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

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 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.