

TECHNISCHE UNIVERSITÄT DRESDEN

CENTER FOR INFORMATION SERVICES
& HIGH PERFORMANCE COMPUTING
PROF. DR. WOLFGANG E. NAGEL

Performance Tuning & Parallelisation of Inchworm

Ankur Sharma

Dresden, March 23, 2014

Contents

1	Introduction	3
1.1	Biological Background	3
1.1.1	Sequence Assembly	5
1.1.2	Algorithm	5
1.2	Technical Background	5
2	Trinity	5
2.1	Structure	5
2.2	Phases	5
2.2.1	0 - Jellyfish	5
2.2.2	1 - Inchworm	5
2.2.3	2 - Chrysalis	5
2.2.4	3 - Butterfly	5
2.3	Applications	5
3	Inchworm	5
3.1	Phases	5
3.1.1	Parsing	5
3.1.2	Sorting	5
3.1.3	Pruning	5
3.1.4	Assembling	5
3.2	Implementation	5
3.3	Performance Bottlenecks	5
3.4	Conclusion	5
4	Performance Optimisation	5
4.1	Parsing	5
4.1.1	Memory Mapped IO	5
4.1.2	Benefits	5
4.1.3	How it is used?	5
4.1.4	Impact on Inchworm	5
4.2	Sorting	5
4.3	Pruning	5
4.3.1	Parallel Pruning	5
4.3.2	Performance Gain	5
4.4	Assembly	5
4.4.1	Bottlenecks	5
4.4.2	Parallel Assembling	5
4.4.3	Benefits	5
4.4.4	Impact on inchworm results	5
5	Conclusion	5
5.1	Future work	5

1 Introduction

The high demand for low-cost sequencing has driven the development of high-throughput sequencing, which is also termed as Next generation sequencing (NGS). Thousands or millions of sequences concurrently produced in next-generation sequencing process. RNA-sequencing is a technology that uses the capabilities of next-generation sequencing to reveal the snapshot of RNA presence and quality from a genome at given moment in time. A number of assembly programs are available. Although these programs have been generally successful in assembling genomes, transcriptome assembly presents some unique challenges. Whereas high sequence coverage for a genome may indicate the presence of repetitive sequences, for a transcriptome, they may indicate abundance. Trinity is one of those assemblers that has superior quality and represents a novel method for efficient and robust de novo reconstruction of transcriptomes from RNA-seq data. Careful performance analysis of *Trinity* software demonstrated that few of the components of the application specially *Inchworm* can be manipulated in order to achieve high performance gain. This document thus discusses the performance optimisations and parallel master slave approach of computing sequence assemblies deployed in *Inchworm* that boosted the performance to a great extent.

1.1 Biological Background

The detailed analysis of *Inchworm* involves familiarity with a lot of biological terms and techniques. This section discusses some of those key biological elements that play crucial role in better understanding of implementation and a deeper insight of how the changes in the assembly algorithm is effecting the final results.

1.1.1 Sequence Assembly

The core of *Inchworm* is in the assembly that is deployed presently using a simple but very efficient greedy algorithm. In order to extend a kmer, the algorithm simply chooses a kmer with k-1 overlap having highest count from the catalog and continues it until the extension is possible.

1.1.2 Algorithm

1.2 Technical Background

2 Trinity

2.1 Structure

2.2 Phases

2.2.1 0 - Jellyfish

2.2.2 1 - Inchworm

2.2.3 2 - Chrysalis

2.2.4 3 - Butterfly

2.3 Applications

3 Inchworm

3.1 Phases

3.1.1 Parsing

3.1.2 Sorting

3.1.3 Pruning

3.1.4 Assembling

3.2 Implementation

3.3 Performance Bottlenecks

3.4 Conclusion

4 Performance Optimisation

4.1 Parsing

4.1.1 Memory Mapped IO

4.1.2 Benefits

4.1.3 How it is used?

4.1.4 Impact on Inchworm

4.2 Sorting

4.3 Pruning

4.3.1 Parallel Pruning

4.3.2 Performance Gain

4.4 Assembly

4.4.1 Bottlenecks

4.4.2 Parallel Assembling

4.4.3 Benefits

4.4.4 Impact on inchworm results

5 Conclusion

5.1 Future work

