Efficient Smith-Waterman on multi-core with FastFlow

Marco Aldinucci
Computer Science Dept. - University of Torino - Italy

Massimo Torquati
Computer Science Dept. - University of Pisa - Italy

Massimiliano Meneghin
IBM Technology Campus, Dublin Software Lab, Ireland
Outline

- Motivations
- FastFlow
  - Performance
- Smith-Waterman benchmark
  - Performance
- Conclusions & Commercial
[< 2004] Shared Font-Side Bus (Centralized Snooping)
[2005] Dual Independent Buses (Centralized Snooping)
[2007] Dedicated High-Speed Interconnects (Centralized Snooping)
This and next generation Multi-cores

- Are programmed at “concurrent assembler” level
  - Complex, not portable, not efficient

- Exploit cache coherence
  - Memory fences are expensive
  - Will worsen with core count
  - Atomic ops do not solve the problem (still fences)

- Fine-grained parallelism is off-limits
  - I/O bound problems, High-throughput, Streaming, Irregular DP problems
  - Automatic and assisted parallelisation

[2009] i7 QuickPath
(MESI-F Directory Coherence)
E.g. farm (a.k.a. master-workers)

- Common paradigm
- Model foreach loop and Divide & Conquer
- Exploit it as a high-order language construct
  - Why should we re-code it from scratch each application?
  - A C++ template factory exploiting highly optimised implementation
E.g. farm with POSIX lock/unlock

Average execution time per task

Speedup

Number of Cores

Ideal 50 μS 5 μS 0.5 μS
Lock-free and CAS-free (fence-free)

- Single-Producer-Single-Consumer FIFO queues
  - Lamport et al. 1983 Trans. PLS (Sequential consistency only - passive)
  - Giacomoni et al. 2008 PPoPP (Relaxed consistencies (e.g. TSO) - passive)

- Multiple-Producers-Multiple-Consumers FIFO queues
  - with CAS (at least one) - passive ... a plethora
  - without CAS - passive ➔ Cannot be done
  - without CAS - active ➔ FastFlow
FastFlow: A step forward

Efficient applications for multicore and manycore
Smith-Waterman, N-queens, QT, C4.5, FP-Growth, ...

Autonomic Behav.Skeletons ...
Simulation Montecarlo Accelerator self-offloading

Streaming networks patterns
Skeletons: Pipeline, farm, D&C, ...

Arbitrary streaming networks (building blocks)
Lock-free SPSC, SPMC, MPSC, MPMC queues

Simple streaming networks (building blocks)
Lock-free SPSC queues and general threading model

Multi-core and many-core
cc-UMA or cc-NUMA featuring sequential or weak consistency

Problem Solving Environment

High-level programming

Low-level programming

Run-time support

Hardware

Applications

High-level programming

Implemented on top of lock-free/fence-free non-blocking synchronizations

C++ STL-like implementation

Wednesday, February 17, 2010
Coarse grain (50 μS workload)
Medium grain (5 μS workload)
Medium grain (5 μS workload)

TBB manual says: no less than 10,000 clock cycles, i.e. about 5 μS
Fine grain (0.5 μS workload)
Smith-Waterman algorithm
Local alignment - dynamic programming - $O(nm)$
Fast Smith-Waterman

- Smith-Waterman algorithm
  - Local alignment
  - Time and space demanding $O(mn)$, often replaced by approximated BLAST
  - Dynamic programming
  - Real-world application
    - It has been accelerated by using FPGA, GCPU (CUDA), SSE2/x86, IBM Cell

- Best software implementation up to now
  - SWPS3: evolution of Farrar’s implementation
    - SSE3 + POSIX IPC

BioBits
Experiment parameters

Affine Gap Penalty: 10-2k, 5-2k, ...
Substitution Matrix: BLOSUM50

A matrix $H$ is built as follows:

\[
H(i, 0) = 0, \quad 0 \leq i \leq m \\
H(0, j) = 0, \quad 0 \leq j \leq n
\]

\[
H(i, j) = \max \left\{ \begin{array}{c}
0 \\
H(i-1, j-1) + w(a_i, b_j) \\
H(i-1, j) + w(a_i, -) \\
H(i, j-1) + w(-, b_j)
\end{array} \right. 
\begin{array}{c}
\text{Match/Mismatch} \\
\text{Deletion} \\
\text{Insertion}
\end{array} 
\right\}, \quad 1 \leq i \leq m, 1 \leq j \leq n
\]

Where:
- $a, b =$ Strings over the Alphabet $\Sigma$
- $m =$ length($a$)
- $n =$ length($b$)
- $H(i,j)$ - is the maximum Similarity-Score between the substring of $a$ of length $i$, and the substring of $b$ of length $j$
- $w(c, d)$, $c, d \in \Sigma \cup \{',-\}$, $'$ is the gap-scoring scheme

- Substitution Matrix: describes the rate at which one character in a sequence changes to other character states over time
- Gap Penalty: describes the costs of gaps, possibly as function of gap length
Each query sequence (protein) is aligned against the whole protein DB.

- E.g. Compare unknown sequence against a DB of known sequences.

**SWPS3 implementation exploits POSIX processes and pipes**

- Faster than POSIX threads + locks
Smith Waterman (10-2k gap penalty)
Smith Waterman (10-2k gap penalty)
Smith Waterman (5-2k gap penalty)
Smith Waterman (5-2k gap penalty)
Is FastFlow easy to use? Yes!
Conclusions

FastFlow efficiently supports streaming applications on commodity SCM (e.g. Intel core architecture)

- More efficiently than POSIX (standard or CAS lock), Cilk, OpenMP, TBB

Smith Waterman algorithm with FastFlow

- Obtained from SWPS3 by syntactically substituting read and write on POSIX pipes with fastflow push and FastFlow pop an push

  - In turn, POSIX pipes are faster than POSIX threads + locks in this case

- Scores twice the speed of best known parallel implementation (SWPS3) on the same hardware (Intel 2 x Quad-core 2.5 GHz)
DON'T BELIEVE?

CHECK IT OUT!
Thank you! Questions?

Started in Nov '09
at today over 600 downloads

http://sourceforge.net/projects/mc-fastflow/

In the top ten of reddit.com/c++
from several weeks