PASH 2.0: SCALEABLE SEQUENCE ANCHORING FOR NEXT-GENERATION SEQUENCING TECHNOLOGIES

CRISTIAN COARFA

[†]*Human Genome Sequencing Center, Department of Molecular and Human Genetics,*
Baylor College of Medicine, One Baylor Plaza
Houston, Texas 77030, USA

ALEKSANDAR MILOSAVLJEVIC

Human Genome Sequencing Center, Department of Molecular and Human Genetics,
Baylor College of Medicine, One Baylor Plaza
Houston, Texas 77030, USA

Many applications of next-generation sequencing technologies involve anchoring of a sequence fragment or a tag onto a corresponding position on a reference genome assembly. Positional Hashing method, implemented in the Pash 2.0 program, is specifically designed for the task of high-volume anchoring. In this article we present multi-diagonal gapped kmer collation and other improvements introduced in Pash 2.0 that further improve accuracy and speed of Positional Hashing. The goal of this article is to show that gapped kmer matching with cross-diagonal collation suffices for anchoring across close evolutionary distances and for the purpose of human resequencing. We propose a benchmark for evaluating the performance of anchoring programs that captures key parameters in specific applications, including duplicative structure of genomes of humans and other species. We demonstrate speedups of up to tenfold in large-scale anchoring experiments achieved by PASH 2.0 when compared to BLAT, another similarity search program frequently used for anchoring.

1. Introduction

Next generation sequencing technologies produce an unprecedented number of sequence fragments in the 20-300 basepair range. Many applications of next-generation sequencing require anchoring of these fragments onto a reference sequence, which involves comparison of these fragments to determine their position in the reference. Anchoring is required for the purpose of various mapping applications or for comparative sequence assembly (also referred to as comparative genome assembly and templated assembly). Anchoring is also a key step in the comparison of assembled evolutionarily related genomes. Due to the sheer number of fragments produced by next-generation sequencing technologies

†This research was partially supported by the National Human Genome Research Institute grant 5R01HG004009-02, by the National Cancer Institute grant 1R33CA114151-01A1 and the National Science Foundation grant CNS 0420984 to AM.

and the size of reference sequences, anchoring is rapidly becoming a computational bottleneck.

The de facto dominant paradigm for similarity search is that of "Seed-and-Extend" embodied in algorithms such as BLAST [1, 2], BLAT [3], SSAHA [4], PatternHunter [5, 6]. and FASTA [7, 8]. While not initially motivated by the anchoring problem, the Seed-and-Extend paradigm is employed by most current anchoring programs. We recently proposed Positional Hashing, a novel, inherently parallelizable and scaleable approach to specifically address the requirements of high-volume anchoring [9]. We first review key concepts behind Positional Hashing; then, we present the Pash 2.0 program, a new implementation which overcomes a number of deficiencies in the initial implementation of Positional Hashing. Pash 2.0 includes multidiagonal collation of gapped kmer matches to enhance accuracy in the presence of indels, and improvements that enhance speed when mapping large volumes of reads onto mammalian-sized genomes. The goal of this article is to show that gapped kmer matching with cross-diagonal collation suffices for anchoring across close evolutionary distances and for the purpose of human resequencing. To demonstrate this, we evaluate Pash by comparing its accuracy and speed against Blat, a Seed-and-Extend program that is widely used for anchoring. We determine parameters for Pash such it achieves comparable accuracy with Blat while providing several-fold speedups by avoiding basepair-level computation performed by Blat. To complement real-data experiments, we propose a simulation benchmark for evaluating performance of anchoring programs that captures key parameters in specific applications, including duplicative structure of the genomes such as that of humans. Using both real data and the simulation benchmark, we demonstrate speedups of up to tenfold without significant loss of sensitivity or accuracy in large-scale anchoring experiments when compared to BLAT.

2. Two approaches to anchoring: Seed-and-Extend vs. Positional Hashing

2.1. *The seed-and-extend paradigm*

The seed-and-extend paradigm currently dominates the field of sequence similarity search [2, 3, 4, 5, 6, 7, 10, 11]. This paradigm originally emerged to address the key problem of searching a large database using a relatively short query to detect remote homologies. A homology match to a gene of known function was used to derive a hypothesis about the function of the query sequence. The first key requirement for this application is sensitivity when

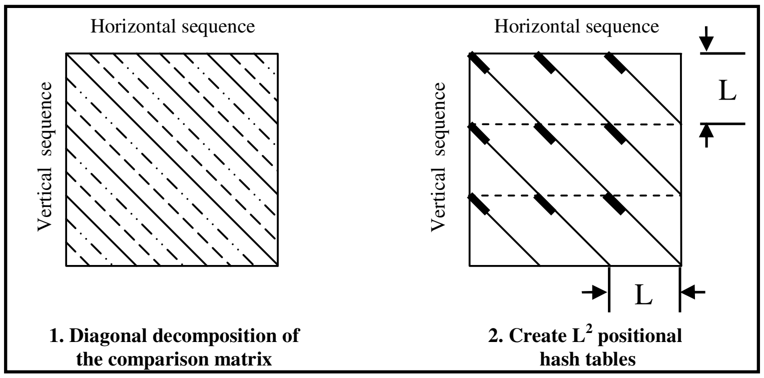

Horizontal sequence
Vertical sequence
Horizontal sequence
Vertical sequence
L
L
1. Diagonal decomposition of the comparison matrix
2. Create L^2 positional hash tables

Figure 1. Positional Hashing. 1. The positional hashing scheme breaks the anchoring problem along the L diagonals of the comparison matrix; each cluster node detects and groups matches along a subset of the L diagonals. 2. Each diagonal is split into horizontal and vertical windows of size L. Short bold lines indicate positions used to calculate hash keys for positional hash table H(0,0)

comparing sequences across large evolutionary distances. The second key requirement is speed when searching a large database using a short query. The first generation seed-and-extend algorithms such as BLAST [2] and FASTA [7] employed pre-processing of the query to speed up the database search while second-generation seed-and-extend algorithms such as BLAT [3] and SSAHA [4] employed in-memory indexing of genome-sized databases for another order of magnitude of speed increase required for interactive lookup of genome loci in human genome browsers using genomic DNA sequence queries.

2.2. *Positional Hashing specifically addresses the anchoring problem*

It is important to note that the anchoring problem poses a new and unique set of requirements. First, the detection of remote homologies is less relevant for anchoring than discrimination of true orthology relations when comparing closely related genomes. Second, with the growth of the genome databases and the emergence of next-generation sequencing technologies the query itself may now contain tens of millions of fragments or several gigabases of assembled sequence. To address the requirements specific to the anchoring problem, we recently developed the Positional Hashing method [9]. The method avoids costly basepair-level matching by employing faster and more scaleable gapped kmer matching [2,5,6,9]; this is performed using distributed position-specific hash tables that are constructed from both compared sequences.

To better formulate the difference between Positional Hashing and the classical Seed-and-Extend paradigms, we first introduce a few definitions. A “seed” pattern P is defined by offsets $\{x_1,\ldots,x_w\}$. We say that a “seed” match—a gapped kmer match where k equals w-- is detected between sequences S and T in respective positions i and j if $S[i+x_1]= T[j+x_1]$, …, and $S[i+x_w]=T[j+x_w]$. To further simplify notation, we define pattern function f_P at position i in sequence S as $f_P(S,i) = S[i+x_1]…S[i+x_w]$. Using this definition, we say that a "seed" match is detected between sequences S and T in respective positions i and j if $f_P(S,i)= f_P(T,j)$. A Seed-and-Extend method extends each seed match by local basepair alignment. The alignments that do not produce scores above a threshold of significance are discarded.

In contrast to the Seed-and-Extend paradigm, Positional Hashing groups all collinear matches—i.e., those falling along the same diagonal or, in Pash 2.0, a set of neighboring diagonals in the comparison matrix-- to produce a score. The score calculated by grouping the matches suffices for a wide range of anchoring applications, while providing significant speedup by eliminating the time-consuming local alignment at the basepair level. In further contrast to the Seed-and-Extend paradigm, Positional Hashing involves numerous position-specific hash tables, thus allowing extreme scalability through parallel computing. The positional hashing scheme breaks the anchoring problem along its natural diagonal structure, as illustrated in the Figure 1.1. Each node detects and groups matches along a subset of diagonals. More precisely, matches along diagonal d=0,1,…L-1, of the form $f_P(S,i)= f_P(T,j)$, where i=j+d (mod L) are detected and grouped in parallel on individual nodes of a computer cluster. Position-specific hash tables are defined by conceptually dividing each alignment diagonal into

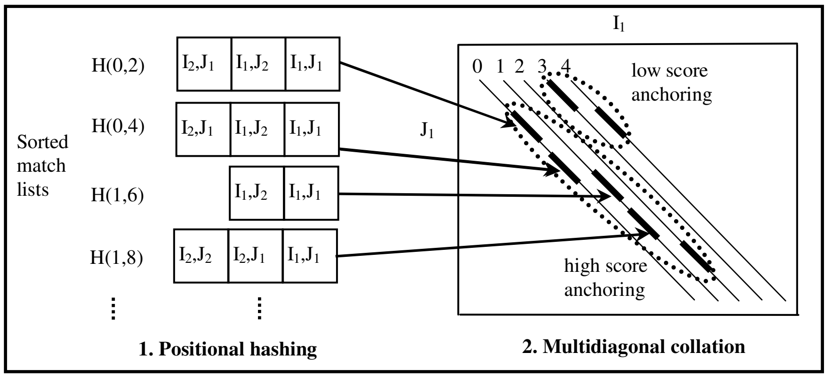


Figure 2. Positional hashing and multi-diagonal collation. 1. Lists of match positions for diagonals 0-5 induced by the appropriate hash tables are generated in the inversion step, for horizontal windows I_1 and I_2 and for vertical windows J_1 and J_2; the lists are sorted from right to left. A priority queue is used to quickly select the set of match positions within the same horizontal and vertical L-sized window, on which multidiagonal collation needs to be performed. 2. A greedy heuristic is used to determine the highest scoring anchoring across multiple diagonals; in the figure we depict matches within horizontal window I_1 and vertical window J_1, across diagonals 0-4.

non-overlapping *windows* of length L, as indicated by dashed lines in Figure 1.2. A total of L^2 positional hash tables H(d,k) are constructed for each diagonal d=0,1,…L-1 and diagonal position k=0,1,… L-1. Matches are detected by using the values of $f_P(S,i)$ and $f_P(T,j)$ as keys for storing horizontal and vertical window indices I=[i/L] and J=[j/L] into specific hash table bins. A match of the form $f_P(S,i)= f_P(T,j)$ where i=j+d (mod L) and j=k (mod L) is detected whenever I and J occur in the same bin of hash table H(d,k), as also shown in Figure 2.1. Further implementation details are described in [9].

3. Improved implementation of Positional Hashing

3.1. *Multidiagonal collation*

A key step in Pash is represented by the collation of matching kmers across diagonals. In Pash 1.0, collation was performed across a single diagonal only; an indel would split matching kmers across two or more neighboring diagonals. For Sanger reads, typically 600-800 base pairs long, Pash 1.0 could find enough information on either side of an indel to accurately anchor a read. For the shorter reads, generated by the next generation sequencing technologies, it might not be possible to find matching kmers on either side of an indel to anchor the read. The use of pyrosequencing, which causes insertion/deletion errors in the presence of homopolymer runs, further amplified this problem.

To overcome the problem, Pash 2.0 collates kmer matches across multiple diagonals. Pash detects similarities between two sequences, denoted a vertical sequence and a horizontal sequence (as indicated in Figure 1). After performing hashing and inversion for multiple diagonals, Pash generates one list of horizontal and vertical sequence positions of the matching kmers for each diagonal and positional hash table pair; these lists are sorted by the horizontal then by the vertical position of the matching kmer. Next, Pash considers simultaneously all lists of matching kmers for the set of diagonals that are being collated, and traverses them to determine all the matching positions between a horizontal and vertical window of size L (see Figure 2.1). To collate across k diagonals, Pash first selects matching positions across the same vertical and horizontal window from the kL lists of matching kmer positions. It uses a priority queue, with a two-part key: first the horizontal positions are compared, followed by the vertical position of matches, as shown in Figure 2.1. Kmers in each such set are collated, by performing banded alignment not at basepair level but at the kmer level. We used a greedy method to collate the matches across a diagonal set, and select the highest scoring match, as shown in Figure 2.2. By collating kmers across *k* diagonals, Pash is in effect anchoring across indels of size *k-1*; a user can control through command-line parameters the maximum indel size detectable by Pash. Pash 2.0 scores matches across indels using an affine indel penalty. Let *m* be the number of matching bases; for each indel *I* let *s(I)* be the indel length. The score of an anchoring is then $2m - \sum_{I} (s(I) + 1)$.

3.2. *Efficient hashing and inversion*

Pash version 1.0 was hashing both the vertical and the horizontal sequence. For comparisons against large genomes, such as mammalian genomes, hashing the whole genome during the hashing/inversion phase required significant time and memory. In Pash 2.0, only one of the sequences is hashed, namely the vertical sequence. For the horizontal sequence, instead of hashing it, Pash 2.0 traverses the horizontal kmer lists and then matches each kmer against the corresponding bin in the hash table created by hashing the vertical sequence. If a match is detected, the corresponding kmer is added to the list of matching kmers prior to proceeding to the next horizontal kmer. This improvement substantially accelerated the hashing and inversion steps.

4. Experimental Evaluation

Our experimental platform consisted of compute nodes with dual 2.2GHz AMD Opteron processors, 4GB of memory, running Linux, kernel 2.6. We used Pash 2.0, and BLAT Client/Server version 32. All experiments were run sequentially; when input was split in multiple chunks, we reported total compute time. The focus of this section is on comparing Pash 2.0 to Blat. When comparing Pash 2.0 against Pash 1.2, we determined overall speed improvements of 33%, similar accuracy for Sanger reads, and significant accuracy improvements for pyrosequencing reads. For Pash 2.0 we used the the following pattern of weight 13 and span 21: *111011011000110101011*. Code and licenses for Pash, Positional Hashing, and auxiliary scripts are available free of charge for academic use. Current access and licensing information is posted at http://www.brl.bcm.tmc.edu/.

4.1. *UD-CSD benchmark*

The choice of a program for an anchoring application depends on a number of data parameters, data volume, and computational resources available for the task. To facilitate selection of the most suitable program it would therefore be useful to test candidates on a benchmark that captures key aspects of the problem at hand. Toward this end, we developed a benchmark that includes segmental duplications, an important feature of mammalian genomes, and particularly of the genome of humans and other primates. The duplications are especially challenging because they limit the sequence uniqueness necessary for anchoring. The UD-CSD benchmark is named after five key aspects: *Unique* fraction of the genome; *Duplicated* fraction; *Coevolution* of duplicated fraction during which uniqueness is gradually developed; *Speciation*; and *Divergence* of orthologous reads. As illustrated in Figure 3, the UD-CSD benchmark is parameterized by the following four parameters: number of unique reads k; number of duplicated reads n; coevolution parameter *x;* and divergence parameter *y*; we are in fact simulating genomes as a concatenation of reads. For example, the divergence parameter y=1% may be appropriate for human-chimpanzee anchoring and y=5% anchoring of a rhesus monkey onto human. Note that in a human genome resequencing study, the divergence parameter y would be set to a very small value due to relatively small amount human polymorphism but the duplicative structure of the human genome could be captured using remaining three parameters.

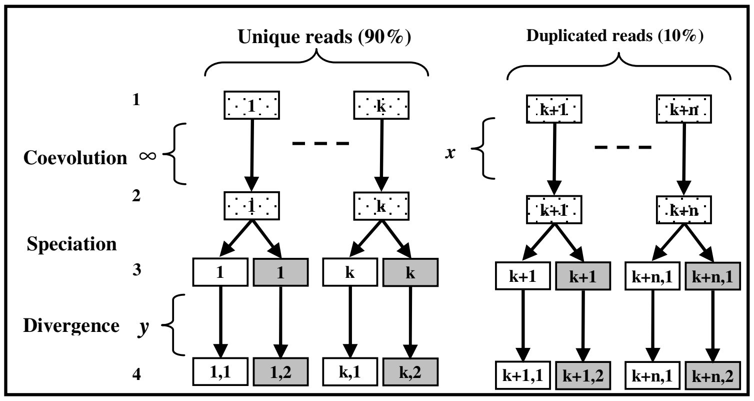
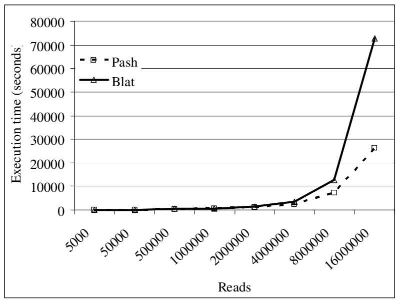
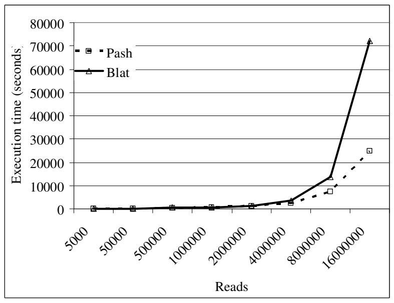


Figure 3. The *UD-CSD* (*Unique,Duplicated-Coevolution,Speciation,Divergence*) Anchoring Benchmark. 1. Randomly generate k *Unique* reads and n *Duplicated* reads. 2. *Coevolution*: each base mutates with probability x. 3. *Speciation:* Each read duplicates. 4. *Divergence*: each base mutates with probability y.

Using the UD-CSD benchmark, we evaluated the sensitivity and specificity of Pash compared to BLAT, a widely used seed-and-extend comparison algorithm. We first generated k+1 random reads of size m base pairs, then we duplicated the last read n-1 times, as illustrated in Figure 3.1, and obtained seed reads s_i, i=1,n+k. This corresponds to a genome where the k reads represent unique regions, and the n duplicated reads represent duplicated regions. Next, we evolved each read s_i, such that each base has a mutation probability of x, and each base was mutated at most once, and obtained reads r_i, i=1,n+k. Out of the mutations, 5% were indels, with half insertions and half deletions; the indel lengths were chosen using a geometric probability distribution with the parameter p=0.9, and imposing a maximum length of 10. The remaining mutations were substitutions. This process approximates a period of *coevolution* of two related species during which duplicated regions acquire uniqueness (parameterized by *x*) necessary for anchoring. Next, two copies of each read were generated, and one assigned to each of two simulated genomes of descendant species, as shown in Figure 3.3; this corresponds to a speciation event. Subsequently, each read evolved independently such that each base had a mutation probability of y, as illustrated in Figure 3.3; this corresponds to a period of *divergence* between the two related species. Finally, we obtained the set of reads $r_{i,1}$ and $r_{i,2}$, with i=1,n+k. We then employed Pash and BLAT to anchor the read set $\{r_{1,1},\ldots,r_{n+k,1}\}$ onto $\{r_{1,2},\ldots,r_{n+k,2}\}$, by running each program and then filtering its output such that only the top ten best matches for each read are kept. Any time a read $r_{i,1}$ is matched onto $r_{i,2}$, we consider this a true positive; we count how many true positives are found to evaluate the accuracy of the anchoring program.

One may raise objection to our considering the top ten best matches and may instead insist that only the top match counts. Our more relaxed criterion is justified by the fact that anchoring typically involves a reciprocal-best-match step. For example, a 10-reciprocal-best-match step would sieve out false matches and achieve specific anchoring as long as the correct match is among the top 10 scoring reads. Assuming random error, one may show that the expected number of false matches would remain constant (10 in our case) irrespective of the total number of reads matched. For our experiment, we chose a read length of 200 bases, and varied the total number of reads from 5,000 to 16,000,000. k and n were always chosen such that 90% of the start reads were unique, and 10% were





1 25% coevolution , 1% divergence   2. 25% coevolution, 5% divergence

Figure 4.1.Anchoring times of Pash and BLAT for coevolution of 25% and divergence of 1%, which may be appropriate for comparing closely related primates such as chimpanzee and human. 2.Anchoring times of Pash and BLAT for coevolution of 25% and divergence of 5%, which may be appropriate for comparing a New World monkey such as rhesus macaque and human.

repetitive. In Figure 4.1 we present the execution times for Pash and BLAT for 25% coevolution and 1% divergence, while in Figure 4.2 we present execution times for Pash and BLAT for 25% coevolution and 5% divergence. Pash was run using a gapped pattern of weight 13 and span 21, and a kmer offset gap of 12, while for BLAT we used the default settings.

In both cases, Pash and BLAT achieve comparable sensitivity (the numbers mate pairs found are within 1% of each other). This result is significant because it indicates that time-consuming basepair-level alignments performed by BLAT are not necessary for accurate anchoring – kmer-level matching performed by Pash suffices. For up to 2 million reads, Pash and BLAT achieve comparable performance. When the number of reads increases to 4, 8, and 16 million reads, however, Pash outperforms BLAT by a factor of 1.5 to 2.7.

4.2. ***Simulated Anchoring of WGS reads***

Next generation technologies enable the rapid collection of a large volume of reads, which can then be used for applications such as genome variation detection. A key step is the anchoring of such reads onto the human genome. In our experiment, we used reads obtained by randomly sampling the human genome (UCSC hg18, http://genome.ucsc.edu/downloads.html) with read sizes chosen according to the empirical distribution of read lengths observed in sequencing experiments using 454 sequencing technology. The set of reads covering the human genome at 6x sequence coverage was independently mapped back onto the reference genome using Blat and Pash. Pash anchored 73 million reads in 160 hours, using kmers of weight 13, span 21, and kmer gap offset of 12. Blat was run with default parameters; it mapped the reads from chromosomes 1 and 2 in 289 hours; this extrapolates to an overall running time of 1823 hours, for a 11.3 fold acceleration of Pash over Blat; Blat mapped only 0.3 percent more reads than Pash; this difference is caused by reads that Pash did not map because of its own default ignoring overrepresented kmers; we could improve this figure by increasing Pash's tolerance for overrepresented kmers. Next, we extracted tags of 25 base pairs from each simulated WGS read, and mapped them on the human genome using Pash and Blat. Pash anchored the tags from chromosomes 1 and 2 in 4.5 hours, while Blat anchored them in 105 hours. However, with default parameters Blat does not perform well for the 25 base pair tags, anchoring back correctly 28% of the tags for chromosome 1 and 31% for chromosome 2, compared to 77% and 85% respectively for Pash.

4.3. *Anchoring of mate pairs*

Sequenced ends of a small-insert or a long-insert clone such as a Fosmids or a Bacterial Artificial Chromosome (BAC) may be anchored onto a related reference genomic sequence. Numerous biological applications rely on this step, such as detection of cross-mammalian conservation of chromosome structure using mapping of sequenced BAC-End Sequences [13,14,15] and reconstruction of the evolution of the human genome [12]. Next-generation sequencing technologies provide a particularly economical and fast method of delineating conserved and rearranged regions using the paired-end method.
The fraction of consistently anchored paired end-sequences from a particular set depends on the accuracy of the anchoring program, making this a natural benchmark for testing anchoring programs. We obtained about 16 million Sanger reads from fosmid end sequences in the NCBI Trace Archive, for a total of 7,946,887 mate pairs, and anchored them onto the human genome with Blat and Pash 2.0. For each read we selected the top 10 matches, then looked for consistently mapped mate pairs. We counted the total number of clone ends that were anchored at a distance consistent with clone insert size (25-50 Kb) and computed their percentage of the expected number of mate pairs. Since anchoring performance also depends on the size of anchored reads, we also simulated five shorter read sizes by extracting 250bp, 100bp, 50bp, 36bp, and 25bp reads respectively from each Sanger read, generating additional sets of simulated short fosmid end sequences. We anchored each of the short read sets onto the human genome, then determined the number of clone ends consistently mapped. We summarize the results of our experiment in Table 1. We used gapped kmers of weight 13 and span 21, and kmer offsets of 12 for Sanger and 250 bp reads, of 6 for 100 bp reads, and of 4 for 50, 36, and 25 bp reads. As evident from Table 1, in all the experiments both Pash and BLAT found a comparable number of consistent mate pairs mapping, while Pash ran 4.5 to 10.2 times faster compared to BLAT. A recent option added to Blat is that of fastMap, which enables rapid mapping of queries onto highly similar targets.

Table 1. Summary of results for actual and simulated mate pair anchoring

Read Type	Pash execution time (hrs)	Percent of matepairs found	Blat execution time (hrs)	Percent of matepairs found
Sanger	102	76%	1045	76%
250 bp	45	76%	421	76%
100 bp	23	75%	154	75%
50 bp	17	68 %	92	68 %
36 bp	8	57%	85	58%
25 bp	4	56 %	154	15 %

We ran Blat with this option, but determined that it yielded very low sensitivity compared to Blat with default parameters, retrieving around 1 percent of the total number of matepairs; we argue the Blat with fastMap is not a good choice for this task. Blat with default parameters performs poorly on 25bp reads.

Pash 2.0 accelerates anchoring the most for very large input data sets. To measure this effect, we partitioned our input of 16 million reads into chunks of 0.5, 1, 2, 4, and 8 million reads each and run Pash on the whole input, computing average time per chunk. Each chunk could be run on a separate cluster node, and the parallel Pash wall time would be the maximum execution time of an input chunk. In Figure 5 we present the Pash execution time per chunk and the overall running time; our results show that while our method has a significant overhead for a small number of reads, its effectiveness improves as the number of input reads per chunk is increased. Pash 2.0 is therefore suitable for anchoring the output of high-volume, high-throughput sequencing technologies.

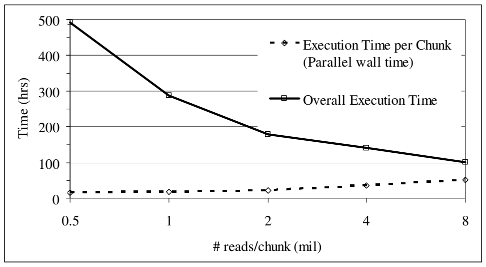


Figure 5. Anchoring time for 16 million Sanger reads onto human genome.

5. Conclusions

We demonstrate that by avoiding basepair-level comparison the Positional Hashing method accelerates sequence anchoring, a key computational step in many applications of next-generation sequencing technologies, over a large spectrum of read sizes -- from 25 to 1000 base pairs. Pash shows similar sensitivity to state-of-the-art alignment tools such as BLAT on longer reads and outperforms BLAT on very short reads, while achieving an order of magnitude speed improvement. Pash 2.0 overcomes a major limitation of previous implementations of Positional hashing, sensitivity to indels, by performing cross-diagonal collation of kmer matches. A future direction is to exploit multi-core hardware architectures by leveraging the low-level parallelism; another direction is to further optimize anchoring performance in the context of pipelines for comparative sequence assembly and other specific applications of next-generation sequencing.

Acknowledgments

We thank Andrew Jackson, Alan Harris, Yufeng Shen, and Ken Kalafus for their help.

References

1. Altschul, S.F., et al., *Basic local alignment search tool.* J Mol Biol, 1990. 215(3): p. 403-10.
2. Altschul, S.F., et al., *Gapped BLAST and PSI-BLAST: a new generation of protein database search programs.* Nucleic Acids Research, 1997. 25(17): p. 3389-402.
3. Kent, W.J., *BLAT--the BLAST-like alignment tool*, in *Genome Res*. 2002. p. 656-64.
4. Ning, Z., A.J. Cox, and J.C. Mullikin, *SSAHA: a fast search method for large DNA databases.* Genome Research, 2001. 11(10): p. 1725-9.
5. Ma, B., J. Tromp, and M. Li, *PatternHunter: faster and more sensitive homology search.* Bioinformatics, 2002. 18(3): p. 440-5.
6. Li, M., et al., *PatternHunter II: Highly Sensitive and Fast Homology Search.* Journal of Bioinformatics and Computational Biology, 2004. 2(3): p. 417-439.
7. Pearson, W.R. and D.J. Lipman, *Improved tools for biological sequence comparison.* Proc Natl Acad Sci U S A, 1988. 85(8): p. 2444-8.
8. Pearson, W.R., *Rapid and sensitive sequence comparison with FASTP and FASTA.* Methods Enzymol, 1990. 183: p. 63-98.
9. Kalafus, K.J., A.R. Jackson, and A. Milosavljevic, *Pash: Efficient Genome-Scale Sequence Anchoring by Positional Hashing.* Genome Research, 2004. 14: p. 672-678.
10. *WU-BLAST*. 2007.
11. Schwartz, S., et al., *Human-mouse alignments with BLASTZ.* Genome Res, 2003. 13(1): p. 103-7.
12. Harris, R.A., J. Rogers, and A. Milosavljevic, *Human-specific changes of genome structure detected by genomic triangulation.* Science, 2007. 316(5822): p. 235-237.
13. Fujiyama, A., et al., *Construction and analysis of a human-chimpanzee comparative clone map.* Science, 2002. 295(5552): p. 131-4.
14. Larkin, D.M., et al., *A Cattle-Human Comparative Map Built with Cattle BAC-Ends and Human Genome Sequence.* Genome Res, 2003. 13(8): p. 1966-72.
15. Poulsen, T.S. and H.E. Johnsen, *BAC end sequencing.* Methods Mol Biol, 2004. 255: p. 157-61.