A Just-In-Time Compiler for Intelligent Manufacturing *

C.A. Lakshman Naresh Xipeng Shen Binil Starly
North Carolina State University
{lcoimba,xshen5,bstarly}@ncsu.edu

Abstract

Conventional Computerized Numerical Control (CNC) machines are operated using G-Code part programs. These machines require constant monitoring and human interaction during machining process to detect and rectify abnormalities. Making changes to the machining process is time consuming and a tedious task because the CNC operator has to have prior knowledge of G-Code programming language. G-Code is a low-level and machine specific programming language. So, G-Code programs do not contain most of the information from CAD/CAM phase of the manufacturing process. And also a G-code program generated for a specific machine cannot be used in a machine manufactured by a different CNC vendor.

This work presents a framework that uses a machine independent programming language as input, does not require CNC operators to have prior knowledge of G-Code programming language, does not require human interaction during machining process, enables real-time remote monitoring of the machining process, performs optimization on the input to a CNC based on the results from the previous runs, allows users to perform their exploration on historical feedback data set, and allows users of the framework to write third-party applications and optimization algorithms that perform optimizations on the input.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

Keywords Just-In-Time Compiler, Manufacturing, G-code, STEP-NC, CNC, Intelligent Machining, FDO, API

1. Introduction

The first wave of Industrial Revolution began in the late 18th century. Currently, Industrial Revolution is in its fourth wave, characterized by a range of new technologies that are integrating the physical, digital and biological worlds, and impacting all disciplines, economies and industries. Fourth Industrial Revolution is also known as an era of **cyber-physical systems** (CPS). A CPS is a system composed of physical entities such as mechanisms controlled or monitored by computer-based algorithms. Today, a precursor generation of cyber-physical systems can be found in areas as diverse as aerospace, automotive, chemical processes, civil infrastructure, energy, healthcare, manufacturing, entertainment, consumer appliances, and transportation.

Cybermanufacturing is a concept derived from cyber-physical systems. Basically, it refers to a modern manufacturing system that offers an information-transparent environment to facilitate asset management, provides reconfigurability, and maintains productivity. The back-bone of cybermanufacturing is claimed to be the intelligent machines. The conventional approach to making a machine *cyber-enabled* is to outfit the machine with an array of multi-modal sensors which are then integrated to the network and enterprise above it. This approach will make development work cumbersome. Alternate approach is to use the feedback from the machines. Almost all industrial machine vendors have closed hardware and software architecture which leads to development work remaining inhouse. Vendor lock-in mode makes it difficult for outside software applications to be written to obtain the feedback data, potentially limiting fast-paced innovation in the cyber-physical manufacturing realm. Hence, this work presents a framework that acts as a middleware between the manufacturing system and the user or operator. This is similar to Android - a common middleware software platform for many of the leading smart-phone manufacturers.

Most of today's Computerized Numerical Control (CNC) machines are operated using G-code programming language because G-code is supported by most of the CNC vendors. The controller in the CNCs are closed; the internal operating method, current machine state, and sensor data are not exposed to the users of the CNCs. Vendor specific controllers make G-code the universally accepted input for CNCs. And also these controllers do not allow users to read the machine data or create their optimization model based on their exploration, and hinder the ability to remotely monitor the machining process, the sensor data, or the tool condition.

This paper presents a Just-In Time(JIT) Compiler for Intelligent Manufacturing framework that overcomes the aforementioned shortcomings of a controller in a vendor specific CNC. This system accepts a high-level language as input and a guideline file that specifies the optimizations to be performed on the input, uses an open-source controller to drive a CNC, performs optimizations on the input using the machining feedback data and sensor values from the previous runs, and exposes application programming interfaces (APIs) for machining feedback data, sensor values, and the guideline file. JIT Compiler for Intelligent Manufacturing framework enables users to remotely monitor the machining process by writing third-party applications using the APIs for the machining feedback data and sensor values, optimize the input by writing third-party optimization algorithms using the APIs for the guideline file, and explore the historical feedback data set using the APIs for the machining feedback data and sensor values.

The remainder of the paper is organized as follows. Section 2 describes the background for this work. Section 3 presents the Just-In-Time Compiler for Intelligent Manufacturing framework. Section 4 describes the design and implementation of Just-In-Time

* This material is based upon work supported by the National Science Foundation.

This work is based upon work supported by NSF 1547105. Any opinions,findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

Compiler for Intelligent Manufacturing prototype. Section 5 describes the related work. Section 6 describes the evaluation of the prototype. Section 7 presents our conclusion and possible future work.

2. Background

2.1 CAD/CAM

CAD/CAM is a term which means computer-aided design and computer-aided manufacturing. CAD involves creating computer models defined by geometrical parameters. These models typically appear on a computer monitor as a three-dimensional representation of a part or a system of parts, which can be readily altered by changing relevant parameters [3]. CAD output typically includes precise dimensions, tolerances, and even material requirements; CAD is frequently integrated with computer-aided engineering (CAE). Computer-aided manufacturing (CAM) is the use of software to control machine tools and related ones in the manufacturing of workpieces. CAM may also refer to the use of a computer to assist in all operations of a manufacturing plant, including planning, management, transportation and storage. Its primary purpose is to create a faster production process and components, and tooling with more precise dimensions and material consistency, which in some cases, uses only the required amount of raw material (thus minimizing waste), while simultaneously reducing energy consumption. CAM is a subsequent computer-aided process after CAD and sometimes CAE, as the model generated in CAD and verified in CAE can be input into CAM software, which then controls the machine tool[12].

2.2 CNC

Numerical control (NC) is the automation of machine tools that are operated by precisely programmed commands encoded on a storage medium, as opposed to controlled manually by hand wheels or levers. Most NC today is computer (or computerized) numerical control (CNC), in which computers play an integral part of the control[13]. The CAD/CAM programs produce a computer file that is interpreted to extract the commands needed to operate a particular CNC by use of a post processor, and then loaded into the CNCs for production. Post processor generates a variant of G-code that is accepted by the CNC. Since any particular component in a product might require the use of a number of different tools - drills, saws, etc., modern machines often combine multiple tools into a single ”cell”[13]. In a CNC, motion is controlled along multiple axes, normally at least two (X and Y), and a tool spindle that moves in the Z (depth). The position of the tool is driven by direct-drive stepper motor or servo motors in order to provide highly accurate movements, or in older designs, motors through a series of step down gears. The workpiece, the cutting tool, or both rotate rapidly, and the overall motion of the workpiece or the cutting tool is precisely controlled by stepper motors or other servo-type mechanisms, which are themselves controlled by the numerical control commands or part program. A CNC has feedback devices which updates the positional values and speed of the axes. This help CNC to operate accurately. When the machine is running, the Display Unit displays the present status such as the position of the machine slide, the spindle RPM, the feed rate, the part programs, etc. In an advanced CNC, the Display Unit can show the graphics simulation of the tool path so that part programs can be verified before the actual machining. Other important information about the CNC system are also displayed for maintenance and installation work such as machine parameters, logic diagram of the programmer controller, error massages and diagnostic data [7].

2.3 G-code Programming Language

G-code programming language is the conventional programming language of numerical controlled machine tools. ISO standard for G-code is ISO 6938. These codes were developed more than 50 years ago with little, if any, intelligence. The initial design of the codes was to hold a set of low-level data that are mostly step-by-step instructions to drive the earliest models of machine tools. This sequential programming language contains simple commands for single movement and switching operations but cannot support more complex geometries or logical structures. Since only limited control of the program execution is allowed during machining, it is difficult to change the program on the shop-floor. Information flow in G-codes was designed unidirectionally, i.e., from CAD to the shop-floor, and does not enable feedback of know-how from the shop-floor to the designer. As a result, this conventional way of NC programming is considered a bottleneck for achieving an intelligent machining environment[11]. There are several variations of G-codes and they are specific to a CNC vendor. These variants are generated by a CAM software using the specific post-processor for the variant. G-code is a machine specific and low-level programming language. G-code program generated for a specific CNC cannot be used in a CNC manufactured by a different CNC vendor. G-code has more than three thousand variations and post-processors.

2.4 Conventional Manufacturing Process

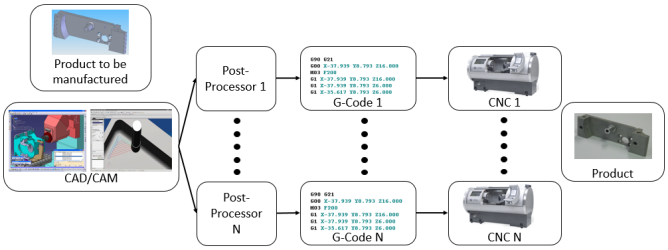


Figure 1: Conventional Manufacturing process.

Figure 1 depicts the conventional manufacturing process. A product is designed in a CAD software and detailed in CAM software. The output of the CAM software is sent to a post-processor to generate G-code part program for a specific CNC. The generated G-code part program is sent as input to a CNC, which reads the G-code part program and manufactures the product. The problem with this approach is that the G-code part programs have to be generated for each different type of CNCs on the shop-floor. If something were to go wrong during a manufacturing operation, the machine has to be stopped and the G-code part program has to be edited manually by the operator to reflect the correct manufacturing operation. Since all the machines on the shop-floor are manufacturing the same product and the G-codes are not the same for CNCs manufactured by different CNC vendors, operators have to stop the machines and manually edit the G-codes every time the product is manufactured. This is a cumbersome work and it will increase the production time. And also it requires the operators to have prior knowledge of the G-code programming language and the specific type of variants required for the CNCs on the shop-floor.

2.5 LinuxCNC

LinuxCNC is a free, open-source GNU/Linux software system that implements numerical control capability using general purpose computers to control CNCs. It can control up to 9 axes of a CNC using G-code as input. It has several GUIs suited to specific kinds of usage (touch screen, interactive development). Due

to the need of fine grained, precise real time control of machines in motion, LinuxCNC requires a platform with real-time computing capabilities. It uses linux kernel with real time extensions (RTAI) or with RT-PREEMPT kernel using LinuxCNC's 'uspace' flavor of RTAPI[14].

2.5.1 LinuxCNC Architecture Overview

LinuxCNC consist of four components: a motion controller (EMCMOT), a discrete IO controller (EMCIO), a task executor which coordinates them (EMCTASK) and several text-mode and graphical User Interfaces. At the coarsest level, LinuxCNC is a hierarchy of three controllers: the task level command handler and program interpreter, the motion controller, and the discrete I/O controller. The discrete I/O controller is implemented as a hierarchy of controllers for spindle, coolant, and auxiliary (e.g., estop, lube) subsystems. The task controller coordinates the actions of the motion and discrete I/O controllers. Their actions are programmed in conventional numerical control "G and M code" programs, which are interpreted by the task controller into Neutral Message Language (NML) messages and sent to either the motion or discrete I/O controllers at the appropriate times. The motion controller receives commands from user space modules via a shared memory buffer, and executes those commands in real-time. The status of the controller is made available to the user space modules through the same shared memory area. The motion controller interacts with the motors and other hardware using the Hardware Abstraction Layer (HAL)[5].

2.6 STEP-NC

STEP-NC (Standard for the Exchange of Product data for Numerical Control) is an ISO standard machine tool control language. It provides an opportunity to overcome the obstacles of G-code especially in realizing intelligent machining operations. The main characteristic of STEP-NC is its high-level and object-oriented data structure. Unlike G-code where a part program is written to describe simple tool movements and functions, the STEP-NC interface is able to work with rich information such as manufacturing features, multiple operations such as finishing and roughing processes, machine tool capability, motor drive power, mechanical efficiency, machining strategy, cutting tool information, and workpiece properties. Since STEP-NC data model describes rich information, quality knowledge and data can be utilized on the shop-floor, which enables advanced optimization analysis to be conducted. Modifications on the shop-floor are possible and machining know-how can be preserved for designers and process planners, thus improving the communication link between design and manufacturing departments. By providing a complete and structured data model, no information is lost. Post-processors for machine-specific adaptations of NC programs are no longer needed. In addition, this rich information content results in higher flexibility enabling last-minute changes or the correction of technological values within the part program [11].

2.6.1 STEP-NC Formats

STEP-NC has two standards that cover the field of STEP-NC: ISO 14649 and ISO 10303-238. Each of them covers its own portion of data exchange between different steps in product development and manufacturing. ISO 14649 is intended for applications where CAM software has total access to all the data from the production, whereas ISO 10303-238 is intended for total integration of CAD, CAM and manufacturing [4].

ISO 14649 describe working process with different working steps that represent higher level geometry elements combined with all the process parameters needed in order to manufacture the product instead of tool movement trajectory. NC controller is responsible for translating these working steps into actual tool movement. This data structure also allows using the same STEP-NC program on any type of machine without the use of post-processors. ISO 10303-238 data model is also know as AP238 data model. The main difference in ISO 10303-238 when compared to ISO 14649 is that this data model also includes a 3D CAD model of the product. It is basically an upgrade of an ISO 14649 standard. A large portion of AP238's data structure implementation was copied from previously released ISO 14649 [4].

2.7 ISO14649 toolkit

ISO14649 toolkit is built at NIST for programming with ISO 14649 Parts 10, 11, and 111. These parts are data models to be used for controlling a machining center. The ISO14649 toolkit has a compiler for 3-axis machining. It reads STEP Part 21 files corresponding to the data model described in Parts 10, 11 and 111 of ISO 14649 and uses a Lex-Yacc parser to generate a parse tree. Later this parse tree is loaded in an internal data structure that represents the contents of the STEP-NC file. It uses the information from the data structure to determine the working steps, geometry of a working step, and operations to be performed for a working step. It uses an internal data structure to monitor the current tool and its dimension, tool position based on the previous commands generated, coolant status, spindle speed, feed-rate, etc.,. Finally, the toolkit generates canonical machining commands using the information from these data structures. The compiler implements only a modest portion of Parts 10, 11, and 111. The compiler includes tool-path generators for rough and finish plane milling, rough and finish rectangular pocket milling, drilling, multistep drilling, reaming, center drilling, countersinking, and counterboring [8].

The object-oriented representation of machining process and the geometry of the working steps in a STEP-NC file enables the ISO14649 toolkit to optimize the tool-path during runtime of the framework. The authors of the ISO14649 toolkit describe it as an interpreter but it actually is a compiler because the toolkit generates an IR, provides the opportunity to optimize the input during runtime, and does not execute the input program instead it generates a different form of representation of the program. The canonical machining commands are atomic commands. Each command produces a single tool motion or a single logical action. There is no standard for canonical machining commands and it might vary based on the implementation. LinuxCNC internally uses canonical machining commands to communicate the operation to be performed in the CNC [9].

3. Design and Implementation

3.1 Just-In-Time Compiler for Intelligent Manufacturing Framework

Today's CNCs have closed source in-house controllers and use G-code part programs as input. These limitations restrict the users of CNCs to use G-code part programs as input. And also the users can not use the feedback data from a vendor specific CNC. Just-In-Time Compiler for Intelligent Manufacturing framework is designed to overcome these drawbacks of CNCs. Figure 2 shows the JIT Compiler for Intelligent Manufacturing framework. The input to the framework is a high-level language program that contains most of the product information from CAD/CAM software, enables optimization on the input, does not require post-processor, and allows users to use the same program across CNCs manufactured by different CNC vendors. Controller is an open-source software that acts as an interface between the user and a CNC. Controller accepts the input, performs optimizations on the input based on the optimization information from either the optimization module or a third-party application, simulates tool-path or controls a CNC, reads the sensor values, stores the feedback machining data and

sensor values in a database, and generates optimized input file that can be used in the subsequent runs of the same product in the same or different CNC. Optimization module reads the machining feedback data and sensor values, determines the optimization to be performed on the input, and sends this information to the controller. Application Programming Interfaces (APIs) enable users to write their applications which use the machining feedback data and sensor data to remotely monitor the machining process, to build an optimization module that sends the optimization information to the controller, and to perform exploration on the historical feedback data set.

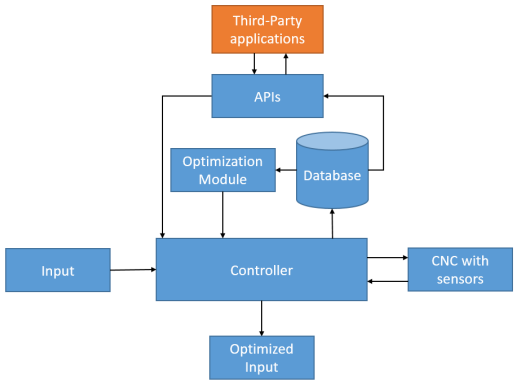


Figure 2: JIT Compiler for Intelligent Manufacturing Framework.

3.2 Prototype Design

This paper presents a prototype of the Just-In-Time Compiler for Intelligent Manufacturing framework. Major parts of the proposed JIT Compiler for Intelligent Manufacturing framework are designed and implemented in the prototype. Figure 3 depicts the system architecture of the prototype. The product to be manufactured is designed in a CAD. The output of the CAD software is passed as input to a CAM software. The output of the CAM software is exported as a STEP-NC file, which is the input to the framework. The input is fed to the ISO14649 toolkit along with a guideline file and a tools file by the user. The guideline file is specific to a product and a machine; each product machined in a CNC will have its guideline file specific for that machine. The guideline file contains optimization information for the product from either the optimization module or a third-party application. ISO14649 toolkit [8] processes the STEP-NC file, guideline file, and tools file to generate appropriate canonical machining commands. The canonical machining commands file is loaded in to LinuxCNC [6] by the user. LinuxCNC reads the canonical machining commands file one line at a time and executes the commands. The output of the execution is displayed in the simulation window of the LinuxCNC software or is used to drive a CNC. The ISO14649 toolkit and LinuxCNC together acts as the controller of the framework.

LinuxCNC generates a lot of real-time machining feedback data that provide real-time information about the machining operation currently being performed. These data are captured by a python script that maps the feedback data and sensor values to a working step and stores the data in real-time in a database. Thus the feedback from the CNC is store in a database in real-time. The prototype exposes application programming interfaces (APIs) that enable users to query the feedback data during machining process and after the product has been manufactured. These APIs provide the opportunity to remotely monitor the machining process in real-time and to perform exploration on the historical feedback data set.

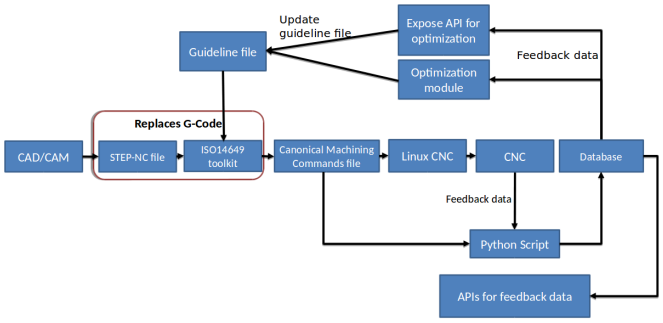


Figure 3: System architecture of JIT Compiler for Intelligent Manufacturing prototype.

And also the prototype has a built-in optimization module that updates the guideline file with optimization information and has APIs to access the guideline file. User can use the built-in optimization module to perform optimizations on the STEP-NC file. This requires the user to input the sensor values and working step information to the built-in optimization module. The optimization module executes an optimization algorithm on the input and updates the guideline file with optimization information for the working step. User can also build a third-party optimization module, which reads the feedback data using the APIs exposed for the feedback data, executes an optimization algorithm, and updates the guideline file using the APIs exposed to access the guideline file. In both the cases, the guideline file is updated with optimization information. Thus optimization information either from the optimization module or from a third-party application is communicated to the controller via the guideline file. This file is used in the subsequent executions of the same product by the ISO14649 toolkit to generate optimized canonical machining commands.

3.3 Prototype Implementation

3.3.1 Controller

ISO14649 toolkit ISO14649 toolkit [8] cannot perform optimization on the input file. So, ISO14649 toolkit is modified to read a guideline file that specifies the optimizations and perform optimization on the input based on this optimization information. The guideline file can be empty or can contain the below structure(s). The structure is defined one per working step and specifies the optimization to be performed for a working step. The structure has two fields, first field is an id field that specifies the working step id and second field is a reduce_by field that represents the value by which the existing depth of cut should be divided. Depth of cut is the depth by which a tool cuts the work piece. Working steps will have different depth of cuts and they are defined in the CAD/CAM phase of the product manufacturing. The actual depth of a working step is a multiple of the depth of cut for the working step. During a pass the tool removes the material in the specified tool path at a given depth of cut. So, the number of passes required to complete the working step is the multiplier by which depth of cut is multiplied to get the actual depth of the working step. For instance, if the reduce_by value is 2 for a working step, the existing depth of cut for the working step will be reduced by half. A working step has one or more operations and an operation has multiple passes. If the depth of cut is reduced by half, the number of passes for the operation increases by 100%, the number of passes for the working step increases by 100% and the number of canonical machining commands for the working step increases to reflect the number of passes. Thus the

amount of material removed in a single pass is reduced by half, the force exerted on the tool is reduced, and the production time is increased. ISO14649 toolkit is modified to change the depth of cut for working steps based on the optimization information from the guideline file. So, the number of canonical machining commands generated for a working step varies based on the reduce_by value for the working step in the guideline file.

```
AXIAL_CUTTER_DEPTH_REDUCTION
{
  id = ""
  reduce_by = 2
}
```

The canonical machining commands file generated by ISO14649 toolkit does not contain working step information. These information are useful to map a line in the canonical machining commands file to a working step. So, ISO14649 toolkit is modified to generate working step information along with the canonical machining commands. These information provide details about where a working step begins. They are generated as comments in the canonical machining commands file because LinuxCNC does not require the working step information and it ignores comments. But these comments are read by a python script to map feedback data to a working step id.

LinuxCNC LinuxCNC [6] accepts only G-code part programs. But internally LinuxCNC converts a G-code into a canonical machining command to simulate tool-path or control a CNC. And also it has a module called canterp that allows the user to load canonical machining command program for a 9-axis CNC. However, this module is not fully developed to simulate tool-path or control a CNC. But the canonical machining command program generated by ISO14649 is for a 2.5-axis CNC. To enable LinuxCNC to accept canonical machining command program for a 2.5-axis CNC and simulate tool-path or control a CNC, the canterp module is modified.

3.3.2 Database

LinuxCNC has python APIs to read the machining feedback data generated in real-time during a machining operation. These data provide information about the machining operation currently being performed. Python script module in the prototype uses these APIs to fetch the machining feedback data at specific time intervals and uses arduino APIs to read the sensor values from the sensors attached to the system. And also it reads the working step information from the canonical machining commands file generated by the ISO14649 toolkit and uses these information to map a line number in the canonical machining commands file to a working step. The machining feedback data from LinuxCNC contains the current line number being executed. Using the working step and line number mapping from the canonical machining commands file and the line number information from the machining feedback data, the python script module maps the machining feedback data and current sensor value to a working step. This mapping is useful for working step specific optimizations. The data collected by the python script module at specific time intervals are stored in a database on a server. Thus the current machining feedback data and sensor values are stored in the database in real-time. At the end of the machining process, the python script module generates a new summary record or updates the existing summary record for the product with the information from the current machining process.

3.3.3 Optimization Module

Just-In-Time Compiler for Intelligent Manufacturing prototype has a built-in optimization module that enables users to perform optimization on the input. User has to input the sensor values and the working step information to the built-in optimization module. For each working step in a product the optimization module is executed once. Optimization module calculates the required change in depth of cut based on the sensor values and the working step information, and updates the guideline file with the reduce_by value for a working step. Optimization module has two optimization models, one for finish milling operation and the other for rough milling operation. Optimization modules reads a configuration file that contains the details about the working steps in a product. The optimization model for the finish milling operation uses a multiple linear regression model [15] to predict the surface roughness of the feature based on average resultant peak force, cutting speed, and feed-rate. The multiple linear regression coefficients are specific to a working step and are specified in the configuration file for a product. If the predicted surface roughness is not the same as the expected surface roughness, the required percentage of change in the average resultant peak force to match the expected surface roughness is calculated and is used to calculate the reduce_by value for the working step. The calculated reduce_by value for the working step is updated in the guideline file. The optimization model for the rough milling operation calculates the change in depth of cut based on the average sensor value of the vibration sensor. The chatter threshold for the working step is specified in the configuration file. If the average vibration sensor value for a working step is greater than the chatter threshold, the reduce_by value for the working step is calculated based on the required percentage of decrease in the average vibration sensor value. The calculated reduce_by value for the working step is updated in the guideline file.

Other Optimizations The prototype supports only depth of cut optimization but the framework is capable of supporting other optimizations. Following are some sample optimizations that can be performed using the framework: coolant usage optimization, change of tool if the current tool is predicted to break or the required tool is not available on the shop-floor, and feed-rate/spindle speed override. Coolant plays a major role in machining because it helps to clear chips, lubricate materials that are sticky, and carry heat away from the cut [2]. Based on the machine condition coolant can be switched on/off by inserting commands at appropriate locations in the generated canonical machining command program. Change of tool optimization can be performed in two scenarios. First scenario is when a tool is not available on the shop-floor, the framework can use a different tool to achieve the same operation by generating a new tool-path using a tool-path generation module similar to the one implemented in the research work SPAIM [10]. This avoids the need to change the tool-path manually outside the framework prior to machining. And also avoids the production delay caused by the unavailability of the tool. The framework should be capable of validating the geometry of the product without human intervention. This can be achieved by developing a module to compare the geometry of the new tool-path and the original tool-path. This validation is required to check for any violation of the product geometry. Next scenario is when an optimization algorithm predicts tool breakage after few runs of the product, the module to change the tool and tool-path can be used to change the current tool with a new tool. The prototype can be extended to perform optimizations on feed-rate and spindle speed in real-time because LinuxCNC is capable of overriding feed-rate and spindle speed in real-time. So, third-party optimization modules can use these variables to control the cutting parameters of the tool.

3.3.4 Application Programming Interface

JIT Compiler for Intelligent Manufacturing prototype exposes APIs that enable users to read feedback data and to access the guideline file. These APIs can be used by third-party applications to re-

motely monitor the machining process, explore historical feedback data set, or perform optimization using the feedback data and update the guideline file. The APIs are written in C++ and compiled as a library. This library can be exported to any remote machine along with the .cshrc file that contains the path to the guideline file and the connection details to the database server. Third-party applications that monitor the feedback data or access the historical feedback data set can use this library from any remote machine. But third-party applications that access the guideline file to perform optimization have to reside in the local machine because the guideline file cannot be accessed over the network. A sample third-party application is implemented to access the exposed APIs and fetch the feedback data. And also the third-party application updates the guideline file with optimized reduce_by value for working steps in a product. The APIs exposed by the prototype are provided in the appendix for reference.

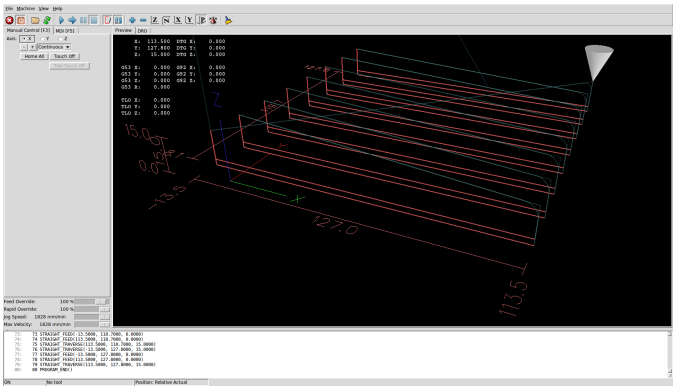


Figure 4: Simulation of face1 product in LinuxCNC.

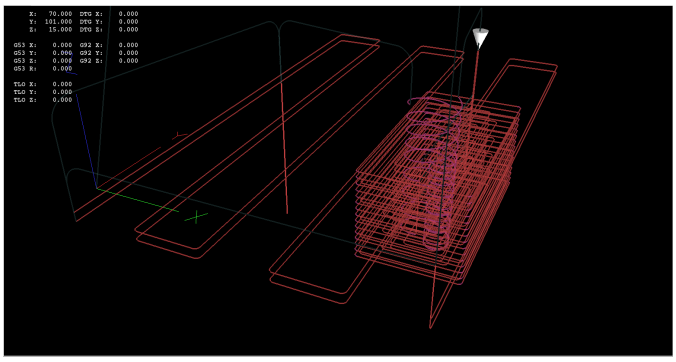


Figure 5: Simulation of ex1_comment product in LinuxCNC.

4. Evaluation

Just-In-Time Compiler for Intelligent Manufacturing prototype was tested in an Intel Core i7-4510U platform. Unoptimized and optimized tool-paths are simulated in LinuxCNC and compared to demonstrate the optimization performed by the prototype using the optimization module and a third-party application. Benefits of the framework are discussed based on the results from the evaluation.

4.1 Simulation

Two sample STEP-NC files are used to simulate the tool-path in LinuxCNC. One of the sample files is used to evaluate the optimization performed using the built-in optimization module and the other is used to evaluate the optimization performed using a third-party application. The sample STEP-NC files used in the evaluation of the prototype are ex1_comment.stp and face1.stp.

face1.stp file is used in the first experimentation. face1.stp file, a tools file and a guideline file are loaded in the ISO14649 toolkit by the user. The tools file is specific to a CNC and is the same for all the products manufactured in a CNC until a tool is removed or added to the CNC. The guideline file for the product face1 and is empty for the first run of the product, and the output file contains the generated canonical machining commands. At the end of the execution of ISO14649 toolkit, the output file contains the generated canonical machining commands for the product face1. This file is loaded in LinuxCNC [6] by the user. Figure 4 shows the user interface of LinuxCNC loaded with the canonical machining commands file generated for the product face1. The preview window on the right in the figure shows the tool-path to be simulated using the canonical machining command program. The manual control window allows the user to change xyz axis settings of the tool before the machining process. Once the user starts the machining process by clicking the "begin executing current file" button, the tool in the simulation window will trace its path along the tool-path specified by the canonical machining commands. Figure 4 shows the final tool-path simulated for the product face1. This is the unoptimized tool-path generated for the product.

The second experimentation is carried out with ex1_comment product. The STEP-NC file and guideline file of ex1_comment product are loaded in ISO14649 toolkit [8] by the user. The ISO14649 toolkit reads the STEP-NC file, guideline file and tools file, and generates canonical machining command program. The canonical machining commands generated for ex1_comment product is loaded in LinuxCNC [6] by the user and the simulation starts when the user clicks the "begin executing current file" button. LinuxCNC simulation output for ex1_comment product is shown in the figure 5.

4.2 Optimization module

Just-In-Time Compiler for Intelligent Manufacturing framework supports a built-in optimization module that accepts the sensor values and updates the guideline file with the optimization information by executing an optimization algorithm on the sensor values. The built-in optimization module has two optimization models; one optimization model is for finish milling [15] and the other optimization model is for rough milling. These two optimization models are experimented using the values from a random number generation function. They are used to demonstrate the functionality of the optimization module.

The product ex1_comment has three working steps those have depth of cut parameter in their machining operation. These working steps are FINISH PLANAR FACE1, FINISH POCKET1, and ROUGH POCKET1. FINISH PLANAR FACE1 and FINISH POCKET1 working steps are finish milling working steps. They are optimized using the finish milling optimization model. This model uses the cutting speed, feed rate, and average resultant peak force to calculate the change in depth of cut. User inputs these values along with the working step id and the path to the configuration file. The optimization module calculates the change in depth of cut for the working step and updates the guideline file with the optimization information for the working step. ROUGH POCKET1 working step is a rough milling working step and is optimized using the rough milling optimization model. This model uses the average vibration sensor value for the working step to calculate the required change in depth of cut for the working step. The updated guideline file is shown in the figure 6. The value of the reduce_by field for the working step FINISH PLANAR FACE1 is 2.078747. This reduces the depth of cut for the working step by half. So, the number of passes

Canonical machining command program	Number of lines
Unoptimized	542
Optimized	504

Table 1: Number of canonical machining commands in the unoptimized and optimized canonical machining command program of the product ex1_comment.

for the working step increases by 100%. The number of passes for the working step ROUGH POCKET1 increases by 80% because the value of the reduce_by field is 1.875001. But for the working step FINISH POCKET1, the number of passes is decreased by 53.33% because the reduce_by field has a value of 0.418156, which increases the depth of cut by 114.29%.

```
AXIAL_CUTTER_DEPTH_REDUCTION
{
        id = "ROUGH POCKET1"
        reduce_by = 1.875001
}
AXIAL_CUTTER_DEPTH_REDUCTION
{
        id = "FINISH POCKET1"
        reduce_by = 0.418156
}
AXIAL_CUTTER_DEPTH_REDUCTION
{
        id = "FINISH PLANAR FACE1"
        reduce_by = 2.078747
}
```

Figure 6: Updated guideline file with the optimization details for ex1_comment product.

The updated guideline file and the STEP-NC file for the product ex1_comment is used to generate the canonical machining commands for the next run of the product. The ISO14649 toolkit [8] generates the optimized canonical machining commands. The number of canonical machining commands in the unoptimized canonical machining command program and optimized canonical machining command program is shown in the table 1. The decrease in the number of passes in the working step FINISH POCKET1 has significant impact on the decrease in the number of canonical machining commands than the increase in the number of passes in the working steps FINISH PLANAR FACE1 and ROUGH POCKET1. This is because the number of commands required to change the tool direction in the working step FINISH POCKET1 is higher than that of other two working steps. The generated optimized canonical machining commands file is loaded and simulated in LinuxCNC [6] by the user. Figure 7 shows the simulated optimized tool-path for the product ex1_comment.

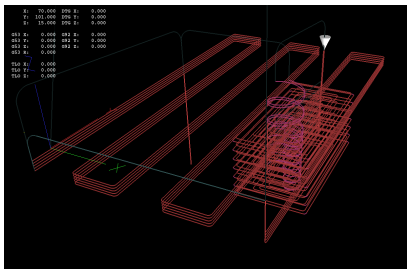
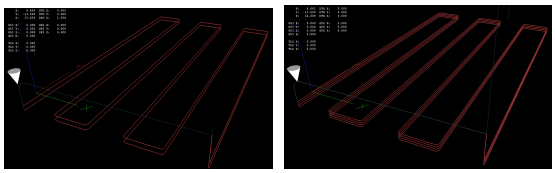


Figure 7: Simulation of optimized tool-path for ex1_comment product.

Figure 8, 9, and 10 shows the tool-paths generated for the unoptimized and optimized canonical machining command programs for the working steps FINISH PLANAR FACE1, FINISH POCKET1, and ROUGH POCKET1. It is clear from these figures that the number of passes for the working steps FINISH PLANAR FACE1 and ROUGH POCKET1 have increased, and the number of passes for the working step FINISH POCKET1 has decreased.

The percentage of increase in the number of passes for the working steps FINISH PLANAR FACE1 and ROUGH POCKET1 are 100% and 80% respectively and the percentage of decrease in the number of passes for working step FINISH POCKET1 is 53.33%. The percentage change in the number of passes for these working steps have significant impact on the machining time of these working steps. The time taken for machining the working steps FINISH PLANAR FACE1 and ROUGH POCKET1 have increased by 126.24% and 67.433% respectively and the time taken for machining the working step FINISH POCKET1 has decreased by 49.333%. These details are shown in the tables 4 and 6.

Table 2 shows the time taken for generation of canonical machining commands and total machining time for unoptimized and optimized versions of the product ex1_comment. The time taken for generation of canonical machining commands and total machining time has increased by 4.894% and 16.316% respectively. The time taken for generation of canonical machining commands is in the magnitude of microseconds and the overhead in the generation time is 4.894%. This does not have significant impact on the production time. The increase in the total machining time is attributed to the increase in the number of passes for the working steps FINISH PLANAR FACE1 and ROUGH POCKET1. The increase in the time taken for machining these working steps has less impact on the increase in the total machining time because these working steps contribute only 20.34% and 24.06% respectively towards the total unoptimized machining time of the product ex1_comment. But the working step FINISH POCKET1 has a contribution of 52.35%. So, the decrease in the machining time caused by this working step has compensated certain percentage of increase in the production time by the other two working steps. Thus the overhead caused by the framework is very less when compared to the time taken to machine the working steps. And also the increase in the machining time of the working steps is proportional to the increase in the number of canonical machining commands for the working steps. The machining time specified in the evaluation are not actual time taken to machine the part. But they are the time taken to simulate the part in LinuxCNC.

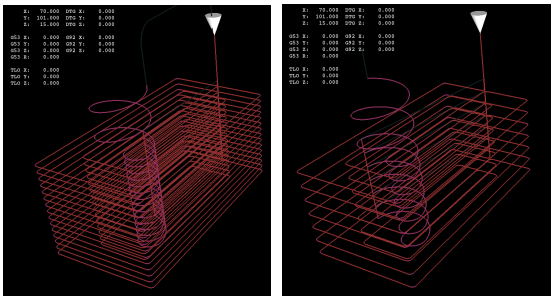


(a) (b)

Figure 8: Unoptimized (a) and Optimized (b) tool-path for the working step FINISH PLANAR FACE1

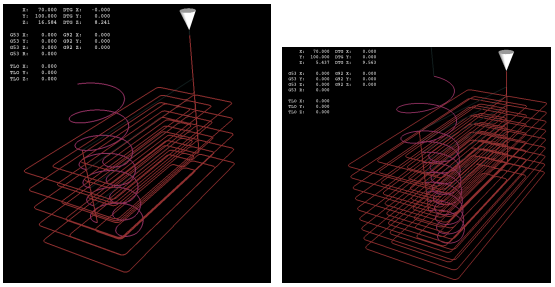
4.3 Third-Party application

A sample third-party application is implemented to experiment the functionalities of the APIs. This application uses the APIs to fetch all the products that has been simulated by the LinuxCNC, fetch all the working steps of the product ex1_comment, fetch all the sensors used in manufacturing the product ex1_comment, get the manufacturing summary of the product ex1_comment, fetch the



(a) (b)

Figure 9: Unoptimized (a) and Optimized (b) tool-path for the working step FINISH POCKET1



(a) (b)

Figure 10: Unoptimized (a) and Optimized (b) tool-path for the working step ROUGH POCKET1

Version	CMC generation time	Machining Time
Unoptimized	2350µs	403.015s
Optimized	2465µs	468.771s

Table 2: Time taken for generating canonical machining commands (CMC) and machining the product ex1_comment.

Version	FINISH PLANAR FACE1	ROUGH POCKET1	FINISH POCKET1
Unoptimized	82.483s	97.84s	211.928s
Optimized	186.126s	163.817s	107.378s

Table 3: Time taken for each working step during unoptimized and optimized machining of the product ex1_comment.

	FINISH PLANAR FACE1	ROUGH POCKET1	FINISH POCKET1
Percentage Change	100% Increase	80% Increase	53.33% Decrease

Table 4: Percentage change in the number of passes for each working step in the product ex1_comment.

	CMC generation time	Machining Time
Percentage Increase	4.894%	16.316%

Table 5: Percentage increase in time taken for optimized Canonical Machining Commands generation, optimized Machining Time of the product ex1_comment.

	FINISH PLANAR FACE1	ROUGH POCKET1	FINISH POCKET1
Percentage Change	126.24% Increase	67.433% Increase	49.333% Decrease

Table 6: Percentage change in time taken for each working step in the product ex1_comment during optimized machining.

sensor 1 values for the working step FINISH PLANAR FACE1 during the first run of the product ex1_comment, fetch the sensor 2 values during the first run of the product ex1_comment, and access the guideline file to update the value of the reduce_by field for the working step FINISH PLANAR FACE1 of the product face1. Figure 11 shows the output of the third party application and the figure 12 shows the updated guideline file. The value of the reduce_by field for the working step FINISH PLANAR FACE1 is 2.000000. So, the depth of cut for the working step is reduced by half and the number of passes for the working step is doubled.

```
Successfully fetched all products
"ex1_comment"
"face1"

Successfully fetched all working steps for the product : ex1_comment
"FINISH PLANAR FACE1"
"ROUGH POCKET1"
"HOLE1 D=22MM"
"FINISH POCKET1"

Successfully fetched all sensors for the product : ex1_comment
"sensor_1"
"sensor_2"

Successfully fetched summary for the product : ex1_comment
Current machine status : "Finished"
Number of failed jobs : 0
Number of successful jobs : 2
Run Count : 2
Total Run Time : 0 days, 869 seconds, and 0 microseconds

Successfully fetched sensor_1 values for the working step FINISH PLANAR FACE1 in the product ex1_comment for run 1
[30, 22, 29, 21, 66, 65, 24, 76, 30, 53]

Successfully fetched sensor_2 values for the product ex1_comment for run 1
[100 ,123 ,115 ,102 ,100 ,121 ,134 ,106 ,129 ,150]

Successfully updated the Depth Of Cut change
```

Figure 11: Output of the third party application.

```
AXIAL_CUTTER_DEPTH_REDUCTION
{
        id = "FINISH PLANAR FACE1"
        reduce_by = 2.000000
}
```

Figure 12: Updated guideline file of the product face1.

The updated guideline file is used by the ISO14649 toolkit [8] in the next run of the product face1. Table 7 shows the number of canonical machining commands in the unoptimized and optimized canonical machining command programs. The number of canonical machining commands in the optimized canonical machining command program is greater than that of unoptimized canonical machining command program because the number of passes in the product face1 has doubled. The updated guideline file is loaded in LinuxCNC to simulate the optimized tool-path. Figure 13 shows the optimized tool-path simulated in LinuxCNC [6]. Figure 4 and 13 show that the number of passes in the optimized tool-path has increased because of the optimization performed by the ISO14649

Canonical machining commands file	Number of lines
Unoptimized file	80
Optimized file	143

Table 7: Number of canonical machining commands in the unoptimized and optimized canonical machining commands file of the product face1.

Version	CMC generation time	Total Machining Time	FINISH PLANAR FACE1 Machining Time
Unoptimized	1419µs	156.280s	156.277s
Optimized	1497µs	307.0824s	306.877s

Table 8: Time taken for generating canonical machining commands, total machining time, and FINISH PLANAR FACE1 machining time of the product face1.

toolkit using the guideline file. The percentage of increase in the number of passes for the working step FINISH PLANAR FACE1 in the optimized tool-path is 100% and it has a significant impact on the machining time of the working step. The machining time of the working step has increased by 96.36%. The increase in the machining time of the working step has increased the total machining time of the product by 96.49%. Thus the increase in the number of passes for the working step FINISH PLANAR FACE1 has significant impact on the total machining time of the product face1. The overhead caused by the generation of optimized canonical machining commands for the product face1 is 5.49%. The canonical machining commands generation time is in the magnitude of microseconds and the percentage change in the canonical machining commands generation time does not have significant impact on the production time of the product face1. Thus the overhead caused by the framework is very less when compared to the time taken to machine the working step. And also the increase in the machining time of the working step is proportional to the increase in the number of canonical machining commands for the working step. The machining time specified in the evaluation are not actual time taken to machine the part. But they are the time taken to simulate the part in LinuxCNC.

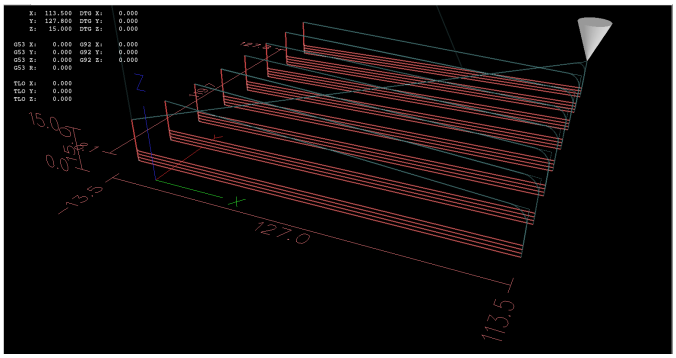


Figure 13: Simulation of optimized tool-path for face1 product.

	CMC generation time	Total Machining Time	FINISH PLANAR FACE1 Machining Time
Percentage Increase	5.49%	96.49%	96.36%

Table 9: Percentage increase in Canonical Machining Commands generation time, Total Machining Time, and FINISH PLANAR FACE1 machining time of the product face1.

4.4 Benefits of the framework

JIT Compiler for Intelligent Manufacturing framework can be used on the shop floor to monitor the machining process remotely, explore historical feedback data set and build customized optimization algorithms to optimize the machining process on the fly. This section presents three scenarios in which the JIT Compiler for Intelligent Manufacturing framework can be used on the shop-floor.

4.4.1 Scenario A

If the CNCs or the cutting tools used in the CNCs on the shop-floor are newer than expected and the depth of cut is optimized to an old tool, the framework can detect this using the surface roughness based optimization model. The surface roughness of the product is predicted to be very less than the desired surface roughness and the model suggests an increase in the depth of cut such that the surface roughness is slightly less than the desired surface roughness. The increase in depth of cut decreases the number of passes required to complete a working step. As shown in the evaluation of the optimization module, the decrease in the number of passes has significant impact on the production time of the product. If there are several working steps in a product and all of them use the same tool, the optimization model decreases the number of passes for all these working steps.

If a product has m working steps and the time taken to machine each working step is y minutes, the total time taken to machine the product is represented as

$$x = y \times m \tag{1}$$

minutes. If n out of m working steps use a relatively new tool and the framework reduces the number of passes for the n working steps by z%, the total time taken to machine the product using optimized depth of cut is

$$x' = y \times n \times (1 - \frac{z}{100}) + y \times (m - n) \tag{2}$$

minutes and the percentage of decrease in the machining time is

$$\%ofdecrease = \frac{(y \times m) - (y \times n \times (1 - \frac{z}{100}) + y \times (m - n))}{y \times m} \times 100 \tag{3}$$

If there are 10 working steps in a product and each working step takes 15 minutes to complete, the total time taken to machine the product is 2 hours and 30 minutes. If 7 out of 10 working steps use a relatively new tool and the percentage of decrease in the number of passes for these working step is 60%, the time taken to machine the optimized tool-path is 1 hour and 27 minutes. The production time is decreased by 42%. If there are 100 machines with a relatively new tool and all these machines manufacture the same product, the production time saved is 105 hours, which allows the shop floor to manufacture 72 more pieces of the same product. Users of the framework can build optimization algorithms customized to the tool condition, tool-path and workpiece material to efficiently

detect working steps whose depth of cut can be increased. This will lead to a much more efficient production of the product.

4.4.2 Scenario B

The optimization module in the framework detects tool wear by monitoring the chatter level and surface roughness. If the tool used in a CNC is old and is wearing off, the finish of the surface is defective. The optimization module detects this and suggests a decrease in the depth of cut for the working steps those use the old tool. The decrease in depth of cut decreases the force exerted on the tool. So, the number of defective pieces manufactured is decreased. But this increases the number of passes for the working steps.

If the defective pieces are manually identified on the shop floor after n cycles of manufacturing a product and there are m machines with an old tool manufacturing the same product,

$$n \times m \tag{4}$$

defective pieces of the product are manufactured. If each machine takes x minutes to manufacture the product and requires y minutes to change the cutting parameters manually, there is a production delay of

$$(x \times n \times m) + (m \times y) \tag{5}$$

minutes. But if the framework is used to detect the tool wear by monitoring the surface roughness and chatter levels, the depth of cut is decreased in the next cycle. So, the force exerted on the material and tool is reduced and there is no damage to the finish of the part. The number of defective pieces manufactured, if the framework is used is

$$1 \times m \tag{6}$$

pieces. The production delay caused is

$$(x \times m) \tag{7}$$

minutes. The percentage of decrease in the number of defective parts manufactured is

$$\%ofdecrease = \frac{(n \times m) - (1 \times m)}{n \times m} \times 100 = \frac{n-1}{n} \times 100 \tag{8}$$

and the percentage of decrease in the production delay is

$$\%ofdecrease = \frac{(x \times n \times m) + (m \times y) - (x \times m)}{(x \times n \times m) + (m \times y)} \times 100 \tag{9}$$

If 10 machines manufacture the same product and the defective pieces are detected after 3 cycles of manufacturing the product, 30 defective pieces are manufactured. If the time taken to manufacture a piece of the product is 1 hour and the average time taken to modify the cutting parameters in a machine is 5 minutes, there is a production delay of 30 hours and 50 minutes. If the framework is used to detect the tool condition and modify the depth of cut, the production delay is reduced to 10 hours and the number of defective pieces manufactured is reduced to 10. The percentage of decrease in the production delay and number of defective pieces manufactured are 67.5% and 66.7% respectively. Users of the framework can build an optimization algorithm that predicts the tool wear and changes the cutting parameters even before the production of first cycle of defective products. This would avoid production of defective products and the production delay caused due to manufacturing defective products.

4.4.3 Scenario C

The material of the workpiece plays a major role in deciding the correct cutting parameters. The cutting parameters are feed-rate, spindle speed, and depth of cut. If the material of the workpiece were to change during a manufacturing process, the optimized cutting parameters have to be fed in to the machine by the operator. Operator can control only the feed-rate and spindle speed because a change in depth of cut causes the tool-path to change and it is not possible to edit the tool-path during machining process. And also there is a delay in the production process caused due to human intervention because the operator has to identify the optimal cutting parameters either by trial and error mechanism or using prior knowledge about the workpiece material. If the hardness of the workpiece material is stored in the database of the framework and historical data sets for the cutting parameters and hardness combinations are available, an algorithm can be devised to identify the optimal cutting parameters based on the hardness of the workpiece material. These parameters can be used to generate canonical machining commands for optimized machining of the product. This does not require the operator to have prior knowledge about the workpiece material and does not require human intervention during manufacturing the product. Thus the production delay caused due to human intervention is avoided.

5. Relevant Work

5.1 Traditional Just-In-Time Compiler in JVM

The Just-In-Time Compiler for Java Virtual Machine aims to reduce the execution time by performing feedback-directed optimization at runtime and limiting the overhead of runtime optimization. There are two types of feedback-directed optimizations: online and offline feedback-directed optimization. Offline feedback-directed optimizations are optimizations in which the application behavior is captured from a prior execution of the program. This approach fails in scenarios where 1) it is impractical to collect a profile prior to execution, or 2) the application's behavior differs from its behavior during the profiling run[1]. Online feedback-directed optimization avoids the drawbacks of offline profiling by performing optimizations based on the profiles collected during runtime. JIT Compiler for JVM achieves this by compiling a java source code into byte-codes using a java compiler and then interprets the bytecode to machine instructions at runtime. The bytecode provides the opportunity to perform optimizations during runtime because bytecode contains high-level information that can be used to perform optimizations.

Optimizations performed on a java program are much more complex than the optimizations performed on a STEP-NC file because java programs have branches, loops, different control flows, and data flows. The control path taken by a java program varies based on the input to the program so the offline profile collected for a java program might not provide the tailored optimization for the subsequent execution environment. In case of STEP-NC programs, the execution environment varies as the machining conditions varies. So the offline profile collected from an execution is valid for the subsequent run in the same machine until the tool wears off or the workpiece material changes. The online feedback-directed optimization of a STEP-NC program is the real-time optimization of the machining operation and it has to be done quick enough to propagate the changes before the next set of machining instructions are executed. So, it has a hard deadline on the optimization time. The type of optimizations performed on a java program varies from the optimizations performed on a STEP-NC program. The optimizations performed on a java program aims to reduce the execution time and space. But the optimizations performed on a STEP-NC program focuses to minimize machining time (Time-Critical) or optimize surface quality (Quality-Critical). Time-Critical machining operations are often for roughing purposes whereby increasing the material removal rate is one of the main goals and Quality-Critical machining operations are often used for finishing purposes where surface quality is of the main concern[11].

5.2 STEP-NC enabled controller for CNC

Over the past years several research works have been carried out to use STEP-NC to drive CNCs, perform online inspection, and optimize the machining parameters. The authors of the research work [11] propose a STEP-NC enabled Machine Condition Monitoring (MCM) system that accepts a STEP-NC ISO 14649 data model file as input, performs initial feed-rate optimization, converts the STEP-NC program in to a Canonical Machining Commands (CMC) part program using an interpreter that is capable of handling optimization data, drives a CNC using a customized machine tool controller that accepts the CMC part program, monitors and records the machining parameters over the network using MTConnect, visualizes and evaluates the machining parameters to obtain another set of optimum parameters, calculates appropriate feed-rates using the optimum parameters for subsequent machining operations, and updates the STEP-NC file with the calculated feed-rate value. This research work proposes a system that is capable of optimizing only the feed-rate based on the evaluation proposed by the work. This system requires human intervention during the machining process and does not allow users to remotely monitor the machining process, perform profiling, or build third-party optimization algorithms customized to the manufactured product.

The work [10] developed and implemented a NC controller that estimates online process data and optimizes machining parameters and tool-paths to eliminate or compensate the results of tool deflection, machine tool and spindle vibrations, geometric errors, thermal effects, chatters, control errors, etc., by enabling users to change the tool if the corresponding tool is not available on the shop floor, generate the modified tool-path automatically on the shop floor, and validate it before machining. This work focuses on optimization of the tool-path based on the tools available in a machine before machining to avoid tracking errors or tool deflection. This approach requires human intervention to validate the generated tool-path. Thus the process is not entirely automated and has production delay whenever the tool has to be changed.

6. Conclusion and Future Work

Evaluation of the JIT Compiler for Intelligent Manufacturing prototype shows that the overhead of the framework is less and has opened the possibility of using the framework on the shop-floor for efficient production. The hypothesis stands valid as the prototype accepts STEP-NC ISO 14649 data model as input, uses an open-source controller as the CNC controller, stores the feedback data in real-time in a database, performs optimizations on the input, exposes APIs to read feedback data and to suggest optimization information to the controller, and enables users to remotely monitor the machining process, build third-party optimization algorithms and explore historical dataset. The STEP-NC ISO 14649 data model contains high-level machining information that allows optimization of the machining process and can be used across CNCs manufactured by different CNC vendors. The framework is flexible enough to replace the input with the latest STEP-NC ISO data model and to control CNCs that are compatible with LinuxCNC. And also the framework allows users to efficiently machine a product using customized third-party optimization algorithms customized for the product.

JIT Compiler for Intelligent Manufacturing prototype performs optimization on the input between runs; optimization is performed based on the feedback from the previous runs. The next step is to modify the prototype to enable real-time optimization of the machining process. The feedback data from the machining process can be used to build a real-time visualization tool that displays the real-time tool-path, tool position, and operation performed in a CNC. The prototype supports only optimization based on depth of cut. It can be extended to support different optimizations such as coolant usage based on the machining condition, change of tool based on the tool condition and tool availability, change in feed-rate, and change in spindle speed. Product geometry validation module can be build to check for any violation of the product geometry after optimization of the input to the framework. The ISO14649 toolkit in the prototype should be replaced with a compiler that accepts the latest STEP-NC data model or AP238 data model and converts it into an intermediate representation that can be used to drive LinuxCNC. The new compiler can be easily modified to perform optimizations on a STEP-NC file because STEP-NC data models have an object oriented representation of the machining process and the geometry of the product. These modifications will enable the framework to accept STEP-NC data model generated by any CAD/CAM software. And also the framework can be used on the shop-floor and the users can utilize the full potential of the framework.

References

[1] M. Arnold, M. Hind, and B. G. Ryder. Online Feedback-Directed Optimization of Java. *OOPSLA '02 Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 37(11):111–129, 2002.

[2] I. CNCCookbook. Coolant and Chip Clearing, 2016. URL `http://www.cnccookbook.com/CCCNCMillFeedsSpeedsCoolant.htm`.

[3] Inc.com. Computer-Aided Design (CAD) and Computer-Aided Manufacturing (CAM), 2013. URL `http://www.inc.com/encyclopedia/computer-aided-design-cad-and-computer-aided-cam.html`.

[4] A. Kržič, and. P Stoic and J. Kopač. STEP-NC: A New Programming Code for the CNC Machines. *Strojniški vestnik - Journal of Mechanical Engineering*, 55(6):406–417, 2009.

[5] linuxcnc.org. LinuxCNC Developer Documentation, 2016. URL `http://www.linuxcnc.org/docs/2.5/html/code/Code_Notes.html`.

[6] linuxcnc.org. Linuxcnc sourcecode. `git://git.linuxcnc.org/git/linuxcnc.git`, 2016.

[7] S. Mohsen. *Reading Materials for IC Training Modules, Computer Numerical Control*. INDUSTRIAL CENTRE, THE HONG KONG POLYTECHNIC UNIVERSITY, 2009.

[8] NIST. A toolkit for iso 14649 data model. `https://github.com/ArcEye/iso-14649-toolkit`, 2009.

[9] F. Proctor, T. Karmer, and J. michaloski. Canonical Machining Commands, 1997. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.9814&rep=rep1&type=pdf`.

[10] M. Rauch, R. Laguionie, J. Hascoet, and S. Suh. An advanced STEP-NC controller for intelligent machining processes. *Robotics and Computer-Integrated Manufacturing*, 28:375–384, 2012.

[11] F. Ridwan and X. Xu. Advanced CNC system with in-process feed-rate optimisation. *Robotics and Computer-Integrated Manufacturing*, 29:12–20, 2013.

[12] wikipedia.org. Computer-aided manufacturing, 2016. URL `https://en.wikipedia.org/wiki/Computer-aided_manufacturing`.

[13] wikipedia.org. Numerical control, 2016. URL `https://en.wikipedia.org/wiki/Numerical_control`.

[14] wikipedia.org. LinuxCNC, 2016. URL `https://en.wikipedia.org/wiki/LinuxCNC`.

[15] J. Z. Zhang and J. C. Chen. The development of an in-process surface roughness adaptive control system in end milling operations. *The International Journal of Advanced Manufacturing Technology*, 31: 877–887, 2007.