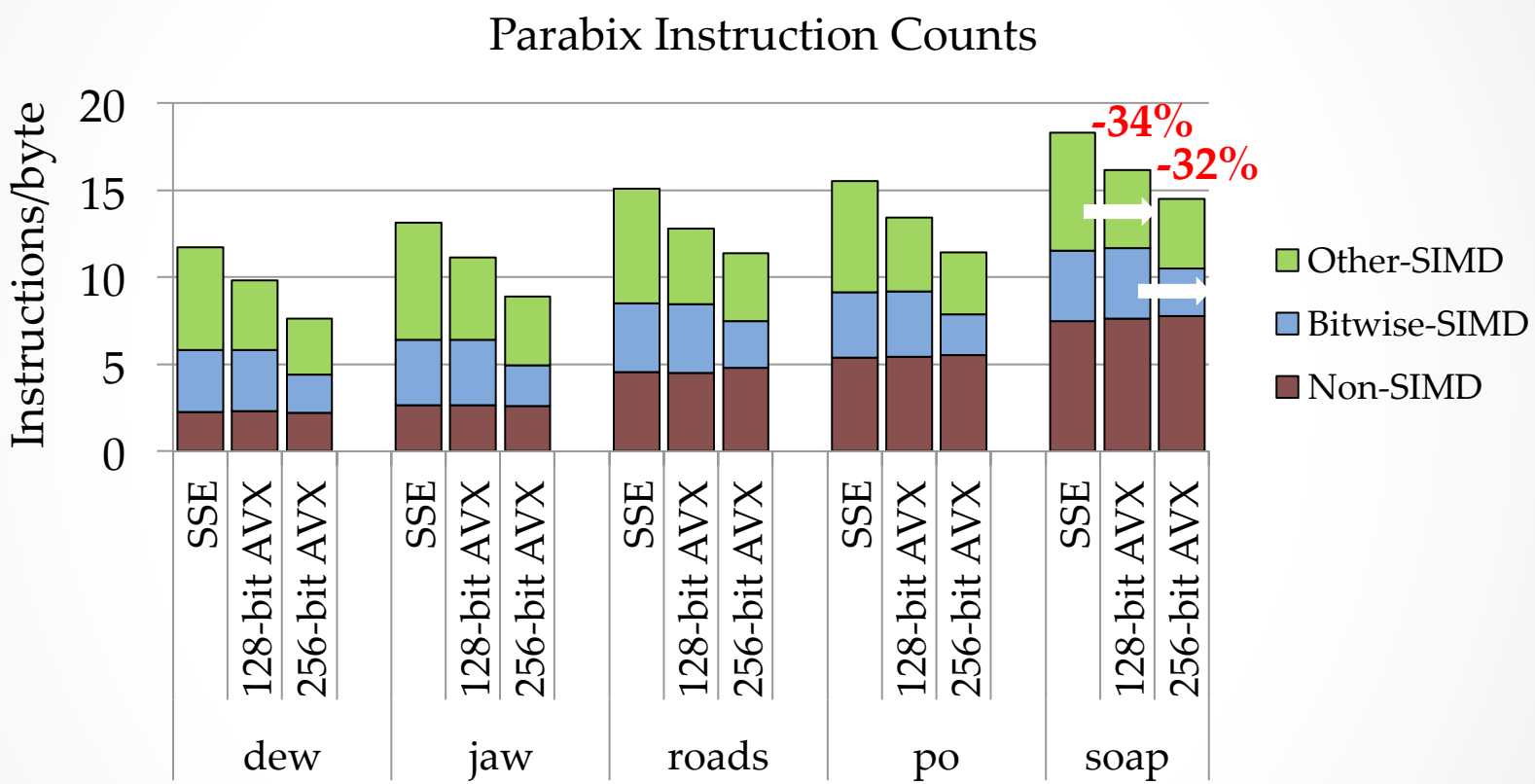
Parabix benefits files with more markup tags.

Parabix on AVX (Advanced Vector Extensions)

Two operand:
a = a + b
Three operand:
c = a + b



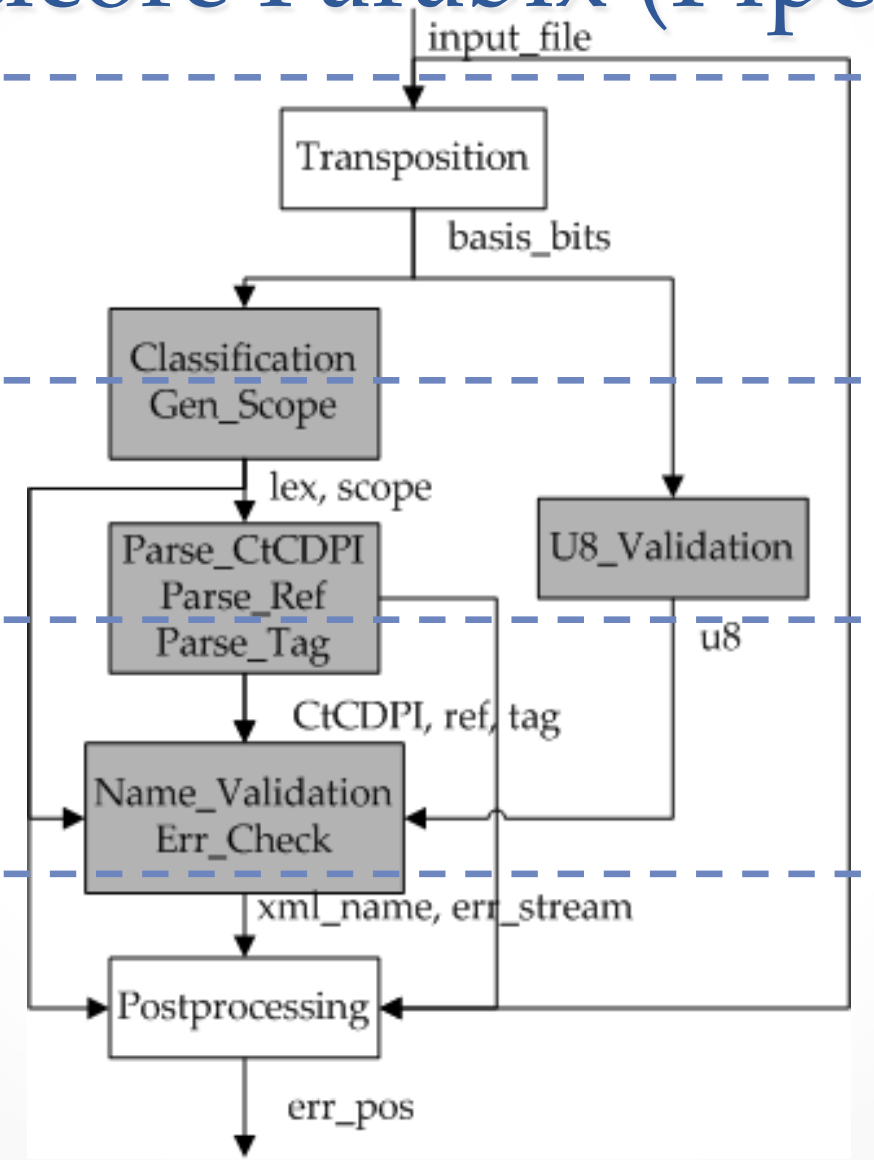
Parabix on AVX



The number of "Other SIMD" instruction reduced by using 3-op AVX.

The number of "Bitwise SIMD" instruction reduced by using 256-bit AVX.

Multicore Parabix (Pipeline)



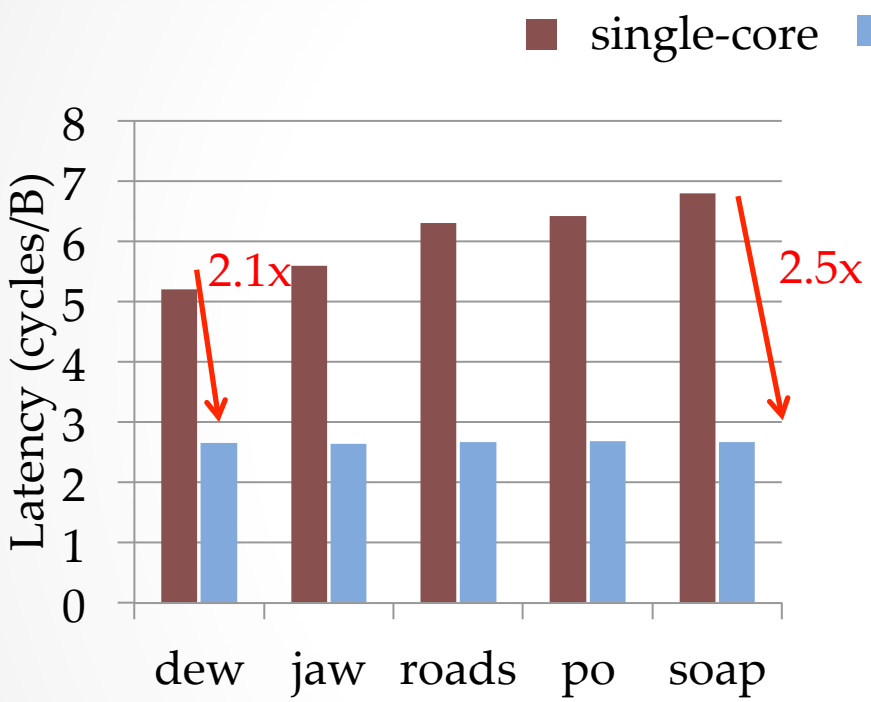
Stage 1
1.97 cycles/byte

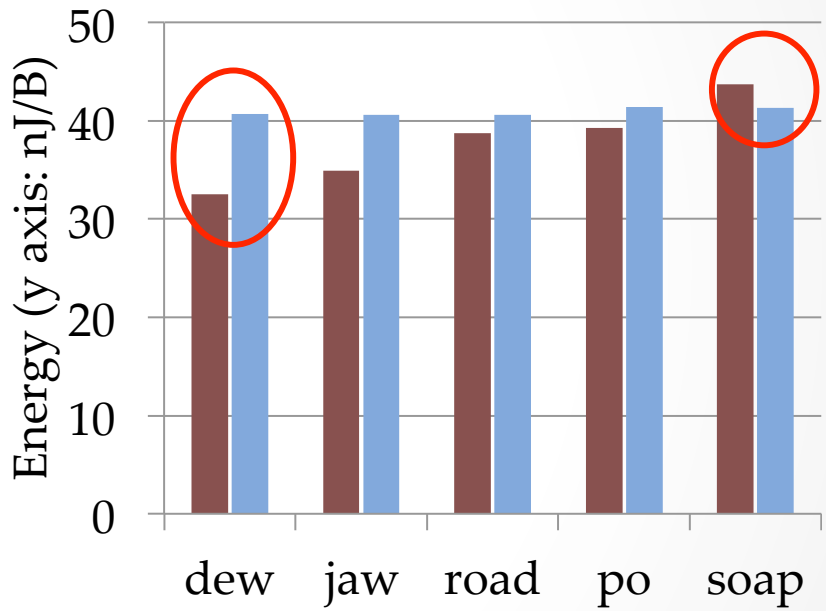
Stage 2
1.22 cycles/byte

Stage 3
2.03 cycles/byte

Stage 4
1.32 cycles/byte

Multicore Parabix (Pipeline)





4-core Parabix achieves >2x speedup over single core.

Core workloads are better balanced for high-density files.

- Better performance and energy utilization.

Summary

- Parabix: a software toolchain and runtime framework for high-performance text processing
- Parabix exploits the SIMD units found on commodity processors. Parabix XML parser:
  - 2x to 7x improvement in performance.
  - 4x improvement in energy.
- Multicore Parabix XML parser
  - further 2x improvement in performance (4 cores).
- Parabix allowed us to port text processing applications without having to change the application source.

Questions?