Performance Enhancement with Speculative Execution Based Parallelism for Processing Large-scale XML-based Application Data

Michael R. Head and Madhusudhan Govindaraju

Grid Computing Research Laboratory
Department of Computer Science
Binghamton University
http://www.cs.binghamton.edu/~{mike,mgovinda}

HPDC 2009
Thursday, June 11, 2009

Binghamton University
State University of New York

Outline

1. Introduction
   - Large XML Data
   - Ubiquity of Multi-processing Capabilities
   - SAX-based parsing
2. Parallel XML
   - PiXiMaL: Parallel Approach for Processing XML
   - Serial NFA Tests
3. Conclusions
   - Final Remarks

Binghamton University
State University of New York

Outline

1. Introduction
   - Large XML Data
   - Ubiquity of Multi-processing Capabilities
   - SAX-based parsing
2. Parallel XML
   - PiXiMaL: Parallel Approach for Processing XML
   - Serial NFA Tests
3. Conclusions
   - Final Remarks

Binghamton University
State University of New York

XML

- Text based (usually UTF-8 encoded)
- Tree structured
- Language independent
- Generalized data format

Motivation from SOAP

- Generalized RPC mechanism (supports other models, too)
- Broad industrial support
- Web Services on the Grid
  - OGSA: Open Grid Services Architecture
  - WSRF: Web Services Resource Framework
- At bottom, SOAP depends on XML

Importance of High Performance XML Processors

- Becoming standard for many scientific datasets
  - HapMap - mapping genes
  - Protein Sequencing
  - NASA astronomical data
  - Many more instances

XML Performance Limitations

- Compared to ``legacy'' formats
  - Text-based
    - Lacks any ``header blocks'' (ex. TCP headers), so must scan every character to tokenize
    - Numeric types take more space and conversion time
  - Lacks indexing
    - Unable to quickly skip over fixed-length records

Limitations of XML

- Poor CPU and space efficiency when processing scientific data with mostly numeric data (Chiu et al 2002)
- Features such as nested namespace shortcuts don't scale well with deep hierarchies
  - May be found in documents aggregating and nesting data from disparate sources
- Character stream oriented (not record oriented): initial parse inherently serial
- Still ultimately useful for sharing data divorced of its application

Explosion of Data

- Enormous increase in data from sensors, satellites, experiments, and simulations
- Use of XML to store these data is also on the rise
- XML is in use in ways it was never really intended (GB and large size files)

Outline

1. Introduction
   - Large XML Data
   - Ubiquity of Multi-processing Capabilities
   - SAX-based parsing

2. Parallel XML
   - Piximal: Parallel Approach for Processing XML
   - Serial NFA Tests

3. Conclusions
   - Final Remarks

Binghamton University
State University of New York

Prevalence of Parallel Machines

- All new high end and mid range CPUs for desktop- and laptop-class computers have at least two cores
- The future of AMD and Intel performance lies in increases in the number of cores

- Despite extant SMP machines, many classes of software applications remain single threaded

XML and Multi-Core

- Most string parsing techniques rely on a serial scanning process
- **Challenge:** Existing (singly-threaded) XML parsers are already very efficient (Zhang et al 2006)

Outline

1. Introduction
   - Large XML Data
   - Ubiquity of Multi-processing Capabilities
   - SAX-based parsing
2. Parallel XML
   - PiXiMaL: Parallel Approach for Processing XML
   - Serial NFA Tests
3. Conclusions
   - Final Remarks

SAX-style XML parsing

- Sequential processing model
  - Program invokes parser with a set of callback functions
  - Parser scans input from start to finish
    - `<element` attributes...>
    - *content*
    - `</element>`
  - Invokes callbacks in file order
    - `startElement()`
    - `content()`
    - `endElement()`

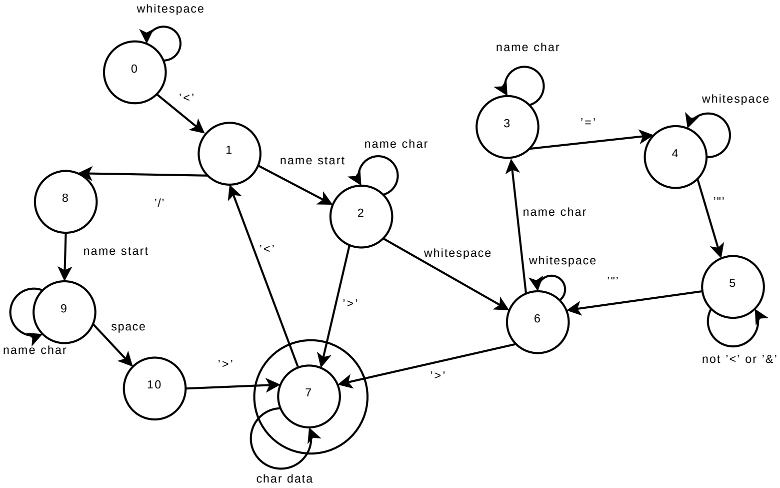
Outline

1. Introduction
   - Large XML Data
   - Ubiquity of Multi-processing Capabilities
   - SAX-based parsing
2. Parallel XML
   - PiXiMaL: Parallel Approach for Processing XML
   - Serial NFA Tests
3. Conclusions
   - Final Remarks

Token-Scanning With a DFA

- DFA-based table-driven scanning is both popular and fast
  - (or at least performance-competitive with other techniques)
- Input is read *sequentially* from start to finish
  - Each character is used to transition over states in a DFA
  - Transition may have associated actions
    - Supports languages that are not ``regular''
- Commonly used in high performance XML parsers, such as TDX (C) and Piccolo (Java)
  - Amenable to SAX parsing
  - Piximal-DFA uses this approach

DFA Used in PIXIMAL-DFA



PiXiMaL-DFA Implementation Details

- `mmap(2)`s input file to save memory
- Uses {length, pointer} string representation
  - Strings (for tagnames, attribute values) point into the mapped memory
  - All the way through the SAX-style event interface
- DFA is encoded as two tables
  - Table of ``next'' state numbers indexed by state number and input character
  - Table of boolean ``action required'' indicators indexed by ``current'' state and ``next'' state
    - Action required $\Longrightarrow$ a function is called to decode and execute the required action
  - DFA table is generated at compile time using a separate generator program

Parallel Scanning With a DFA?

- DFA-based scanning $\Longrightarrow$ sequential operation
- Desire: run multiple, concurrent DFAs throughout the input
  - Generally not possible because the start state would be unknown

Overcoming Sequentiality With an NFA

- Problem: start state is unknown
- Solution: assume every possible state is a start state
  - Construct an NFA from the DFA used in Piximal-DFA
    1. Mark every state as a start state
    2. Remove all the garbage state and all transitions to it
    3. Create an queue for each start state to store actions that should be performed
  - Such an NFA can be applied on any substring of the input
- Piximal-NFA is the parser that does all of this:
  - Partition input into segments
  - Run Piximal-DFA on the initial segment
  - Run NFA-based parsers on subsequent partition elements
  - Fix up transitions at partition boundaries and run queued actions

PIXIMAL-NFA's Parameters

- *split_percent*:
    - The portion of input to be dedicated to the first element of the partition, expressed as a percentage of the total input length
- *number_of_threads*:
    - The number of threads to use on a run
    - The final $(100 - split\_percent)\%$ of the input is divided evenly across the remaining $(number\_of\_threads - 1)$ partitions
        - The final partition element gets up to $number\_of\_threads - 2$ fewer characters

Outline

1. Introduction
   - Large XML Data
   - Ubiquity of Multi-processing Capabilities
   - SAX-based parsing
2. Parallel XML
   - Piximal: Parallel Approach for Processing XML
   - Serial NFA Tests
3. Conclusions
   - Final Remarks

Binghamton University
State University of New York

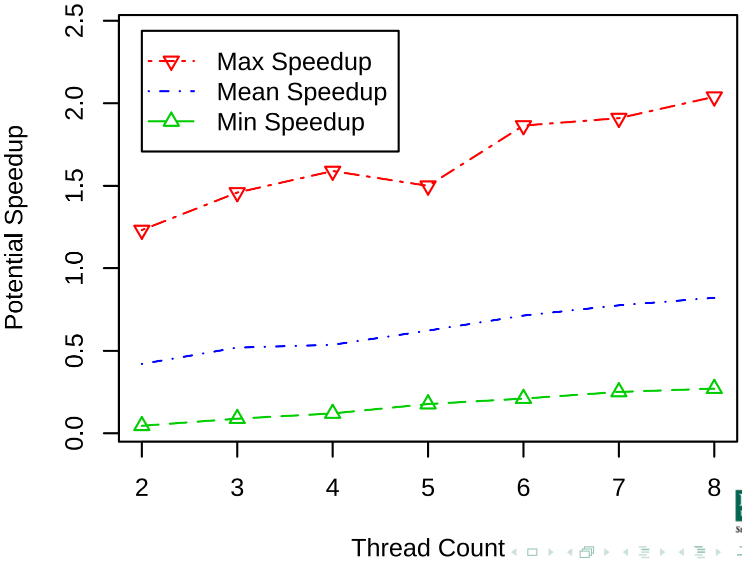
Serial NFA Tests

- Test hypothesis: the extra work required by using an NFA is offset by dividing processing work across multiple threads
  - Run each automaton-parser sequentially and independently
  - Divide the work as usual, with a range of *split_percents* and *number_of_threads*
  - Time each component independently
  - Completely parses the input, generating the correct sequence of SAX events
- The maximum time for all components to complete (plus fix up time) represents an upper bound on the time PIXIMAL-NFA would take with components running concurrently

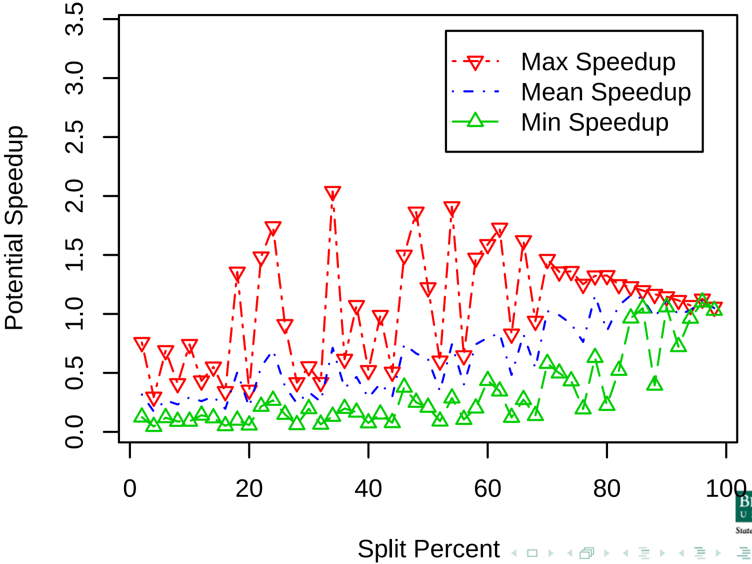
Test conditions

- Synthetic data
  - Arrays of Integers, Strings, Mesh Interface Objects
  - SOAP encoded
  - Same as previously presented in benchmarks
- Across a cluster (taking mean of results)
- Range of input sizes
- Range of parameters (*split_percent*, *number_of_threads*)

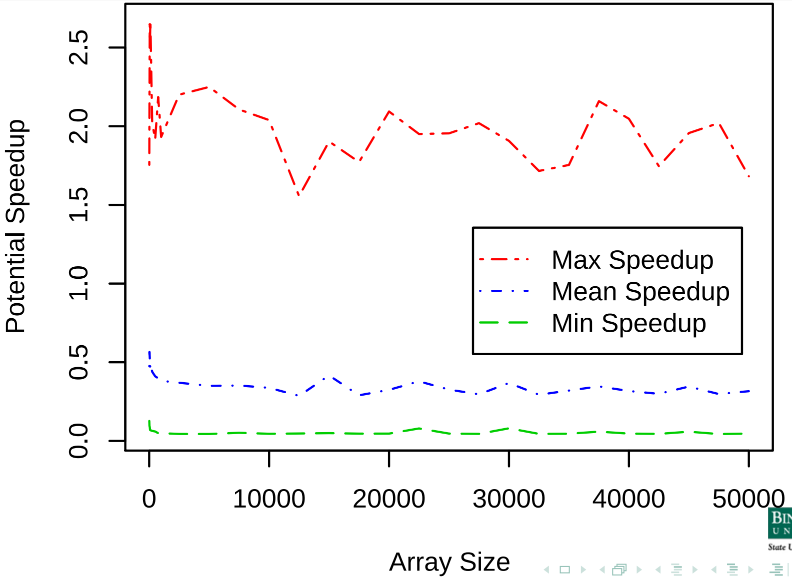
Modest Speedup Scalability for 10,000 Integers



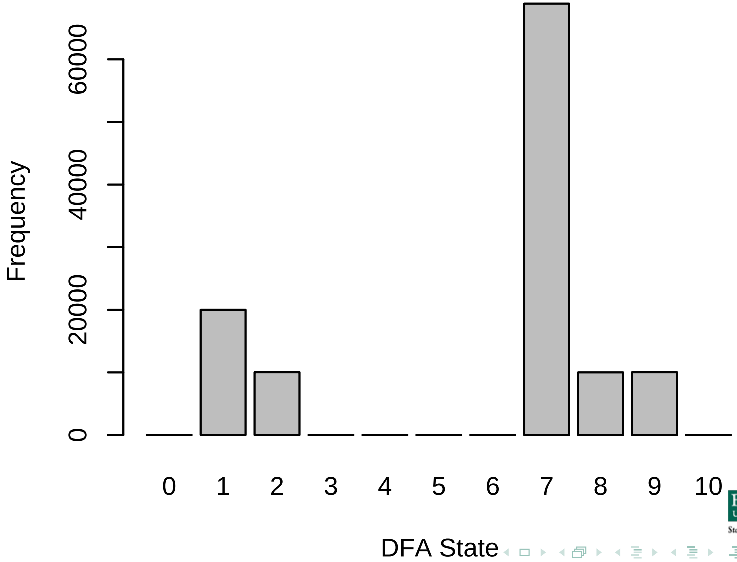
Split_Percent Critical for Speedup for 10,000 Integers



Inconsistent Speedup Over a Range of Array Lengths



Characters in 10,000 Integers in a Range of States

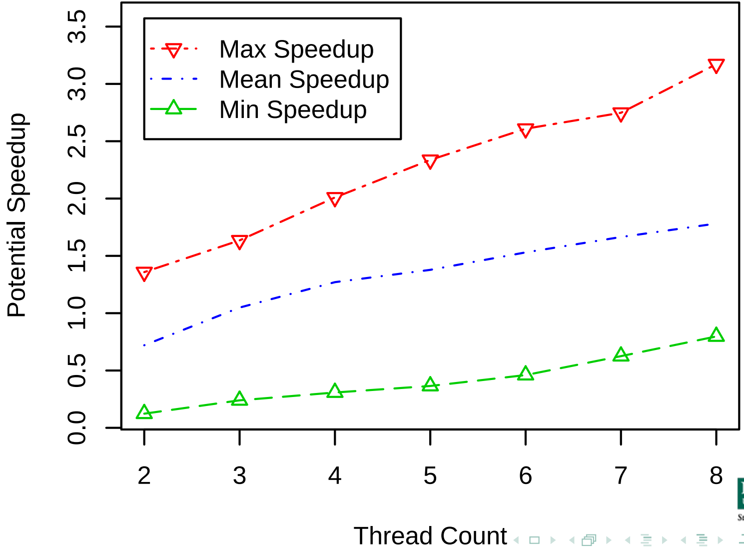


Conclusions From Integer Results

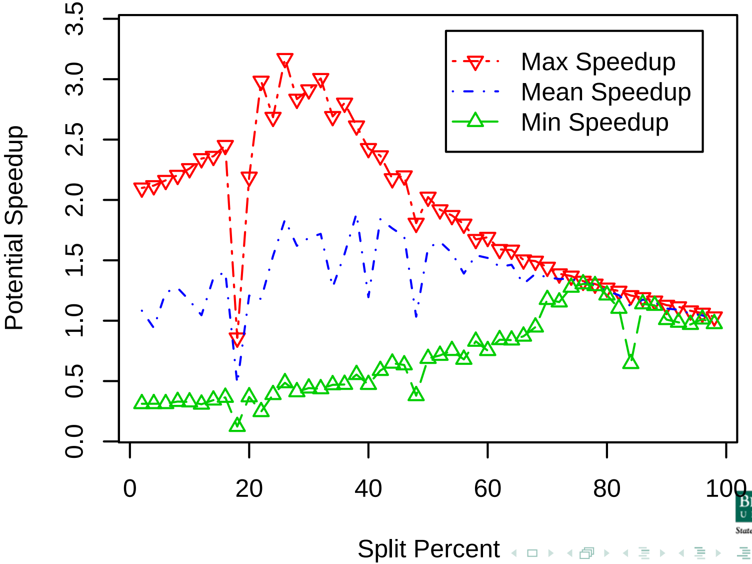
- Speedup is possible in this case
- Choice of split point is critical for achieving any speedup at all
- Characters in content sections account for roughly 60% of the input characters

- Input is 117 KB in length
- Consists mainly of
  `...<i>1234</i><i>1235</i><i>1236</i>...`

Speedup Improves with *Thread_Count* for 10,000 Strings

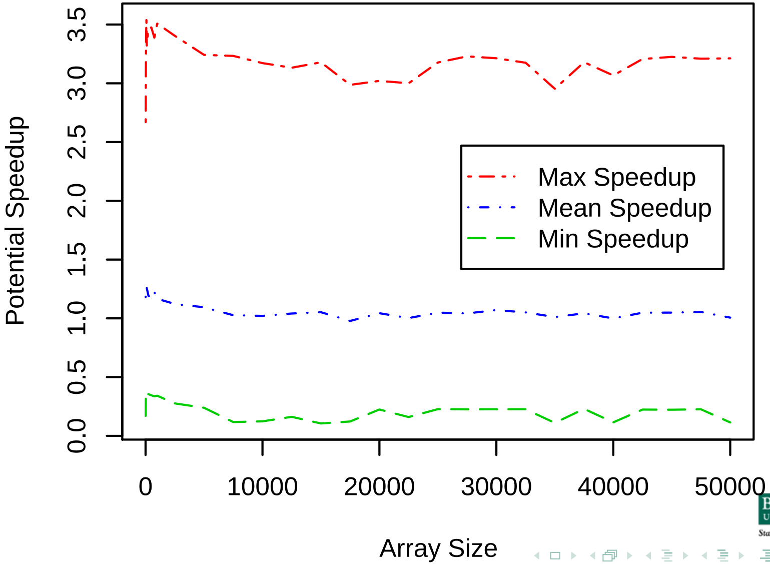


Split_Percent Less Critical for 10,000 Strings

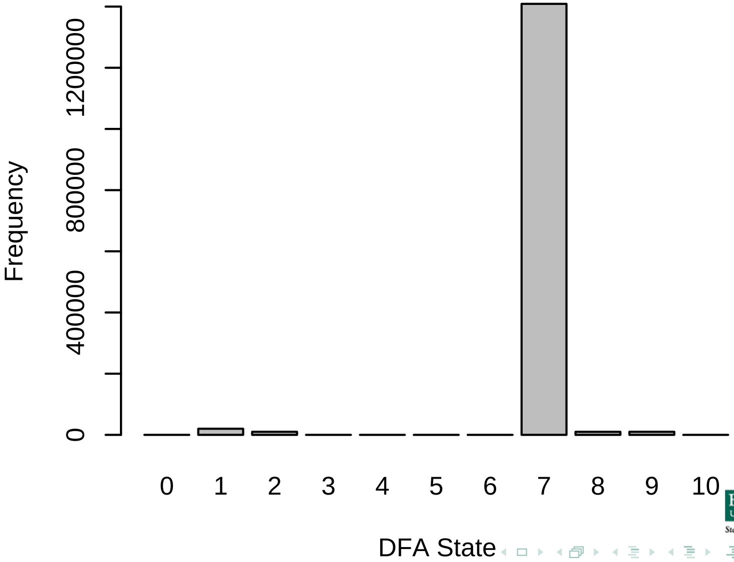


Binghamton University
State University of New York

Consistent Speedup Over a Range of Input Sizes



Characters in 10,000 Strings are Mainly in Content



Conclusions from String Results

- This sort of input is much more amenable to this approach
  - In maximum potential speedup achieved
  - In number of cases where speedup is > 1
- Split point is much less important here
- Characters in content sections account for roughly 99% of the input characters
- Input is 1.4 MB in size (though similar results are seen in inputs that are 117 KB)
- Consists mainly of ...<i>String content for the array element number 0. This is long to test the hypothesis that longer content sections are better for the NFA.</i>...

Conclusions from Serial NFA Test

- Shape of the input strongly determines the efficacy of the PIXIMAL approach
  - MIO has similar state usage and mix of content and tags as the integer and PIXIMAL has a similar performance profile there
  - PIXIMAL works well on inputs with longer content sections punctuated by short tags
- Starting in a content section helps because the '<' character eliminates a large number of execution paths through the NFA
  - If '>' could be treated similarly by the parser, starting in a tag would be less harmful

Outline

Binghamton University
State University of New York

Conclusions

- Scientific applications strain existing XML infrastructure
- A parallel parsing approach is necessary to achieve increased parser performance as document sizes grow
- Restricting XML slightly should provide better performance at a low semantic cost
- PIXIMAL's applicability is dependent on the characteristics of the input file

Summary

1. Introduction
   - Large XML Data
   - Ubiquity of Multi-processing Capabilities
   - SAX-based parsing
2. Parallel XML
   - PiXiMaL: Parallel Approach for Processing XML
   - Serial NFA Tests
3. Conclusions
   - Final Remarks

Thank you for your time.

Questions?

Extra Slides

The following slides are additional and not part of the presentation.

Limitations

- PThread overhead during concurrent runs
- Restrictions on XML format
  - Namespaces
  - CDATA
  - Unicode
  - Processing Instructions
  - Validation
- Optimal splitting algorithm unknown

Related Work in High Performance XML Processing

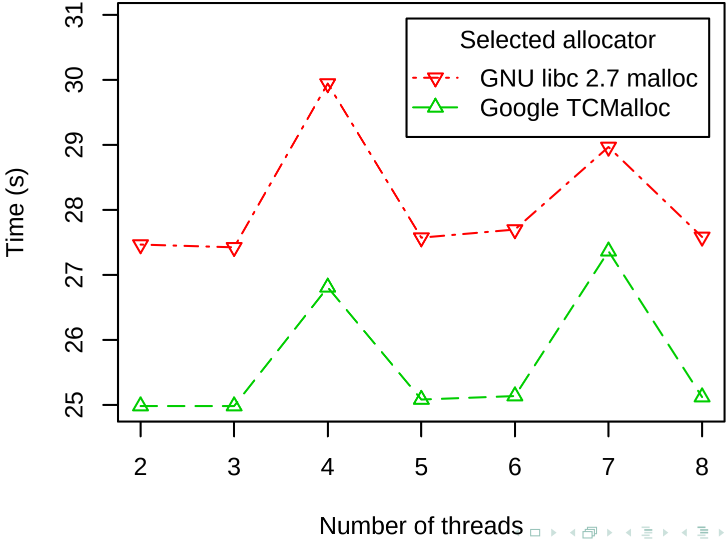
- Look-aside buffers/String caching (gsoap, XPP)
- Trie data structure with schema-specific parser (Chiu et al 02, Engelen 04)
- One pass table-driven recursive descent parser (Zhang et al 2006)
- Pre-scan and schedule parser (Lu et al 2006)
- Parallelized scanner, scheduled post-parser (Pan et al 2007)

Comparison with Expat

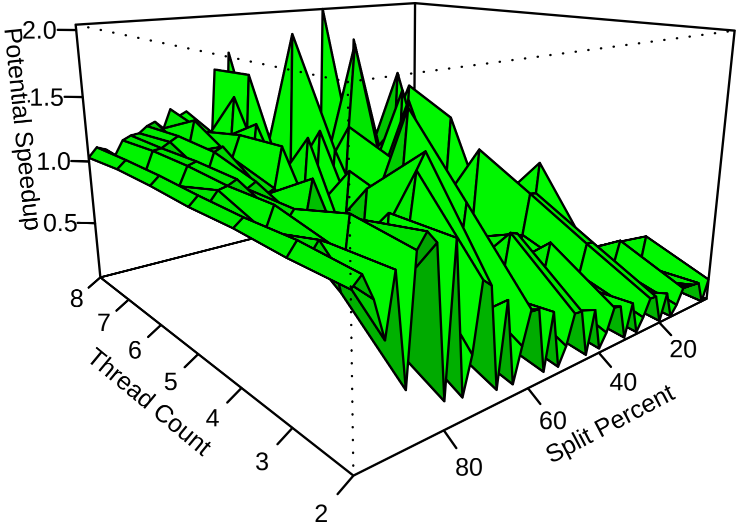
Input file	Expat	Piximal-dfa	Piximal-nfa
psd-7003	15.51	17.47	14.18

Table: Parse time, in seconds per parse, of high performance parsers

Comparison Between GLibC and TCMalloc



Perspective Plot for 10,000 Integers



Perspective Plot for 10,000 Strings