INTEL® HPC DEVELOPER CONFERENCE
FUEL YOUR INSIGHT

INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

Reshaping core genomics software tools for the many-core era

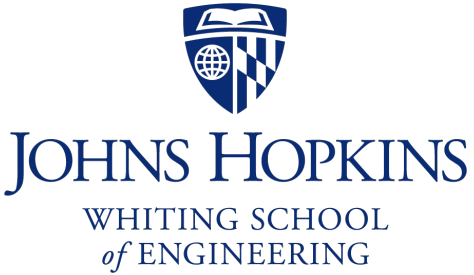
Ben Langmead

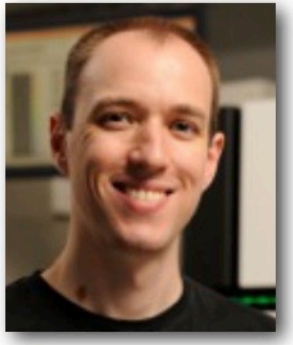
Assistant Professor

Johns Hopkins University, Department of Computer Science

November 2016

The Intel Parallel Computing Center at





Ben Langmead

Valentin Antonescu

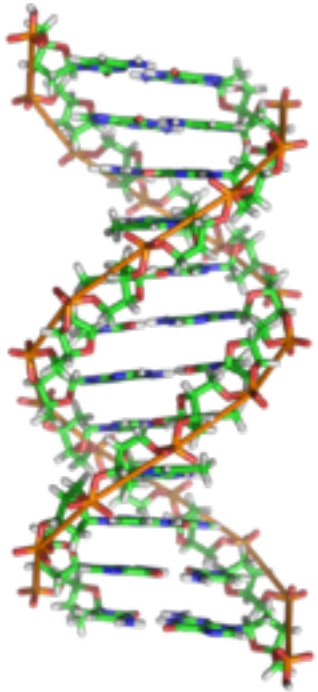
Chris Wilks

Steven Salzberg

Daehwan Kim

Outline

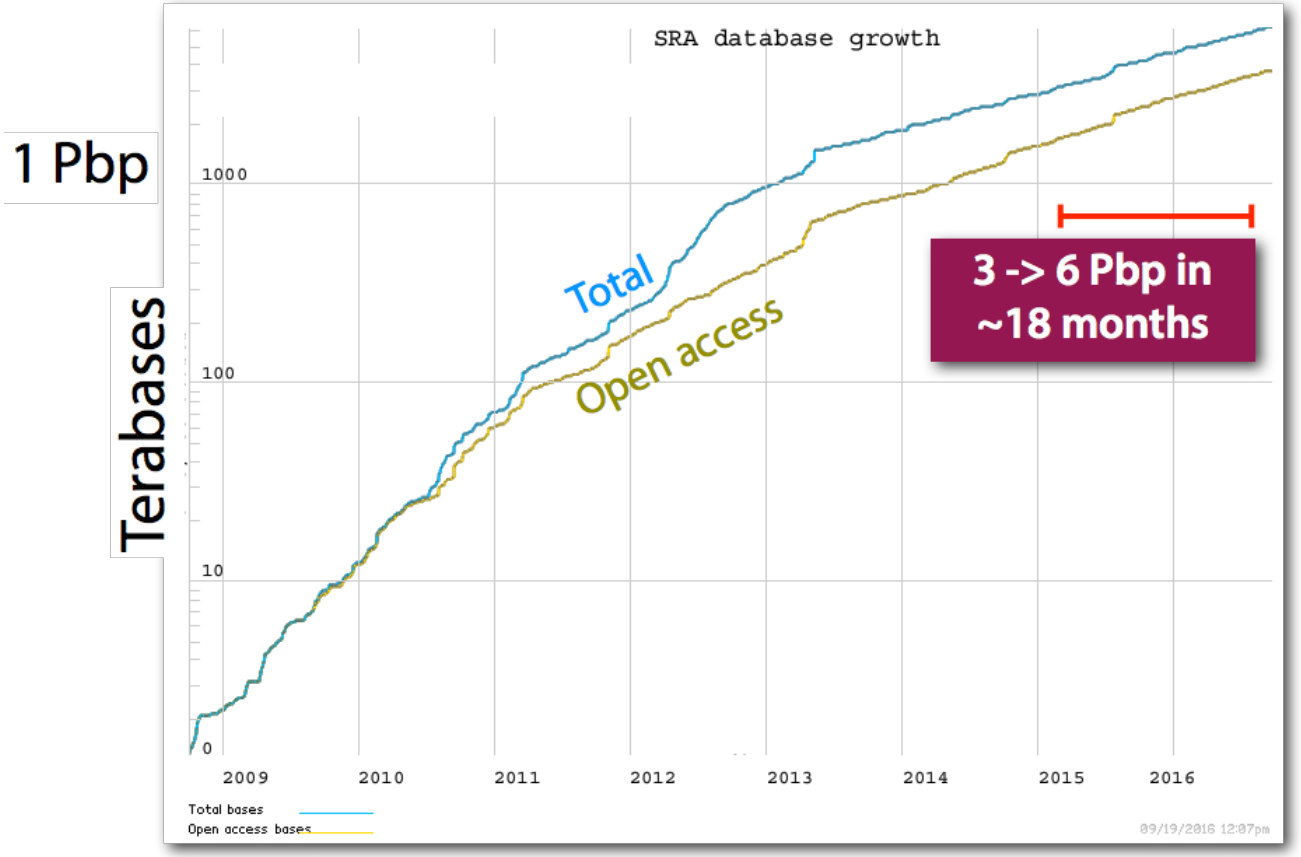
- Introduction to sequencing data analysis & Bowtie
- Thread scaling improvements using TBB
  - Choice of mutex
  - Two-stage parsing
- AVX2, AVX512-KNC & AVX512-KNC improvements
- Impact on the field



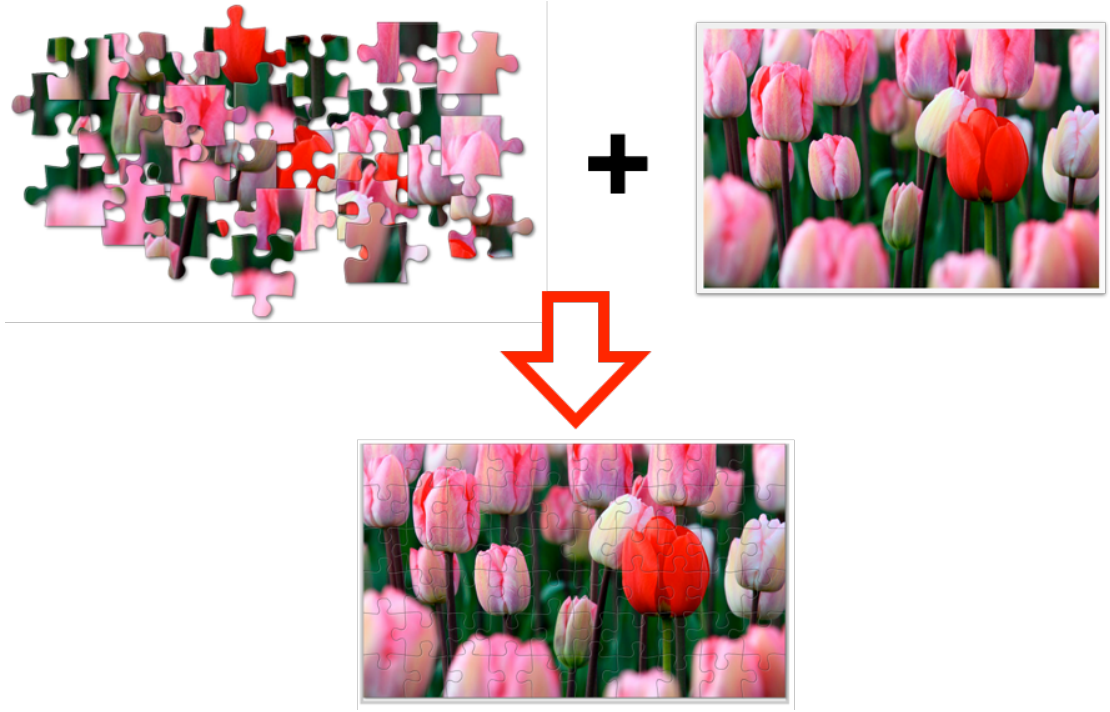
Sequencing

Sequencing

Sequencing

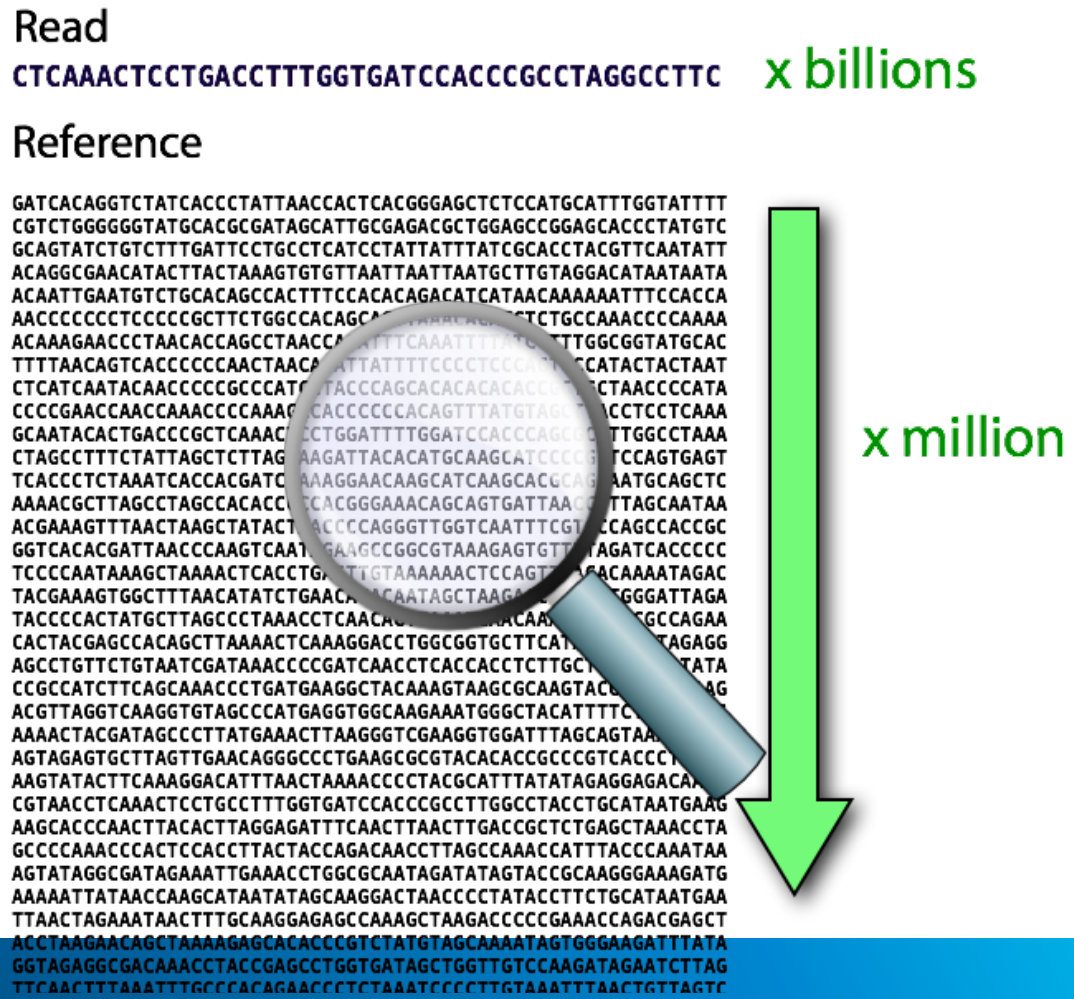

1 Pbp
Terabases
SRA database growth
1000
100
10
0
2009
2010
2011
2012
2013
2014
2015
2016
Total
Open access
3 -> 6 Pbp in
~18 months
Total bases
Open access bases
09/19/2016 12:07pm

Read alignment



Read alignment

- Needle in a haystack
- Billions of reads from a single week-long sequencing run
- Human reference genome is ~3B bases (letters) long



Bowtie and Bowtie 2

Software

Open Access

Ultrafast and memory-efficient alignment of short DNA sequences to the human genome

Ben Langmead, Cole Trapnell, Mihai Pop and Steven L Salzberg

- Together cited by >12K other scientific studies since 2009
- Bundled with dozens of other tools & many Linux distros

NATURE METHODS | BRIEF COMMUNICATION

Fast gapped-read alignment with Bowtie 2

Ben Langmead & Steven L Salzberg

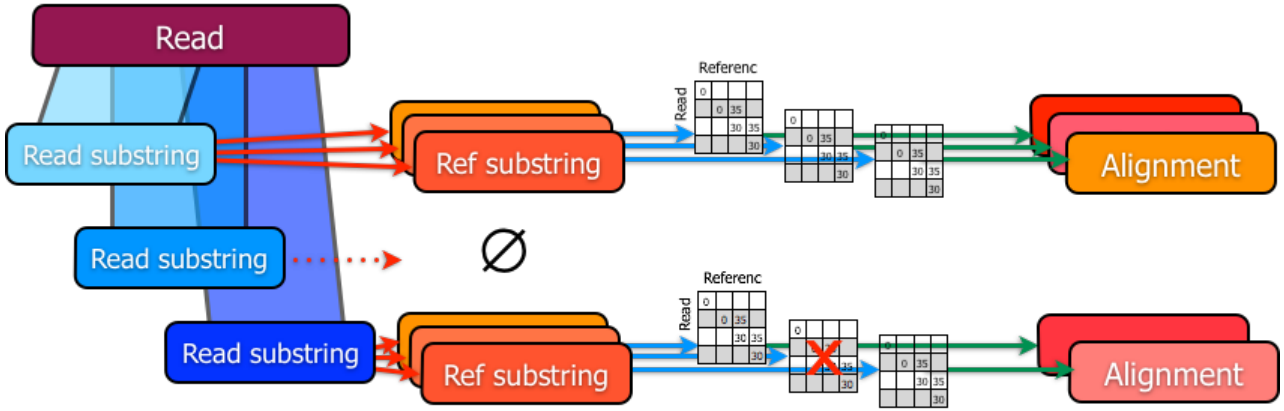
HISAT

- Based on Bowtie 2 and a leading *spliced* aligner for RNA sequencing data
- Cited in >75 scientific studies since 2015

Design of Bowtie & Bowtie 2

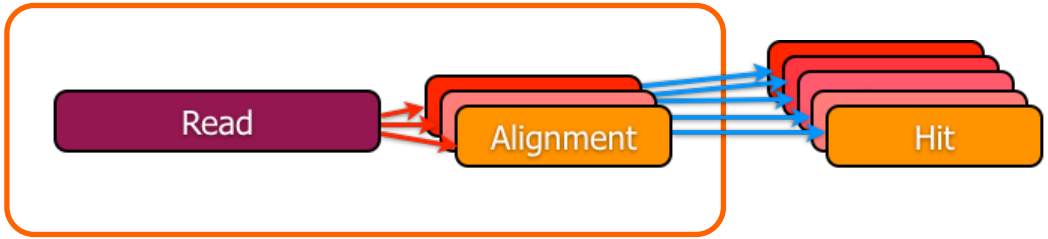
Bowtie 1

Bowtie 2



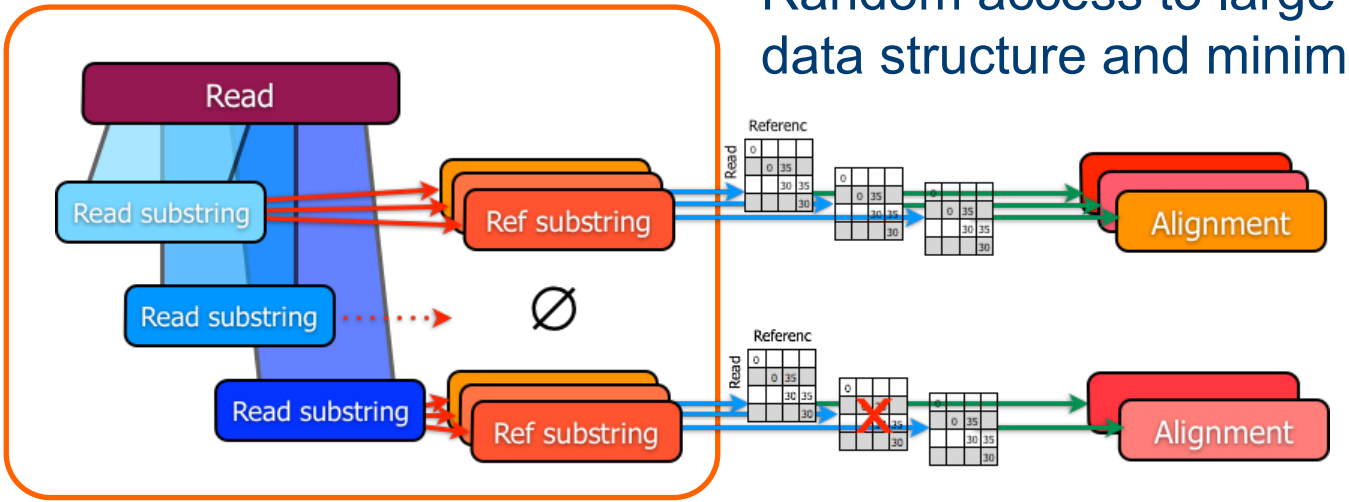
Design of Bowtie & Bowtie 2

Bowtie 1



Random access to large index data structure and minimal ILP

Bowtie 2

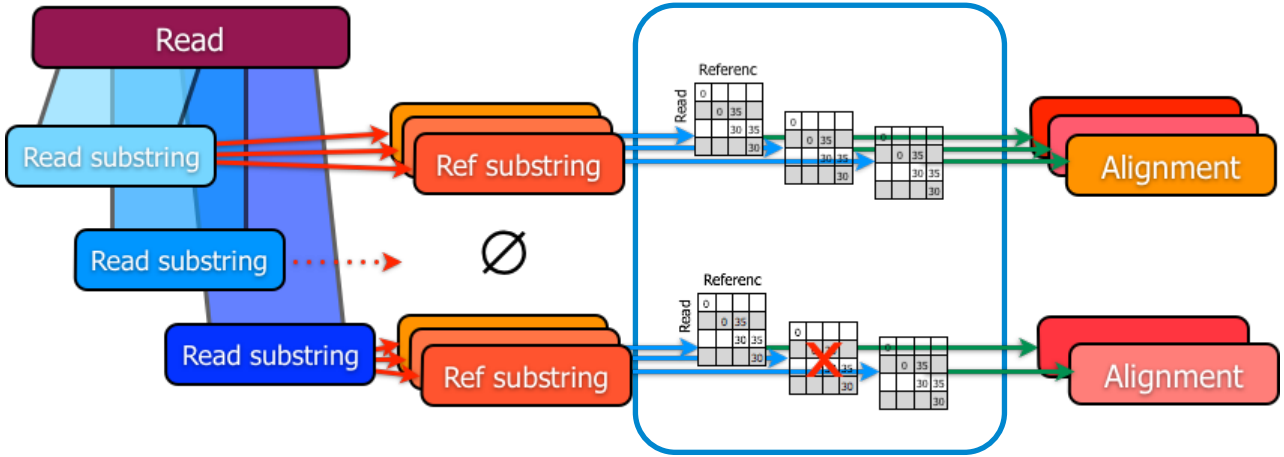


Design of Bowtie & Bowtie 2

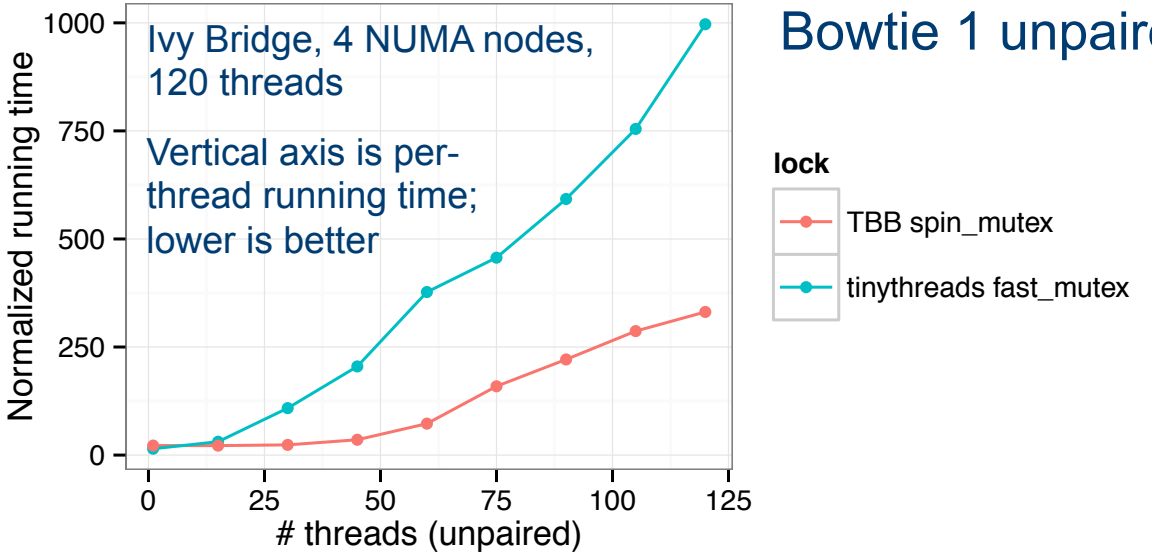
Bowtie 1

Dynamic programming, lots of ILP

Bowtie 2

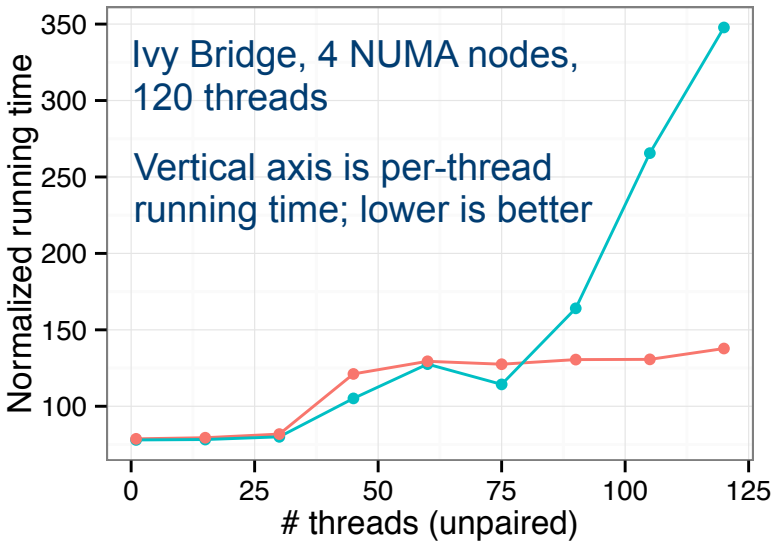


Thread scaling



- Switching to analogous TBB lock could bring big improvement

Thread scaling



Bowtie 2 unpaired

lock

None (stubbed I/O)

TBB spin_mutex

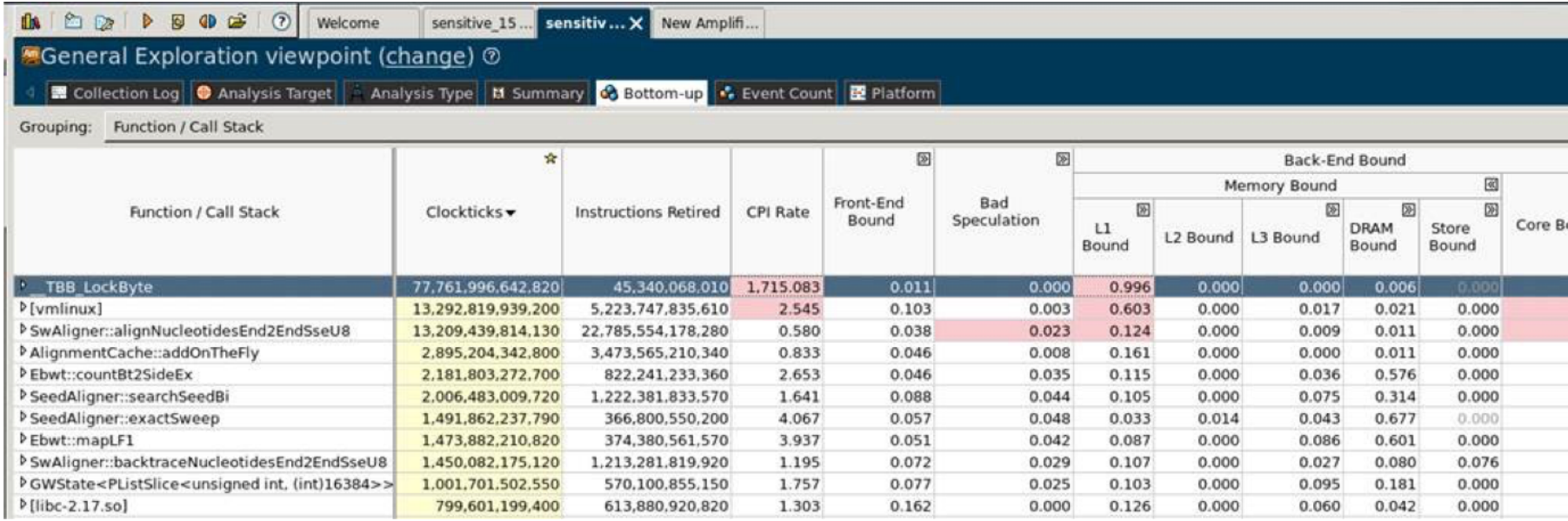
version

Original parsing

- Removing synchronization by “stubbing” input lock gives further improvement

Thread scaling

- Vtune investigation indicates synchronization itself (e.g. see __TBB_LockByte) is taking the time



Welcome | sensitive_15 ... | sensitiv ... | New Amplifi ...

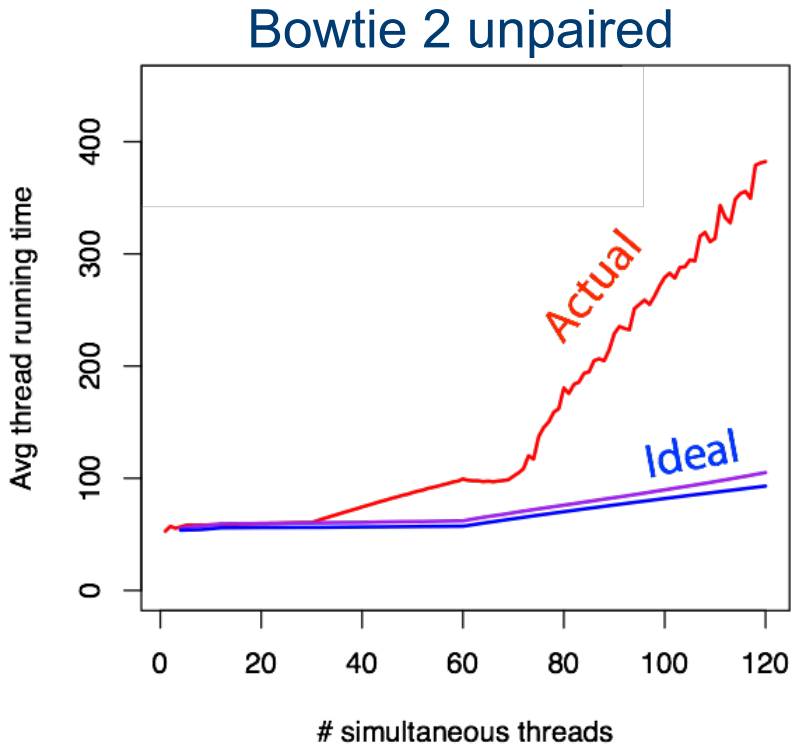
General Exploration viewpoint (change)

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Event Count | Platform

Grouping: Function / Call Stack

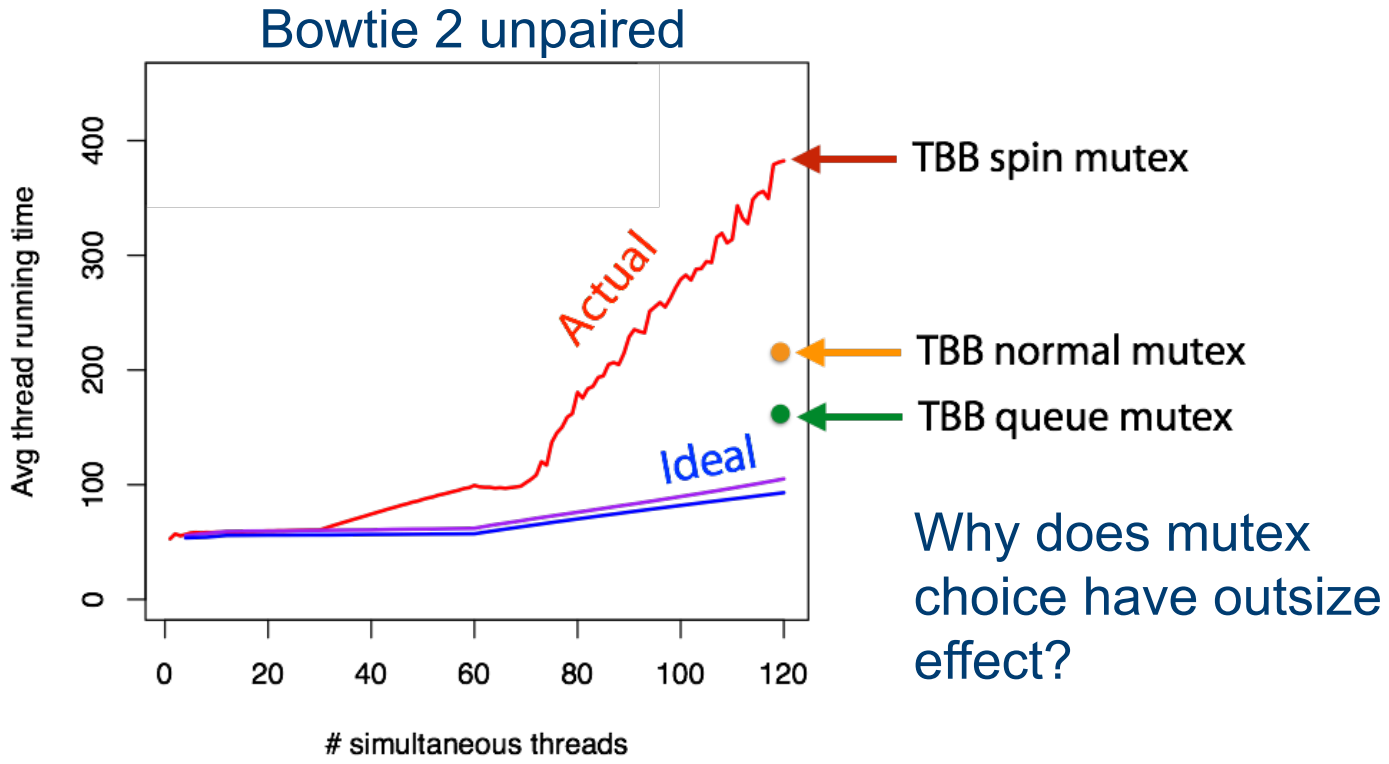
Function / Call Stack	Clockticks	Instructions Retired	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound					
						Memory Bound					
						L1 Bound	L2 Bound	L3 Bound	DRAM Bound	Store Bound	Core B
TBB_LockByte	77,761,996,642,820	45,340,068,010	1.715.083	0.011	0.000	0.996	0.000	0.000	0.006	0.000	
[vmlinux]	13,292,819,939,200	5,223,747,835,610	2.545	0.103	0.003	0.603	0.000	0.017	0.021	0.000	
SwAligner::alignNucleotidesEnd2EndSseU8	13,209,439,814,130	22,785,554,178,280	0.580	0.038	0.023	0.124	0.000	0.009	0.011	0.000	
AlignmentCache::addOnTheFly	2,895,204,342,800	3,473,565,210,340	0.833	0.046	0.008	0.161	0.000	0.000	0.011	0.000	
Ebwt::countBt2SideEx	2,181,803,272,700	822,241,233,360	2.653	0.046	0.035	0.115	0.000	0.036	0.576	0.000	
SeedAligner::searchSeedBi	2,006,483,009,720	1,222,381,833,570	1.641	0.088	0.044	0.105	0.000	0.075	0.314	0.000	
SeedAligner::exactSweep	1,491,862,237,790	366,800,550,200	4.067	0.057	0.048	0.033	0.014	0.043	0.677	0.000	
Ebwt::mapLF1	1,473,882,210,820	374,380,561,570	3.937	0.051	0.042	0.087	0.000	0.086	0.601	0.000	
SwAligner::backtraceNucleotidesEnd2EndSseU8	1,450,082,175,120	1,213,281,819,920	1.195	0.072	0.029	0.107	0.000	0.027	0.080	0.076	
GWState<PListSlice<unsigned int, (int)16384>>	1,001,701,502,550	570,100,855,150	1.757	0.077	0.025	0.103	0.000	0.095	0.181	0.000	
[libc-2.17.so]	799,601,199,400	613,880,920,820	1.303	0.162	0.000	0.126	0.000	0.060	0.042	0.000	

Thread scaling



How to close the gap between actual and ideal performance?

Thread scaling



Why does mutex choice have outsize effect?

Thread scaling

- Mutex spinning on atomic op (compare-and-swap, test-and-set), spurs exchange of cache coherence messages
- Image by Kayvon Fatahalian, Copyright 2015 Carnegie Mellon University

Test-and-set lock performance

Benchmark: total of N lock/unlock sequences (in aggregate) by P processors
Critical section time removed so graph plots only time acquiring/releasing the lock

Figure credit: Culler, Singh, and Gupta

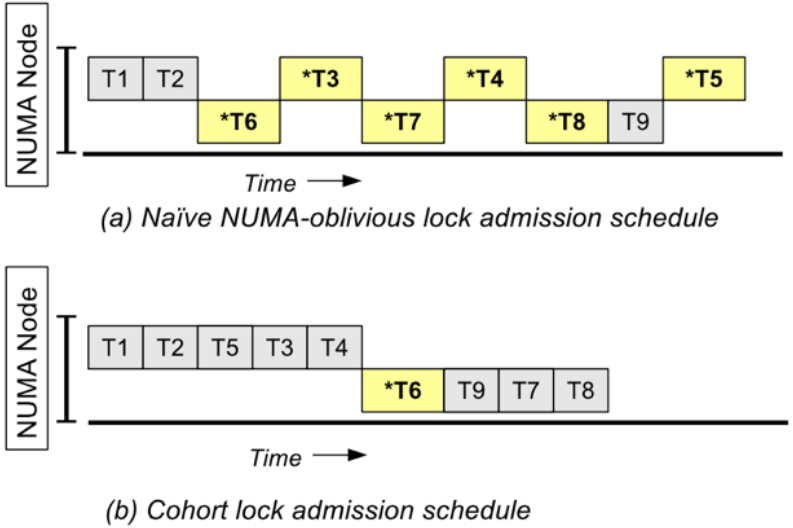
Thread scaling

- Even a standard pthreads mutex was outperforming the spin lock when running one thread per available core
  - More evidence that cache coherence traffic is culprit

- Queue locks are known to have better cache properties
  - Waiting thread spins on normal (non-atomic) read
  - Cache line read belongs exclusively to that thread and can live in L1

Thread scaling

- We hypothesized a NUMA-aware “cohort lock” could help further



Dice, David, Virendra J. Marathe, and Nir Shavit. "Lock cohorting: a general technique for designing NUMA locks." *ACM SIGPLAN Notices*. Vol. 47. No. 8. ACM, 2012.

Cohort locking

```
class CohortLock {
public:
  CohortLock() : lockers_numa_idx(-1) {
    starvation_counters = new int[MAX_NODES]();
    own_global = new bool[MAX_NODES]();
    local_locks = new TKTLock[MAX_NODES];
  }
  ~CohortLock() {
    delete[] starvation_counters;
    delete[] own_global;
    delete[] local_locks;
  }
  void lock();
  void unlock();
private:
  static const int STARVATION_LIMIT = 100;
  static const int MAX_NODES = 128;
  volatile int*  starvation_counters; // 1 per node
  volatile bool* own_global;          // 1 per node
  volatile int   lockers_numa_idx;    // 1 per node
  TKTLock* local_locks;               // 1 per node
  PTLLock global_lock;
};
```

- Each NUMA node has per-node ticket lock
- Other per-node information tracks when to pass lock to other threads on same node
- Single global partitioned ticket lock

Cohort locking

```
void CohortLock::lock() {
  const int numa_idx = determine_numa_idx();
  local_locks[numa_idx].lock();
  if(!own_global[numa_idx]) {
      global_lock.lock();
  }
  starvation_counters[numa_idx]++;
  own_global[numa_idx] = true;
  lockers_numa_idx = numa_idx;
}

void CohortLock::unlock() {
  assert(lockers_numa_idx != -1);
  int numa_idx = lockers_numa_idx;
  lockers_numa_idx = -1;
  if(local_locks[numa_idx].q_length() == 1 ||
     starvation_counters[numa_idx] > STARVATION_LIMIT)
  {
    global_lock.unlock();
    starvation_counters[numa_idx] = 0;
    own_global[numa_idx] = false;
  }
  local_locks[numa_idx].unlock();
}
```

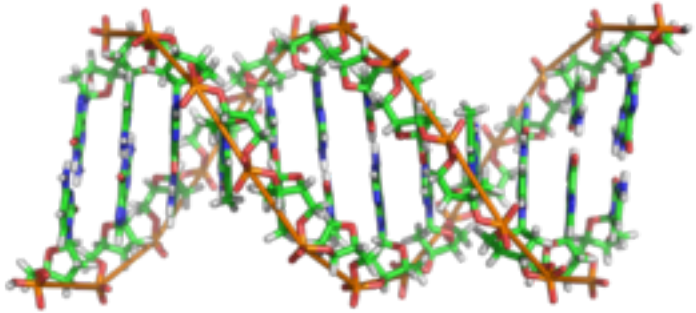
- When locking:
  - Grab local lock
  - Once grabbed, grab global lock if not already owned by this node
- When unlocking:
  - Is another thread on same node queued? If so, hand lock to next in queue
  - Otherwise release global & local locks
  - Override hand-off if others are starving

Cohort locking

- Another implementation of cohort locking available in ConcurrencyKit: http://concurrencykit.org
  - https://github.com/concurrencykit/ck/blob/master/include/ck_cohort.h

Thread scaling

- Chris Wilks added TBB queue locks, JHU/TBB Cohort locks (2 flavors) to Bowtie 2, Bowtie & HISAT
- Available in public branches, with all but cohort locks available in master branch and in recent releases

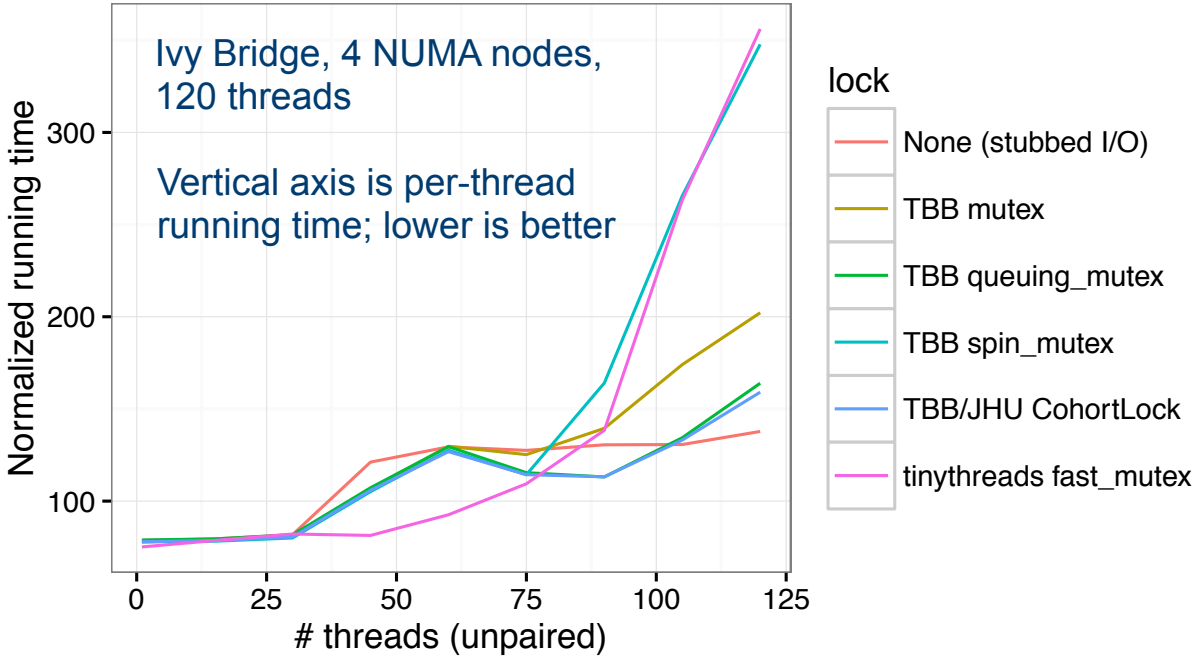


Thread scaling

- Novel strategy splits input parsing into two “phases”
- First (“light parsing”) rapidly detects record boundaries, requiring synchronization but with very brief critical section
- Second (“full parsing”) fully parses each record (pictured, right) with no synchronization
- Minimizes time spent in crucial critical section

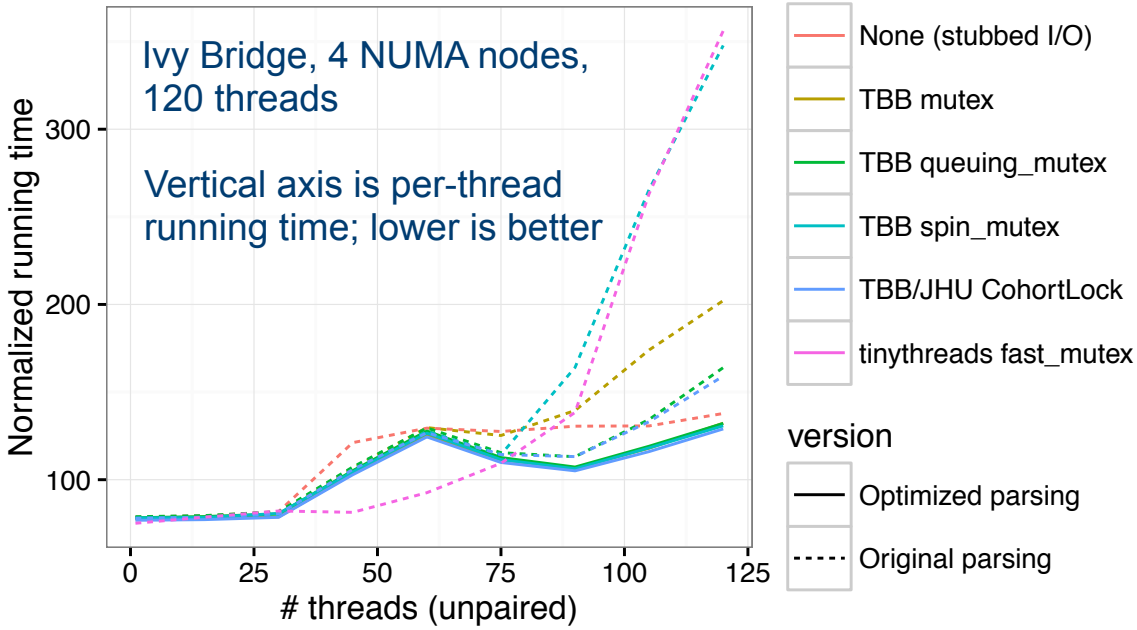
```
@ABC_123_1
GCTATTATGCTAT
+
JJSYEGGU8233^
@ABC_424_1
GTGATATGCAT
+
SYEG!U8@233
@ABCD_9_1
GCTATTATGCTATAAAC
+
JJSYEGGU8233^32FR
@D_91231_1
GCTATTATGCTAT
+
JJSYEGGU8233^
…
```

Thread scaling: Bowtie 2 unpaired



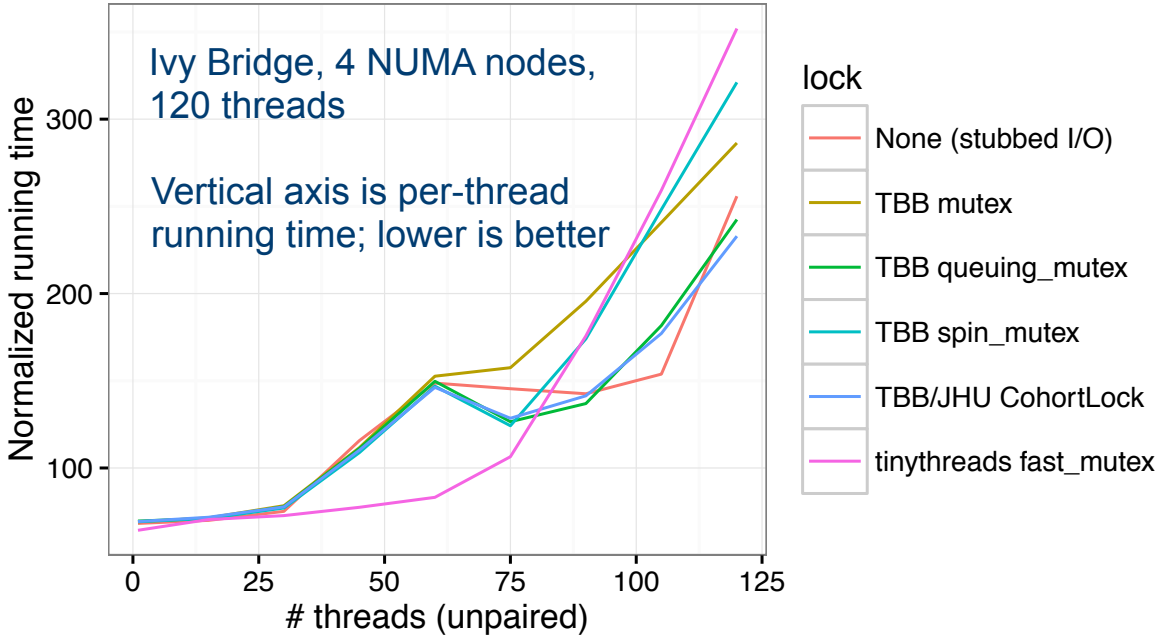
- TBB queuing_mutex and TBB/JHU cohort lock perform best

Thread scaling: Bowtie 2 unpaired



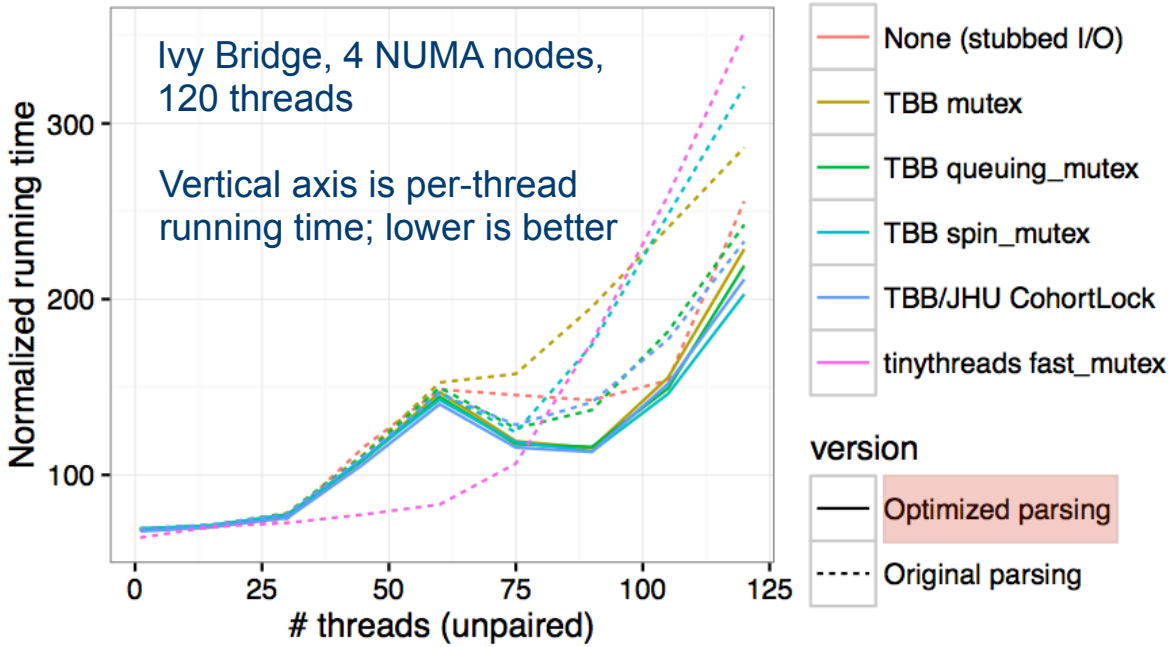
- Two-phase parsing yields substantial thread-scaling boost; close to perfect up to 120 threads, regardless of mutex

Thread scaling: Bowtie 2 paired-end



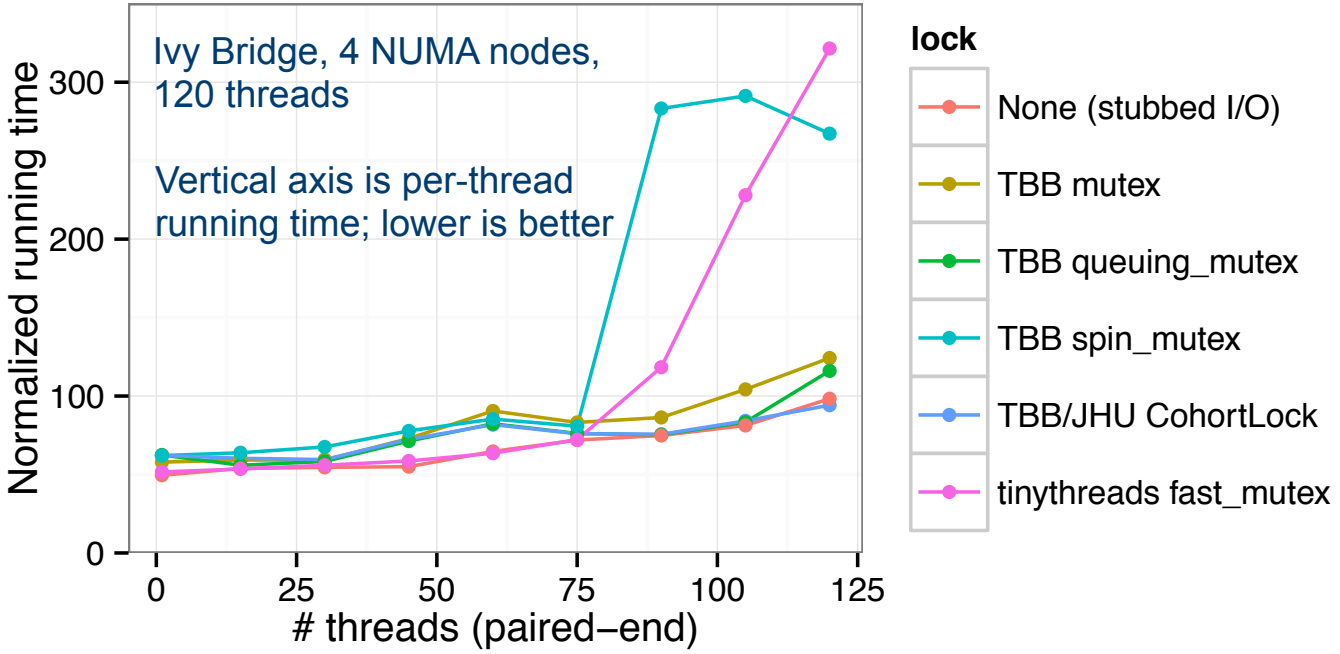
- queuing_mutex and cohort lock again perform the best, near ideal

Thread scaling: Bowtie 2 paired-end



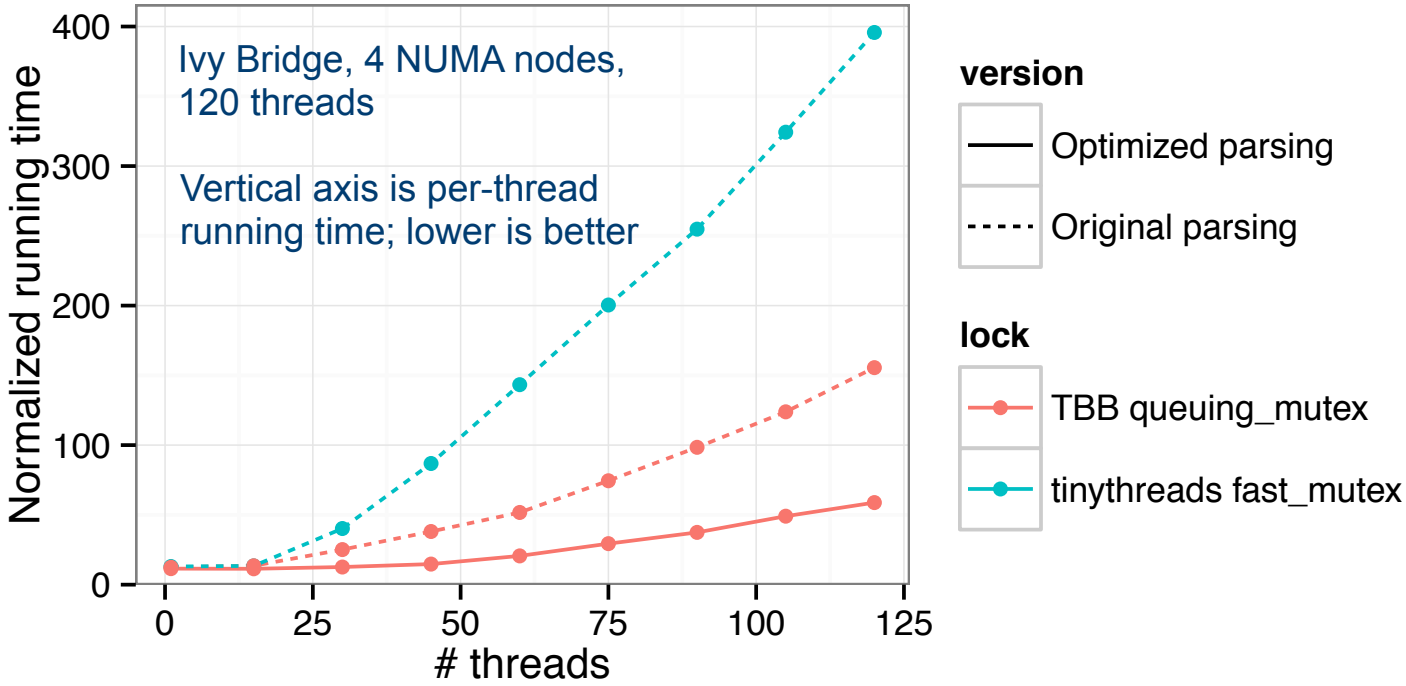
- Two-phase parsing yields substantial thread-scaling boost; close to perfect up to 120 threads, with mutex having smaller impact

Thread scaling: Bowtie



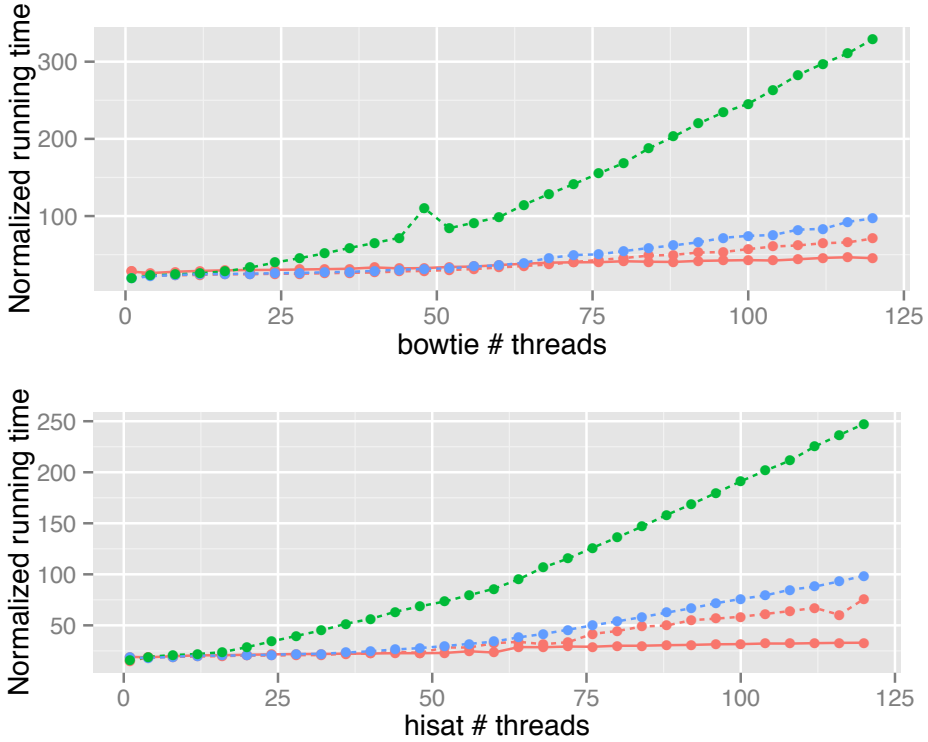
- As with Bowtie 2, near-ideal scaling with queuing and cohort locks

Thread scaling: HISAT unpaired



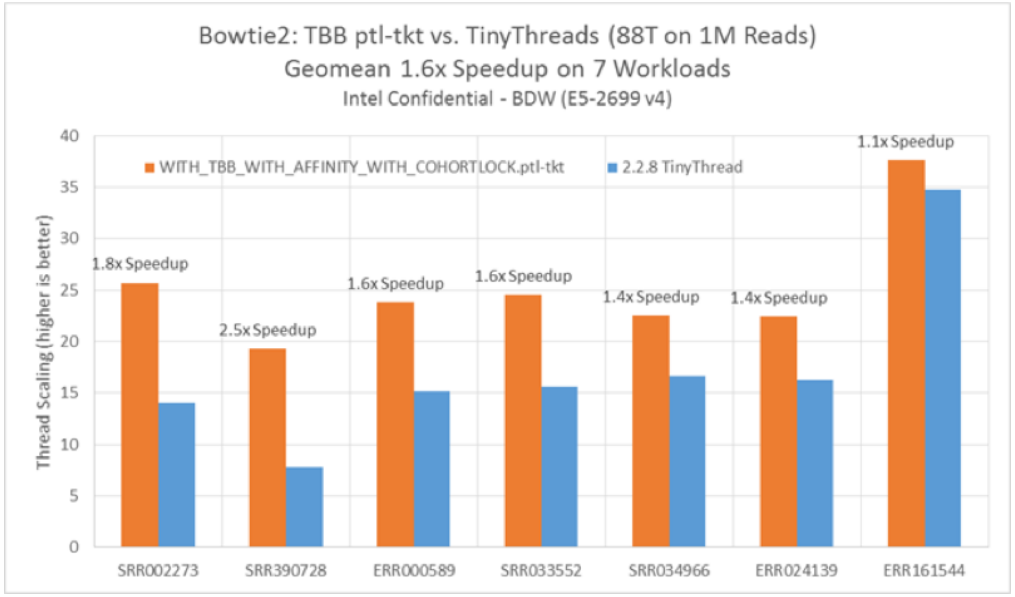
- Huge improvements with queuing_lock and two-phase parsing

Thread scaling



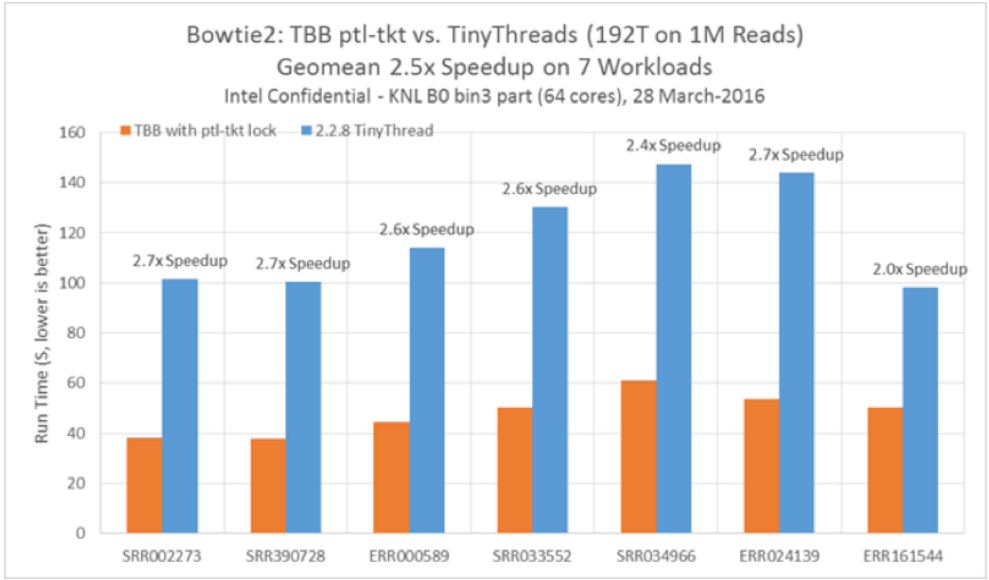
- Further gains possible with batch parsing, where the first phase “lightly” parses several reads at once, reducing # critical section entrances

Thread scaling: Bowtie 2 on Broadwell



- Experiment conducted by John Oneill at Intel
- TBB + optimized parsing yields speedups of 1.1x - 1.8x on 88 threads on Broadwell E5-2699 v4 part. TBB/JHU Cohort lock outperforms other mutexes.

Thread scaling: Bowtie 2 on Knight's Landing

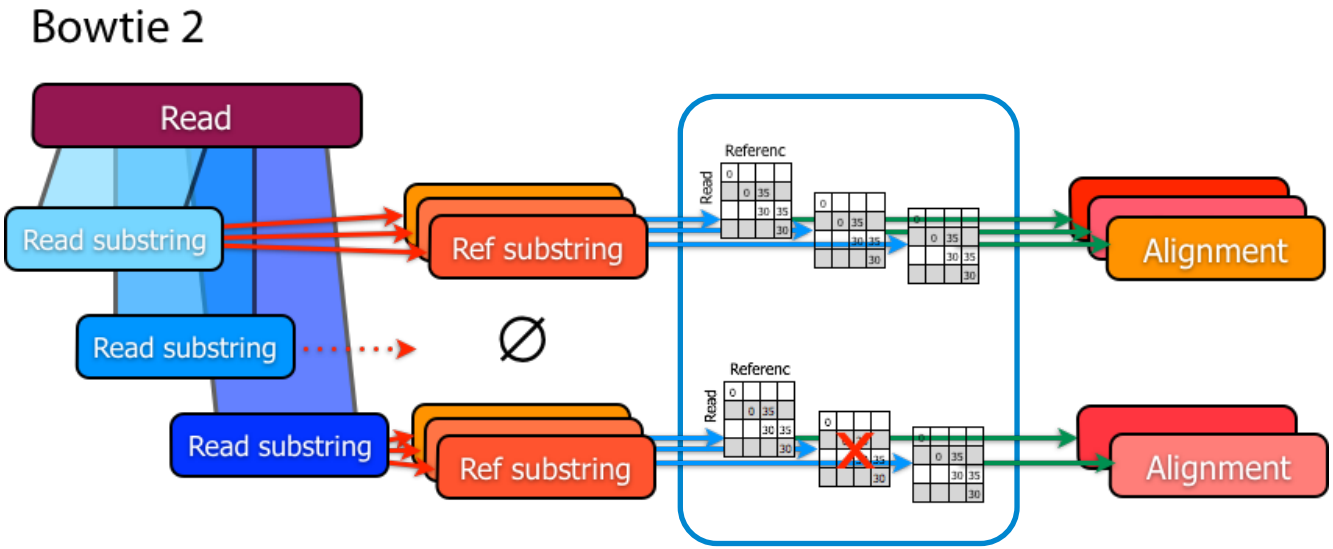


- Experiment conducted by John Oneill at Intel
- TBB + optimized parsing yields speedups of 2x - 2.7x on 192 threads on KNL B0 bin3 part. TBB/JHU Cohort lock outperforms other mutexes.

Thread scaling: summary

- Using a queue mutex / cohort lock can yield *big* improvement over spin / normal lock
- Achieved near-ideal scaling up to 120 threads with (a) queue/cohort locks and (b) cleaner parsing for Bowtie, Bowtie 2.
- Promising scaling results on KNC & KNL; more to do
- Cohort locks were best option in Broadwell & KNL experiments
- Cohort locks seem to put KNL in a better position to outperform Xeon on genomics workloads

Vectorization of Bowtie 2 inner loop



- Dynamic programming alignment not unique to Bowtie 2
- Common to many sequence alignment problems

Vectorization of Bowtie 2 inner loop

```
// Outer loop to process the database sequence
for i := 0 … dbLen
   // Initialize F value to zeros.  Any errors to vH values
   // will be corrected in the Lazy-F loop.
   vF := <0, …, 0>;

   // Adjust the last H value to be used in the next
   // segment over
   vH := vHStore[segLen - 1] << 1;

   // Swap the two H buffers
   swap (vHLoad, vHStore);

   // Inner loop to process the query sequence
   for j := 0 … segLen
      // Add the scoring profile to vH
      vH := vH + vProfile[i][j];

      // Save any vH values greater than the max
      vMax := max (vMax, vH);

      // Adjust vH with any greater vE or vH values
      vH := max (vH, vE[j]);
      vH := max (vH, vF);

      // Save the vH values off
      vHStore[j] := vH;

      // Calculate the new vE and vF based on the
      // gap penalties for this search
      vH := vH – vGapOpen;
      vE[j] := vE[j] – vGapExtend;
      vE[j] := max (vE[j], vH);
      vF := vF – vGapExtend;
      vF := max (vF, vH);

      // Load the next vH value to process
      vH := vHLoad[j];
   endfor
```

```
// - - -  Lazy-F Loop  - - -
// Shift the vF left so its values can be used to
// correct the next segment over.
vF := vF << 1;

// Correct the vH values until there are no elements
// in vF that could influence the vH values.
j := 0;
while (AnyElement (vF > vHStore[j] – vGapOpen)
   vHStore[j] := max (vHStore[j], vF);
   vF := vF – vGapExtend;
   if (++j >= segLen)
      // If we processed the entire segment, we need
      // to carry the vF values to the next segment.
      vF := vF << 1;
      j := 0;
   endif
endwhile
endfor
```

Outer loop iterates over columns

Main loop fills

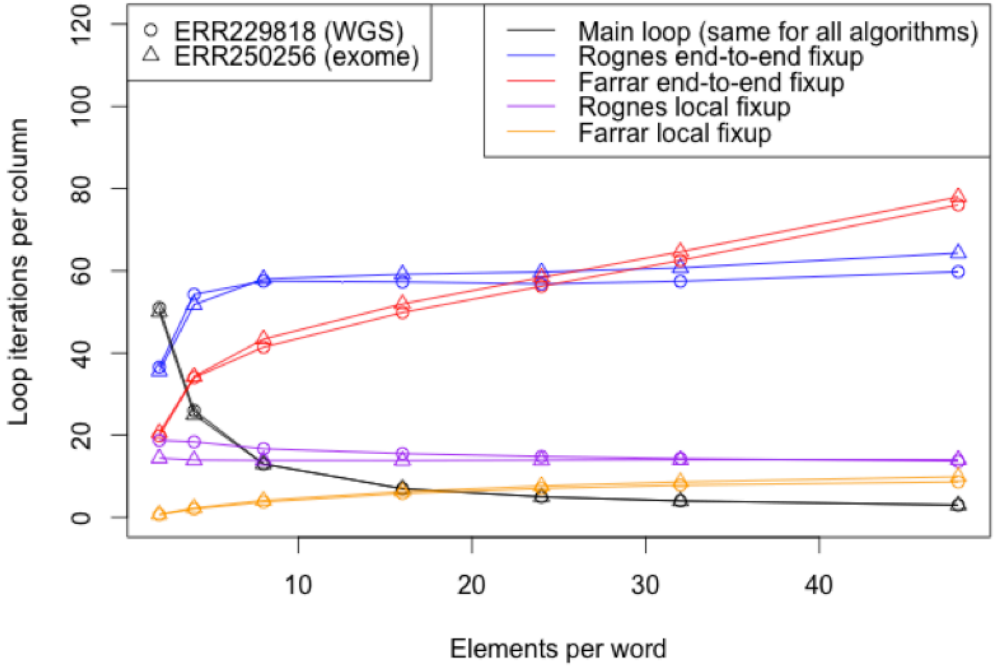
Fixup loop adjusts, taking intra-chunk dependencies into account

Farrar M. **Striped Smith-Waterman speeds database searches six times over other SIMD implementations.** *Bioinformatics*. 2007 Jan 15;23(2):156-61.

Vectorization of Bowtie 2 inner loop

The wider the vector word, the more times the fixup loop iterates

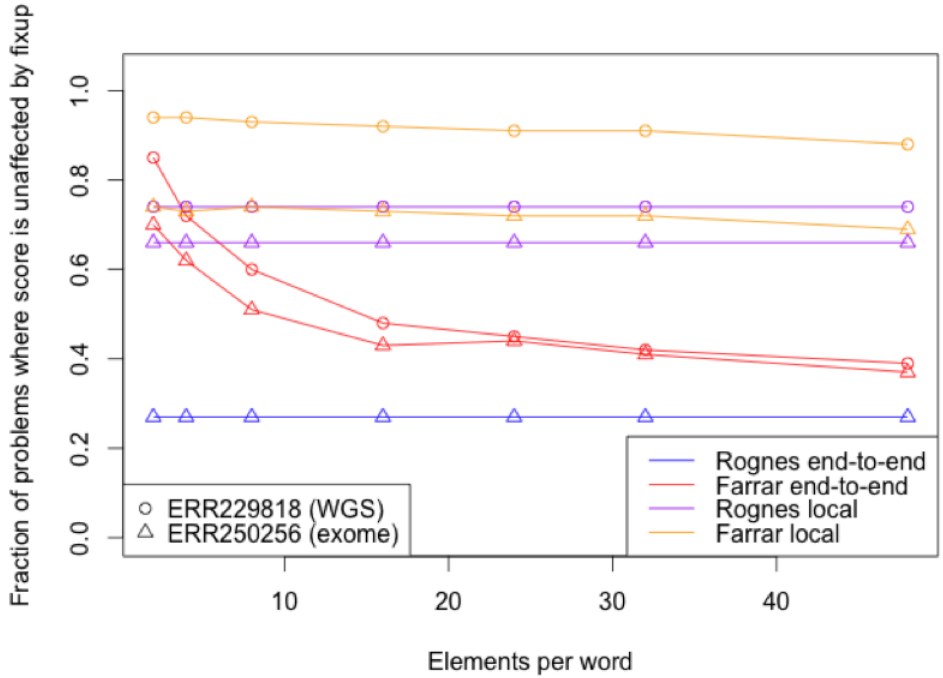
- Mitigates the benefit of having wider words



Vectorization of Bowtie 2 inner loop

…but in some situations, the fixup loop can be skipped with little or no downside

- Important future work is to determine whether selective suppression of fixup loop can remove most or all of the downside of having wider words



Impact on the field

1.1.0 - 7/19/2014

- Added support for large and small indexes, removing 4-billion-nucleotide barrier. Bowtie can now be used with reference genomes of any size.
- No longer releasing 32-bit binaries. Simplified manual and Makefile accordingly.
- Phased out CygWin support.
- Improved efficiency of index files loading.
- Fixed a bug that made `bowtie-inspect` fail in some situations.
- (This release was briefly given version number 2.0.0, but we changed it to 1.1.0 to avoid confusion with Bowtie 2.)

1.0.1 release - 3/14/2014

- Improved index querying efficiency using "population count" instructions available since SSE4.2.
- Credits to the Intel(r) enabling team for performance optimizations included in this release. Thank you!

Bowtie on GitHub - 4/11/13

- Bowtie source now lives in a public GitHub repository.

1.0.0 release - 4/9/13

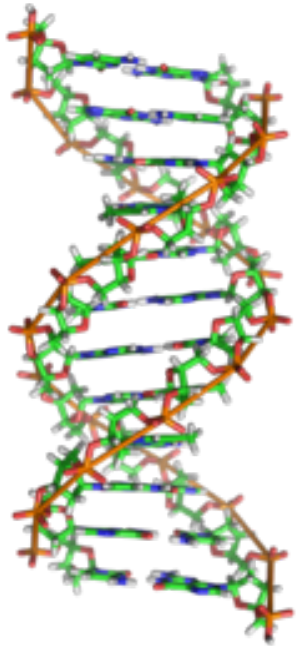
- Finally, a 64-bit Windows binary!
- Due to general performance improvements spinlocking is now used by default. The `EXTRA_FLAGS=-DNO_SPINLOCK` may be used to reverse this during

Bowtie 2: Fast, accurate read alignment
Crossbow: Genotyping, cloud computing
Tophat: RNA-Seq splice junction mapper
Cufflinks: Isoform assembly, quantitation
Myrna: Cloud, differential gene expression
Lighter: Fast error correction
Other tools using Bowtie

Pre-built indexes

Consider using Illumina's iGenomes collection. Each iGenomes archive contains pre-built Bowtie and Bowtie 2 indexes.

H. sapiens, UCSC hg18	**2.7 GB**
or: part 1 - 1.7 GB, part 2 - 1.0 GB	
colorspace: full, or part 1, part 2	
H. sapiens, UCSC hg19	**2.7 GB**
or: part 1 - 1.7 GB, part 2 - 1.0 GB	
colorspace: full, or part 1, part 2	
H. sapiens, NCBI v36	**2.7 GB**
or: part 1 - 1.7 GB, part 2 - 1.0 GB	



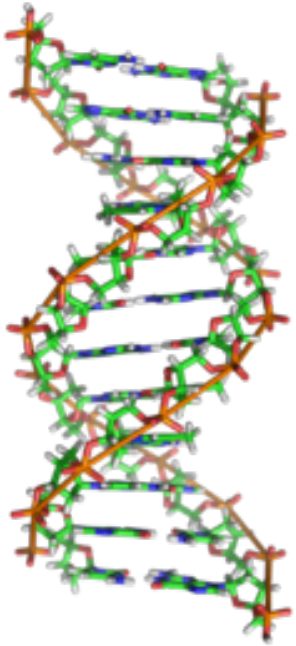
- As of Bowtie 1.0.1 release / Bowtie 2 2.2.0 release, Intel improvements are "in the wild," assisting life science researchers

Impact on the field

Recent news

1.1.2 - 6/23/2015

- Fixed the building process for Mac OS X Yosemite.
- Added `install` target (`make install`) for Linux to better aid package building process and the overall installation process.
- Added support for Intel TBB threading, providing better thread scaling in most situations. The default build still uses TinyThread but TBB is used with `make WITH_TBB=1`.
- Fixed minor issue related with managing the number of threads spawned.
- Fixed minor issue which may have caused a memory leak after an exception was thrown.
- Fixed bug that caused bowtie to crash if a read was trimmed more than the read's length on 5' end.
- Added minor corrections/addition to the manual.
- Fixed bug that caused the wrapper to incorrectly identify the bowtie binary.

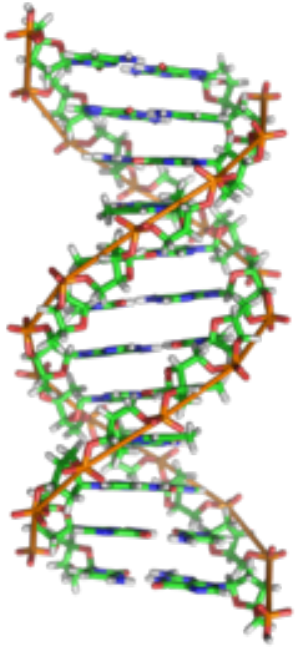


- Added TBB to Bowtie 1.1.2, Bowtie 2 2.2.6. Also added to public branch of HISAT. Plan to make TBB the default threading library in upcoming release.

Impact on the field

Version 2.2.7 - Feb 10, 2016

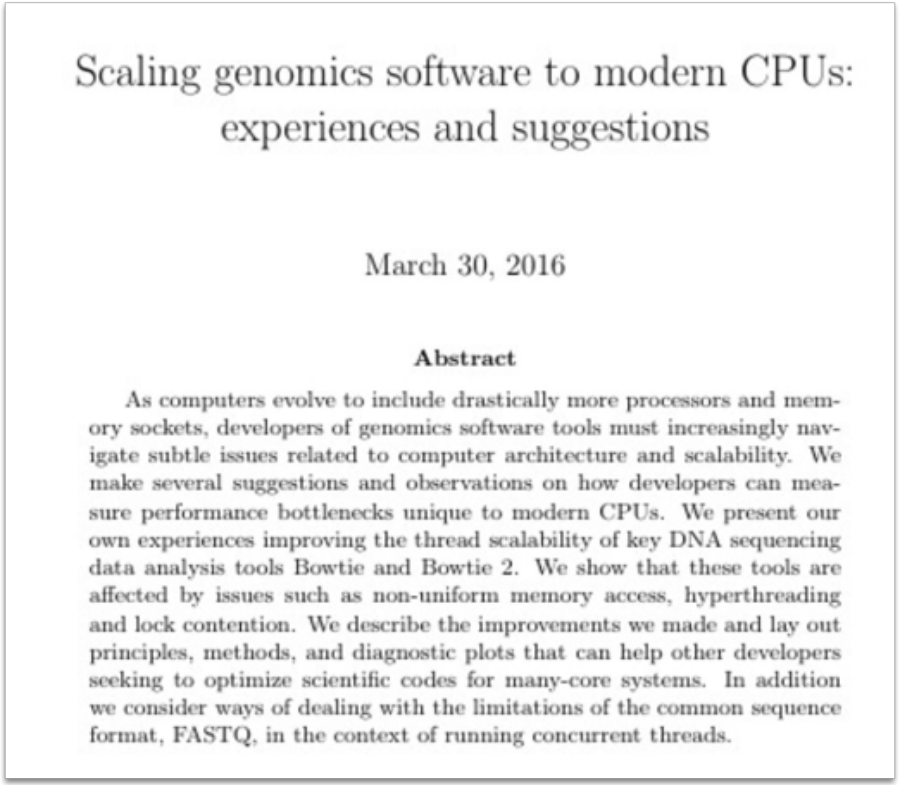
- Added a parallel index build option: bowtie2-build --threads <# threads>.
- Fixed an issue whereby IUPAC codes (other than A/C/G/T/N) in reads were converted to As. Now all non-A/C/G/T characters in reads become Ns.
- Fixed some compilation issues, including for the Intel C++ Compiler.
- Removed debugging code that could impede performance for many alignment threads.
- Fixed a few typos in documentation.



- Daehwan Kim of JHU IPCC team parallelized the index building process in Bowtie 2; TBB version of parallel index building available as of 2.2.7

Impact on the field

- With changes fully reflected in Bowtie 1.2.0 and Bowtie 2 2.3.0, JHU team drafting manuscript describing improvements and lessons learned



Scaling genomics software to modern CPUs: experiences and suggestions

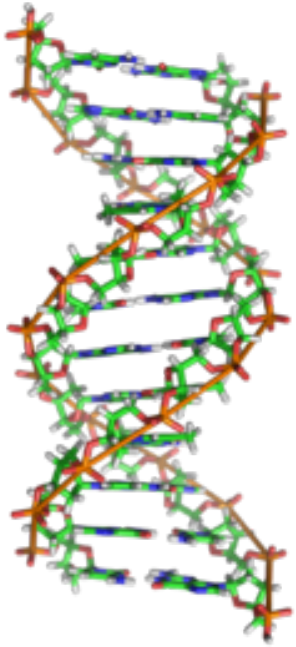
March 30, 2016

Abstract

As computers evolve to include drastically more processors and memory sockets, developers of genomics software tools must increasingly navigate subtle issues related to computer architecture and scalability. We make several suggestions and observations on how developers can measure performance bottlenecks unique to modern CPUs. We present our own experiences improving the thread scalability of key DNA sequencing data analysis tools Bowtie and Bowtie 2. We show that these tools are affected by issues such as non-uniform memory access, hyperthreading and lock contention. We describe the improvements we made and lay out principles, methods, and diagnostic plots that can help other developers seeking to optimize scientific codes for many-core systems. In addition we consider ways of dealing with the limitations of the common sequence format, FASTQ, in the context of running concurrent threads.

Future directions

- Where and why does the cohort lock help?
- Does cohort lock have a future in TBB?
- Can selective suppression of Bowtie 2 fixup loop unlock power of wider vector words?
- Can all of the above yield a big Knight's Landing throughput win?



Other resources

- http://www.langmead-lab.org
- https://www.coursera.org/learn/dna-sequencing
  - YouTube videos for above: http://bit.ly/ADS1_videos

Algorithms for DNA Sequencing

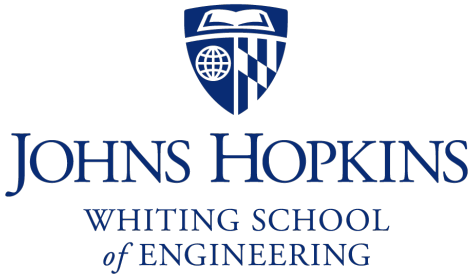
We will learn computational methods -- algorithms and data structures -- for analyzing DNA sequencing data. We will learn a little about DNA, genomics, and how DNA sequencing is used. We will use Python to implement key algorithms and data structures and to analyze real genomes and DNA sequencing datasets.

Ben Langmead, PhD
Johns Hopkins University

Jacob Pritt
Johns Hopkins University

Thank you

- John Oneill, Ram Ramanujam, Kevin O'leary, and many other great Intel engineers we spoke to and worked with
- Lisa Smith, Brian Napier and others in IPCC program
- Langmead lab team: Chris Wilks, Valentin Antonescu
- Salzberg lab team: Steven Salzberg, Daehwan Kim
- Intel



INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

Thank you for your time

Ben Langmead

langmea@cs.jhu.edu

www.intel.com/hpcdevcon