Approaching a Parallelized XML Parser Optimized for Multi-Core Processors

Michael R. Head\textsuperscript{1} \quad Madhusudhan Govindaraju\textsuperscript{2}

Department of Computer Science
Grid Computing Research Laboratory
Binghamton University (SUNY)
\{\textsuperscript{1}mike, \textsuperscript{2}mgovinda}@cs.binghamton.edu

Workshop on Service-Oriented Computing Performance: Aspects, Issues, and Approaches, 2007
*Supported in part by NSF grant CNS-0454298 and DOE grant DE-FG02-08ER25803
Outline

1 Motivation
   - Prevalence of Large XML Documents
   - Multi-Core
   - Optimizations

2 Experiments and Results
   - Experimental Setup
   - Results
Motivation

Experiments and Results

Summary

Prevalence of Large XML Documents

Multi-Core

Optimizations

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)
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
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
  - Multi-threaded programming considered “hard”
  - Reinforced in the current curricula and by existing languages and tools
Most string parsing techniques rely on a serial scanning process

**Challenge:** Existing (singly-threaded) XML parsers are already very efficient [Zhang et al 2006]
Research directions

- Pre-scan and schedule parser [Lu et al 2006]
- Parallelized scanner

Requires some communication/buffering between threads
But first...

- Try some easy, generic techniques
- Offload disk input to another thread/core
- Introduce two parsers which extend the existing, high performance **Piccolo** parser [Head et al 2006]
  - **Runahead**: opens two file descriptors for the input file
    - Start a thread that repeatedly calls `read()` on one of the file descriptors
    - Pass the other file descriptor to the existing Piccolo parser in the main thread
  - **Readahead**: opens one file descriptor for the input file, and one pipe
    - Start a thread that reads from the file descriptor and writes to the pipe
    - Pass the pipe to the existing Piccolo parser in the main thread*
Test run

- Run each parser (Piccolo, Runahead, and Readahead) on a large (GB-scale) XML file
  - Specifically, a protein sequence database file, pdf7003.xml
- No user code is run for any SAX event – just the parser itself is tested
- File cache is cleared between each run running a separate process that reads multiple gigabyte files
- Each test is run 50 times for each parser
- Hotspot is warmed by running the parser on another input file with identical content before timing begins
Two Environmental Conditions Tested

- **Architectures**
  - **UP**: Classic Uniprocessor P4-based machine (Dell workstation)
  - **SMP**: Classic Symmetrical MultiProcessing P4-based machine (has server-class I/O system) (IBM e-server)
  - **CMP**: Modern Chip MultiProcessing Core 2 Duo-based machine (Dell workstation)

- **System conditions**
  - **Cached**: The input file is read (hence loaded into the system file cache) before timing begins
  - **Uncached**: The input file is not read before timing begins (and flushed between each run)
Data Analysis

- Speedup for both of the proposed parsers is computed to compare across architectures.
- Baseline value is computing by averaging the times for each run of the unmodified Piccolo parser.
- Speedup for each run is computed by dividing the baseline by the time at each test point.
- Data presented is from the most recent tests (same systems as in the paper, but with an equitable amount of RAM in the CMP machine).
Speedup for the Readahead Parser Relative to Architecture

(Input Resides in Filesystem Cache)
Speedup for the Runahead Parser Relative to Architecture

(i) Input Resides in Filesystem Cache

 CMP
 SMP
 UP

Michael R. Head, Madhusudhan Govindaraju  Parallel XML
Motivation
Experiments and Results
Summary

Experimental Setup
Results

Speedup for the Runahead Parser Relative to Architecture

- SMP
- CMP
- UP

Run Number
(Input Flushed from Filesystem Cache)
Speedup for the CMP Architecture Relative to Parser Type

Runahead
Readahead

Run Number
(Input Flushed from Filesystem Cache)
Limitations

- Haven’t yet tested against a suite of differently sized and structured XML files
- Haven’t yet examined effect on system (CPU utilization, I/O load)
- Very simple optimization (for just 2 cores), may cause an overall slowdown in an application which may be able to use the core more effectively
Still, in some cases, the benefit is surprisingly substantial for such a simple change
- Compare performance against more complicated approaches