ระบบการคำนวณแบบกระจายสำหรับหอกลั่น

นาย สัมฤทธิ์ ลิ่มวงศ์สุวรรณ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคณนา ภาควิชาคณิตศาสตร์

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2544

ISBN 974-03-1643-3

DISTRIBUTED COMPUTING SYSTEM FOR A DISTILLATION COLUMN

Mr. Sumrid Limvongsuwan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computational Science
Department of Mathematics
Faculty of Science
Chulalongkorn University
Academic Year 2544
ISBN 974-03-1643-3

Thesis Title	Distributed Computing System for a Distillation Column
By	Mr. Sumrid Limvongsuwan
Field of study	Computational Science
Thesis Advisor	Pornpote Piumsomboon, Assoc. Prof. Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

……………………………………… Deputy Dean for Administrative Affairs,
Acting Dean, Faculty of Science

(Associate Professor Pipat KranTiang, Ph.D.)

THESIS COMMITTEE

……………………………………………….. Chairman

(Associate Professor David Ruffolo, Ph.D.)

……………………………………………….. Thesis Advisor

(Associate Professor Pornpote Piumsomboon, Ph.D.)

……………………………………………….. Member

(Professor Chidchanok Lursinsap, Ph.D.)

สัมฤทธิ์ ลิ่มวงศ์สุวรรณ : ระบบการคำนวณแบบกระจายสำหรับหอกลั่น.(DISTRIBUTED COMPUTING SYSTEM FOR A DISTILLATION COLUMN) อ. ที่ปรึกษา : รศ. ดร. พรพจน์ เปี่ยมสมบูรณ์ จำนวนหน้า 92 หน้า. ISBN 974-03-1643-3.

งานวิจัยนี้เป็นการพัฒนาระบบการคำนวณแบบกระจาย เพื่อจำลองพลศาสตร์สำหรับหอกลั่น ประสิทธิภาพของการคำนวณแบบขนานขึ้นกับขนาดของหอกลั่นและการสื่อสารบนเครือข่าย แบบจำลองหอกลั่นใช้ระบบสมการพื้นฐาน MESH และประยุกต์การส่งผ่านข้อมูลด้วย MPI ในระบบเครือข่ายคอมพิวเตอร์ด้วยเครื่องคอมพิวเตอร์ 3 เครื่องชนิด AMD Duron 600 MHz.

งานวิจัยนี้เป็นการศึกษาพฤติกรรมเชิงพลวัตของหอกลั่นชนิดสององค์ประกอบและหลายองค์ประกอบโดยใช้การเทคนิคการคำนวณจุดเดือดในการคำนวณหาคำตอบ แต่ละหน่วยประมวลผลจะคำนวณหอกลั่นที่แบ่งย่อยในขนาดเท่ากัน ในแต่ละขั้นเวลาของการอินทิเกรตส่วนที่ทำงานแบบลำดับจะไม่รองานจากหน่วยประมวลผลอื่น จากการศึกษาเปรียบเทียบผลลัพธ์จากการทำงานแบบอนุกรมและแบบขนานพบว่าการเลือกขนาดของหอกลั่นและปริมาณการถ่ายโอนข้อมูลที่เหมาะสมส่งผลให้ การทำงานแบบขนานเป็นไปได้ด้วยดี

ภาควิชา.......คณิตศาสตร์................. ลายมือชื่อนิสิต...
สาขาวิชา......วิทยาการคณนา........... ลายมือชื่ออาจารย์ที่ปรึกษา.......................................
ปีการศึกษา...2544

4172482923 : MAJOR COMPUTATIONAL SCIENCE

KEY WORD: DYNAMIC PROCESS SIMULATION / DISTRIBUTED MEMORY / PARALLE COMPUTING / MESSAGE PASSING / MESH EQUATION

SUMRID LIMVONGSUWAN: DISTRIBUTED COMPUTING SYSTEM FOR A DISTILLATION COLUMN. THESIS ADVISOR: PORNPOTE PIUMSOMBOON, Assoc. Prof. Ph.D., 92 pp. ISBN 974-03-1643-3.

The use of a distributed memory message-passing system for dynamic chemical process simulation was developed. The computational performance of parallel system depends on the size of distillation column and the data communication. The distillation model was based on MESH equations. The case study example is performed using MPI message-passing system on three AMD DURON 600 MHz. Multicomputers.

In this study, the dynamic behaviors of the binary and multicomponent distillation column were investigated. The multicomponent distillation was solved by the Bubble point method. The distillation column was partitioned into subsystems for each processor. For each subsystem, the serial computing portions were carried out without waiting computation the values from other processors for each time horizon integration. The comparative results between serial and parallel computations suggest that there is an optimum size of the distillation column and that of the transferred data to obtain the better result with parallel processing.

Department Mathematics .... Student's signature.....................................

Field of study Computational Science.. Advisor's signature......................................

Academic year 2001

ACKNOWLEDGEMENT

The author would like to express his sincere thanks to Assoc. Prof. Dr. Pornpote Piumsomboon, thesis advisor for his excellent guidance and extreme assistance toward the completion of the thesis. To the thesis committee, Assoc. Prof. Dr. David Ruffolo and Prof. Dr. Chidchanok Lursinsap.

Most of all, the author would like to express the highest gratitude to his parents, teachers, sister, and friends for their inspiration and encouragement.

CONTENTS

Page

CONTENTS (Continued)

CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

Page

LIST OF FIGURES (Continued)

NOMENCLATURE

LATIN CAPITAL AND LOWER CASE LETTERS

A, B, C	Antoine's constant
B	Bottom flow rate
C	Number of components in a mixture; molal specific heat;
	Constant for unit conversion
D	Distillate flow rate
E	The error or deviation from the set point; Murphree tray efficiency
F	Feed flow rate $(\mathrm{lb_{mol}/hr})$
h	Liquid enthalpy per mole; liquid height
H	Vapor enthalpy per mole
K	Gain; equilibrium ratio
l	Length
L	Liquid flow rate $(\mathrm{lb_{mol}/hr})$
M	Liquid mass holdup
N	Number of stages; Number of processors
NE	The number of equations
NT	The number of trays
P	Pressure

R	Reflux rate
S	Side stream flow rate
t	Time
T	Temperature
V	Vapor flow rate $(\mathrm{lb_{mol}/hr})$
x	Mole fraction in liquid flow rate
y	Mole fraction in vapor flow rate

GREEK LETTERS

α	Relative volatility; serial fraction
β	Tray hydraulics constant
λ	Latent heat
ρ	Liquid density
τ	Feedback-reset time for integral action

SUBSCRIPTS

avg	Average
B	Bottom

C	Controller
D	Distillate
j	Particular component
L	Liquid
max	Maximum
n	Stage
nj	Particular component j in a stream leaving stage n
NT	nth tray
P	Pressure constant
V	Vapor
w	Weir

SUPERSCRIPTS

L	Liquid
NC	Number of components in a mixture
V	Vapor
set	Set point
$-$	Initial
*	In equilibrium

CHAPTER I

INTRODUCTION

1.1 Background

The technological driving force behind parallel computing is VLSI, or very large scale integration, the same technology that created the personal computer and workstation market over the last decade. In 1980, the Intel 8086 used 50,000 transistors; in 2002, the latest AMD Athlon chip contains 37.5 million transistors a factor of 750 increase. The dramatic improvement in chip density comes together with an increase in clock speed and improved design so that the Athlon performs better by a factor of over one thousand on scientific problems than the 8086-8087 chip pair of the early 1980s.

By the year 2000, parallelism is thus inevitable to all computers, from your children's video game to personal computers, workstations, and supercomputers. At present, we see parallelism in the larger machines as we replicate many chips and printed circuit boards to build systems as arrays of nodes. Each unit of which is some variant of the microprocessor. Parallelism allows one to build the world's fastest and most cost-effective supercomputers. Distributed computing systems have become more practical to implement due to advances in computer network technology and the drastic reduction in cost of processors.

Beowulf is a kind of high-performance massively parallel computer built primarily out of commodity hardware components, running a free-software operating system like Linux or FreeBSD, interconnected by a private high-speed network. It consists of a cluster of PCs or workstations dedicated to running high-performance computing tasks. The first Beowulf was built with DX4 processors with Slackware Linux and 10Mbit/s Ethernet by Thomas Sterling and Don Becker (1994) [1]. Beowulfs make a computer with supercomputer performance for a third to tenth the price of a traditional supercomputer. The key component to compatibility is the system software used on

Beowulf. PVM and MPI are system software that write message-passing parallel programs that run on a cluster, in Fortran and C.

The application of advanced computer architectures such as distributed computing systems, with their inherent advantages over conventional unique processor computers, could significantly aid the dynamic simulation of large-scale industrial processes. Distributed computing systems had become more practical to implement due to advances in computer network technology and the drastic reduction in the cost of processors. Dynamic process simulation was a problem of considerable importance to chemical engineers. The predicted transient behavior of processes under different conditions could be used for developing and testing alternative control schemes, for training plant personnel, and for optimizing plant operations, without the expense and possible hazard of plant experimentation.

The success of the chemical process industries depends on the ability to design and operate complex highly interconnected plants that were profitable and that met quality, safety, environmental, and other standards. Towards this goal, process simulation and optimization tools were increasingly being used industrially in every step of the design process and in subsequent plant operations. However, the solution of realistic, industrial-scale process modeling problems for dynamic simulation and optimization is computationally intense, and may require the use of high performance computing (HPC) technology to be done in a timely manner, especially for real-time performance.

In this thesis, we consider dynamics of distillation columns working on the binary and multicomponent systems. The simulation of a distillation column is composed of a large number of ODEs and algebraic equations. The binary column has twenty trays with product controller. There are two ODEs per tray and two algebraic equations per tray. In non-ideal multicomponent column, we study the performance of algorithm by fifteen, sixty, one hundred and twenty, two hundred and forty, and four hundred and eighty trays. Each tray, the following material-balance, phase-equilibrium, mole-fraction-summation, and energy-balance (MESH) was applied. In principle, multicomponent equations are very similar to those for binary systems, but the solution involves substantially more computational effort.

Our goal is to design an algorithm for parallel computation for simulating dynamic processes in chemical industries. Each processor (node) shares information and synchronizes the program executions. The nodes send and receive message over the network. Beowulf machines used MPI-Fortran run up to three AMD Duron 600 MHz on Linux Slackware.

1.2 Objectives

1. Design distributed computing system on network computers.
2. Apply the system to simulate dynamic behavior multicomponent distillation.

1.3 Scope of Work

1. Study network computing theory and network programming.
2. Study steady state and dynamic behavior of binary and multicomponent distillation columns.
3. Design the serial and parallel computer algorithms to describe the dynamics of distillation column.
4. Develop the serial and parallel computer programs.
5. Compare the results obtained from serial and parallel programs.
6. Study the computing condition to obtain the best speed up and efficiency.

1.4 Benefits Expected

1. Demonstrate how to apply parallel computing technique to solve a dynamic distillation behavior.
2. Obtain network computer program that can be used for solving complex problem.

CHAPTER II

THOERY AND LITERATURE REVIEW

Secchi A.R., Morari M., and Biscaia E.C. (1993) [2] investigated the concurrent solution of differential-algebraic equations (DAEs) by the waveform relaxation (WR) method, an iterative method for system integration. The WR method obtains the solution of a system DAEs by partitioning into several subsystems. The efficient implementation results in algorithms with a highly parallelizable concurrent fraction and low sequential overhead, making them suitable for coarse- and medium-grain MIMD distributed memory machines. They solved DAEs for a class of dynamic simulation applications of chemical engineering.

Mallya J.U., Zitney S.E., Choudhary S., and Sadtherr M.A. (1997) [3] suggested that a parallel frontal solver that could significantly reduce the wallclock time required to solve a large sparse system of linear equations of large-scale chemical processes. The algorithm utilized both multiprocessing and vectors processing by using a multilevel approach in which frontal elimination was used for the partial factorization of each front. In 1999, they studied the matrix reordering effects on a frontal solver. The algorithm was based on a bordered block-diagonal matrix form.

Nabil A.J., Brice C., and Costas K. (1998) [4] considered a distributed memory message-passing multicomputer for dynamic simulation of chemical processes. Dynamic models for complex multicomponent distillation column consist of large systems of DAEs that were typically nonlinear, sparse, and stiff. The algorithm based on a dynamic block Jacobi-like iteration was applied. The parallel implementation of the modular integration approach had a significant potential for execution time reductions. It also permitted simple implementation of multi-rate integration. The module integrations must be iterated over each time horizon until all module integrations were converged to satisfy a global error criterion for the iterative solution.

Camarda K.V. and Stadtherr M.A. (1999) [5] considered the matrix ordering strategies for process engineering. The simple graph-partitioning algorithm that created a bordered block-diagonal form was suitable for using parallel algorithms for the solution of the highly asymmetric sparse matrices. The method required much less reordering time than previously used graph partitioning methods.

Borchardt J. (2001) [6] considered the plantwide dynamic simulation in the chemical process industry. The DAEs were partitioned into block such as block-structured Newton-type. The simulation was developed on parallel computers with shared memory. The system covered up to 60,000 equations.

Marakis J.G., Chamico J., Brenner G., and Durst F. (2001) [7] used the Monte Carlo method for combined heat transfer analysis on distributed computing environment. The problem of determination of the temperature field formed under the assumption of radiative equilibrium in an enclosure idealizing an industrial furnace. Carlsson P. (2001) [8] considered the optimization with a two-dimensional drying model on distributed computing. Anido L., Santos J., Caeiro M., Rodriguez J., Fernandez M.J., and Llamas M. (2001) [9] and Lalis S, and Karipidis A. (2001) [10] used the Java Web-computing System (JaWS) and Java for general distributed computing framework.

2.1 A Distillation Model

The distillation utilizes vapor and liquid phases at essentially the same pressure and temperature for the coexisting zones. Various kinds of devices such as structured packing and plates or trays are used to bring the two phases into intimate contact. Trays are stacked one above the other and enclosed in a cylindrical shell to form a column. A typical tray-type distillation column plus major external accessories are shown schematically in Figure 2.1. The feed material, which is to be separated into fractions, is introduced at one or more points along the column shell. Because of the difference in gravity between vapor and liquid phases, liquid runs down the column, cascading from tray to tray, while vapor flows up the column, contacting liquid at each tray. Liquid reaching the bottom of the column is partially vaporized in a heat reboiler to provide boilup, which is sent back up the column. The reminder of the bottom liquid is withdrawn as bottoms, or bottom product. Vapor reaching the top of the column is cooled and condensed to liquid in the overhead condenser. Part of this liquid is return to the column as reflux to provide liquid overflow. The remainder of the overhead stream is withdrawn as distillate, or overhead product. In some cases only part of the vapor is condensed so that a vapor distillate can be withdrawn.

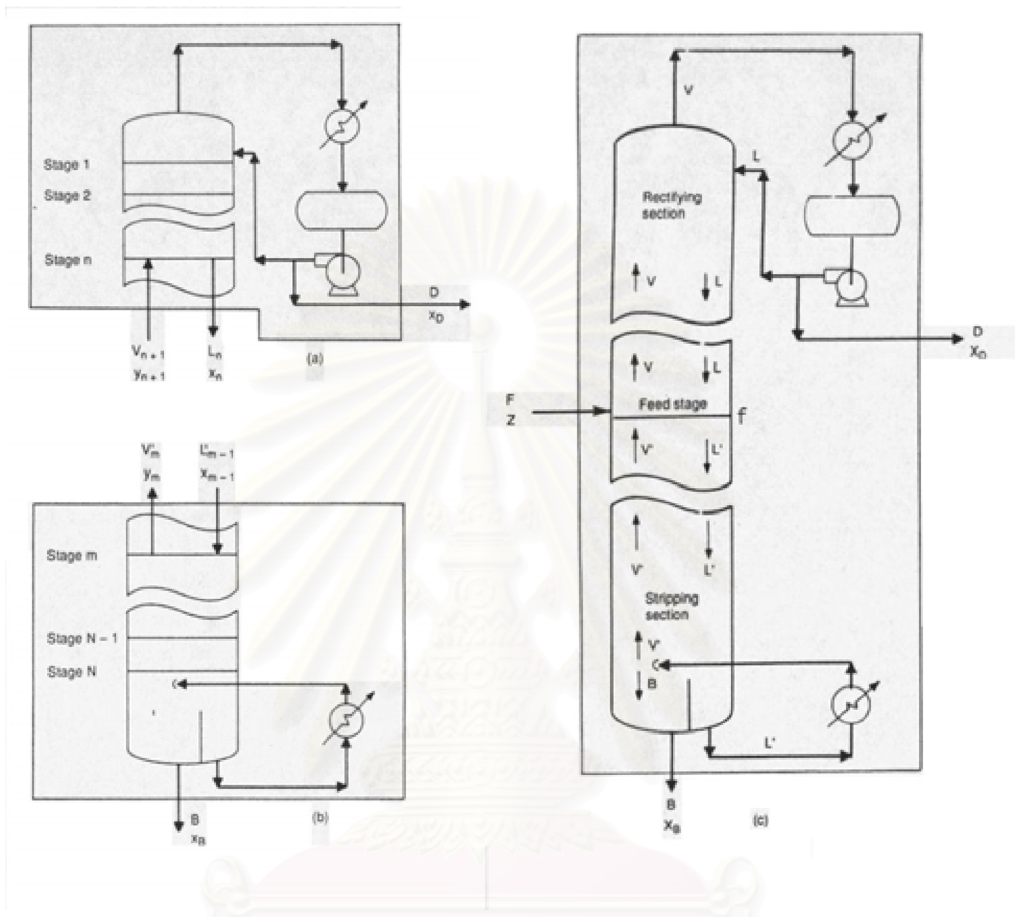


Figure 2.1 Distillation model (a) rectifying (b) stripping (c) overall

(Source: Henry Z. Kister, Distillation design, New York: McGraw-Hill Inc., 1992) [11]

This overall flow pattern in a distillation column provides counter-current contacting of vapor and liquid streams on all the trays through the column. Vapor and liquid phases on a given tray approach thermal, pressure, and composition equilibrium to an extent dependent upon the efficiency of the contacting tray. The lighter (lower-boiling) components tend to concentrate in the vapor phase, while the heavier (higher-boiling) components tend toward the liquid phase. The result in a vapor phase becomes richer in light components as it passes up the column and a liquid phase becomes richer in heavy components as it cascades downward. The overall separation achieved between the distillate and the bottoms depends primarily on the relative volatility of the components, the number of contacting trays, and the ratio of the liquid-phase flow rate to the vapor-phase flow rate. If the feed is introduced at one point along the column shell, the column is divided into an upper section, which is often called the rectifying section, and a lower section, which is often referred to as the stripping section. Dynamic or transient behavior of a continuous-distillation operation is important in determining startup and shutdown procedures, the transition path between steady states, effect of upsets and fluctuations on controllability, residence times and mass-transfer rates, and operating strategies that may involve deliberate imposition of controlled cyclic fluctuations or oscillations. Dynamic behavior may be studied with no controllers in the system to obtain a so-called open-loop response. Alternatively, controllers may be added for certain variables that are to be controlled by manipulating other variables to obtain a so-called closed-loop response. For this latter case, controllers of various levels of complexity [e.g., on-off, proportional (P), proportional with integral action (PI), and proportional with integral and derivative action (PID)] can be considered for various values of tuning parameters, and specific values of known characteristics may be incorporated if desired. Basically, dynamic distillation models are similar to the steady state models, but the dynamic model would consider the accumulative term in each equation. The equations for dynamic models based on first principles must be formulated in terms of fundamental quantities. In chemical engineering, these quantities are mass, energy, and momentum. Under assumptions that are generally valid in chemical engineering systems, these quantities obey the principle of conservation, which is generally stated as

$$\text{Accumulation} = \text{In} - \text{Out} + \text{Generation} - \text{Consumption}$$

When the accumulation is zero, the balance results in an algebraic equation. For a non-zero accumulation, this balance results in a differential equation, which is generally written as

$$\frac{\begin{pmatrix}\text{accumulation of}\\ \text{X within system}\end{pmatrix}}{\text{time period}}=\frac{\begin{pmatrix}\text{flow of X}\\ \text{into system}\end{pmatrix}}{\text{time period}}-\frac{\begin{pmatrix}\text{flow of X}\\ \text{out of system}\end{pmatrix}}{\text{time period}}+\frac{\begin{pmatrix}\text{amount of X}\\ \text{generated in system}\end{pmatrix}}{\text{time period}}$$
$$-\frac{\begin{pmatrix}\text{amount of X}\\ \text{consumed in system}\end{pmatrix}}{\text{time period}},$$

where X is one of the following fundamental quantities: Total mass, mass of a chemical component, energy, and momentum. The model has the proper number of equations when the behavior of the system can be predicted. A correctly formulated model has no degrees of freedom. The concept of degrees of freedom is expressed as

Degrees of freedom = Number of variables – Number of equations

If the number of variables is greater than that of equations, then the system is underspecified and the model must be corrected either by including more appropriate equations or by correctly designating a variable as a specified parameter. If the number of equations is greater than the number of variables, then the system is overspecified and in general no unique solution exists. In this situation, there are one or more dependent equations or constant parameters that ought to be designated as variables.

2.1.1 Binary Distillation Model

We consider the closed-loop response during dynamic distillation of an ideal binary mixture in the column shown in Figure 2.2, under two assumptions of constant relative volatility at a value and constant molar vapor flow for saturated liquid feed to tray. In this model, the two-components mixture is separated by an n-stages distillation column. The liquid holdup on each of the equilibrium trays is assumed to be perfectly mixed, but will vary as liquid rates leaving the trays vary. The vapor rate through all trays of the column is the same, both transiently and at a steady state. The calculation can be described by the set of equations that are defined from four relations. There are two ODEs per tray (total continuity and component continuity equations) and two algebraic equations per tray (vapor-liquid equilibrium relationship and liquid-hydraulic relationship) [12]. Based on the former assumption, it is not necessary to include energy-balance equations for each tray or to treat temperature and pressure as variables. Overhead vapor leaving top tray is total condensed for negligible liquid holdup with condensate flowing to a reflux drum having constant and perfectly mixed molar liquid holdup.

Total continuity

$$\frac{dM_n}{dt} = F_n + L_{n+1} - L_n \tag{2-1}$$

Component continuity

$$\frac{d(M_n x_n)}{dt} = L_{n+1}x_{n+1} + Vy_{n-1} - L_n x_n - Vy_n \tag{2-2}$$

Vapor-liquid equilibrium

$$y_n = \frac{\alpha x_n}{1+(\alpha-1)x_n} \tag{2-3}$$

Liquid-hydraulic

$$L_n = \overline{L}_n + \frac{M_n - \overline{M_n}}{\beta} \tag{2-4}$$

For the condenser-reflux-drum combination:

$$D = V - L_{NT+1} \tag{2.5}$$

$$M_D\left(\frac{dx_B}{dt}\right) = Vy_{NT} - Vx_D \tag{2-6}$$

For the reboiler:

$$B = L_1 - V \tag{2.7}$$

$$M_B\left(\frac{dx_B}{dt}\right) = L_1x_1 - Vy_B - Bx_B \tag{2-8}$$

where F_n is nonzero only for a feed tray, y and x refer to the light component only such that the corresponding mole fractions for the heavy component are (1-y) and (1-x). $\overline{L}$ and $\overline{M}_n$ are the initial steady state values, α is relative volatility, and β is a constant that depends on tray hydraulics. L and V are the liquid and vapor flow rate. B and D are the bottom and distillate product rate.

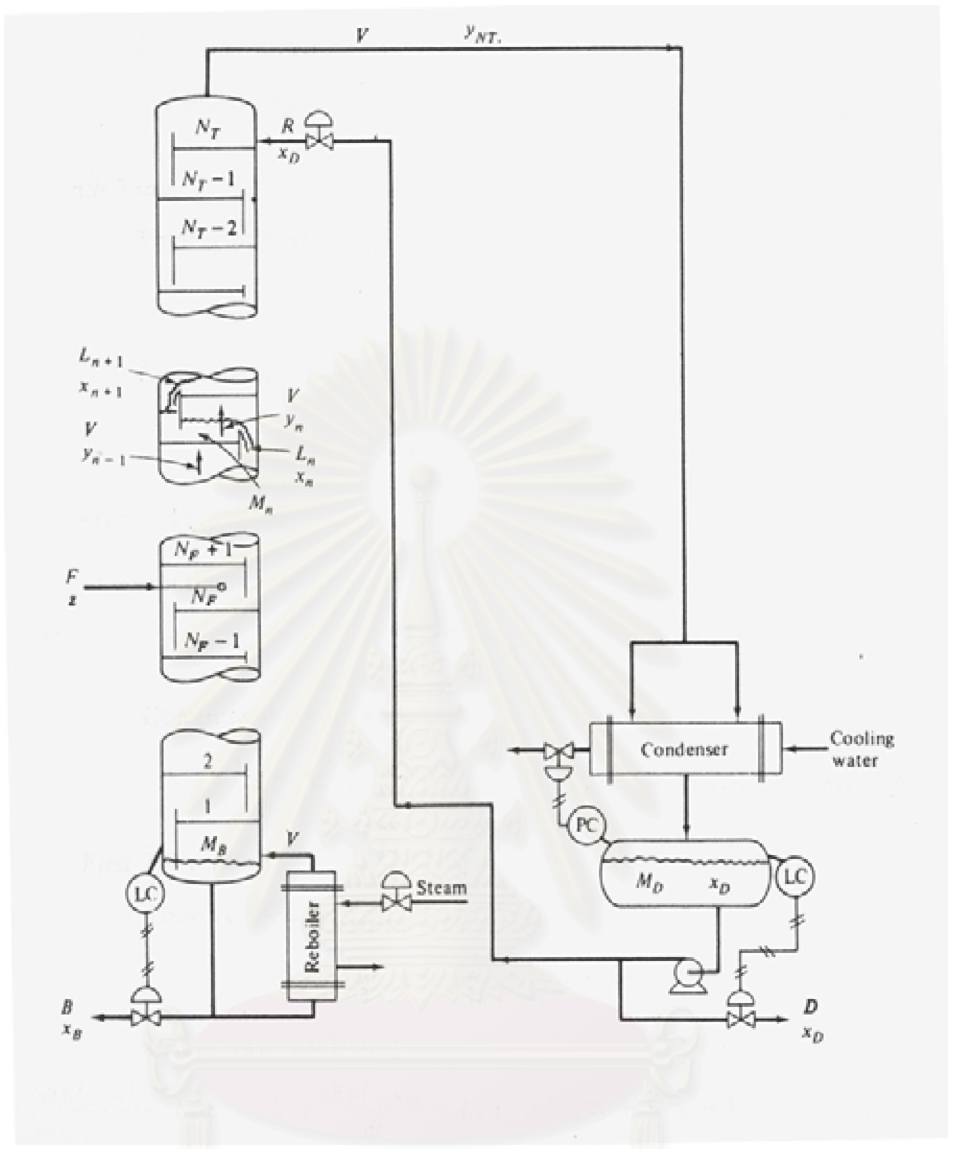


Figure 2.2 Binary distillation column

(Source: W. L. Luyben, Process modeling, simulation, and control for chemical engineering, New York: McGraw-Hill Book Company, 1974) [12]

Let us examine the degrees of freedom of the system. There are 4NT+7 independent equations and 4NT+9 independent variables where NT is the total number of trays.The degree of freedom is equal to two. Thus, there are two controlled variables. The reflux rate R is varied by a proportional-integral (PI) feedback controller to control distillate composition at a set point for the mole fraction x_D of the light component. Holdup of reflux in the line leading back to the top tray is neglected. Under dynamic conditions, y_{NT} may not be equal to x_D. At the bottom of the column, a liquid sump of constant and perfectly mixed molar liquid holdup M_B is provided. A portion of the liquid flowing from this sump passes to a thermosiphon reboiler, with the remainder taken as bottoms product at a molar flow rate B. Vapor boil-up generated in the reboiler is varied by a PI feedback controller to control bottom composition at the set point for the mole fraction x_B of the heavy component. Liquid holdups in the reboiler and lines leading from the sump are assumed to be negligible. The composition of the boil-up y_B is assumed to be in equilibrium with x_B.

The two PI-controller equations are

$$V = \overline{V} - K_{CB}\left(E_B + \frac{1}{\tau_B}\int_t^{t+\Delta t} E_B dt\right) \tag{2.9}$$

$$L_{NT+1} = \overline{L}_{NT+1} - K_{CD}\left(E_D + \frac{1}{\tau_D}\int_t^{t+\Delta t} E_D dt\right) \tag{2-10}$$

where $\overline{V}$ and $\overline{L}_{NT+1}$ are initial values, K_C and τ are respectively feedback-controller gain and feedback-reset time for integral action, and E is the error or deviation from the set point as given by

$$E_B = x_B^{set} - x_B \tag{2.11}$$

$$E_D = x_D^{set} - x_D \tag{2-12}$$

2.1.2 Multicomponent Distillation Model

In principle, multicomponent equations are very similar to those for binary systems, but the solution involves substantially more computational effort. The model of open-loop distillation column is shown in Figure 2.3. For the sake of generality, it is shown that feed streams are introduced and both liquid and vapor side streams are withdrawn from each tray. Energy can be added or removed from each tray. In an actual column, however, many of these values would be zero. The theoretical model for this simulation has the assumption as follows:

- Liquid on the tray is perfectly mixed and incompressible.

- Tray vapor holdups are negligible.

- Dynamics of the condenser and the reboiler will be neglected.
- Vapor and liquid are in thermal equilibrium (same temperature) but not in the phase equilibrium.
- A general nth tray is shown in Figure 2.4.

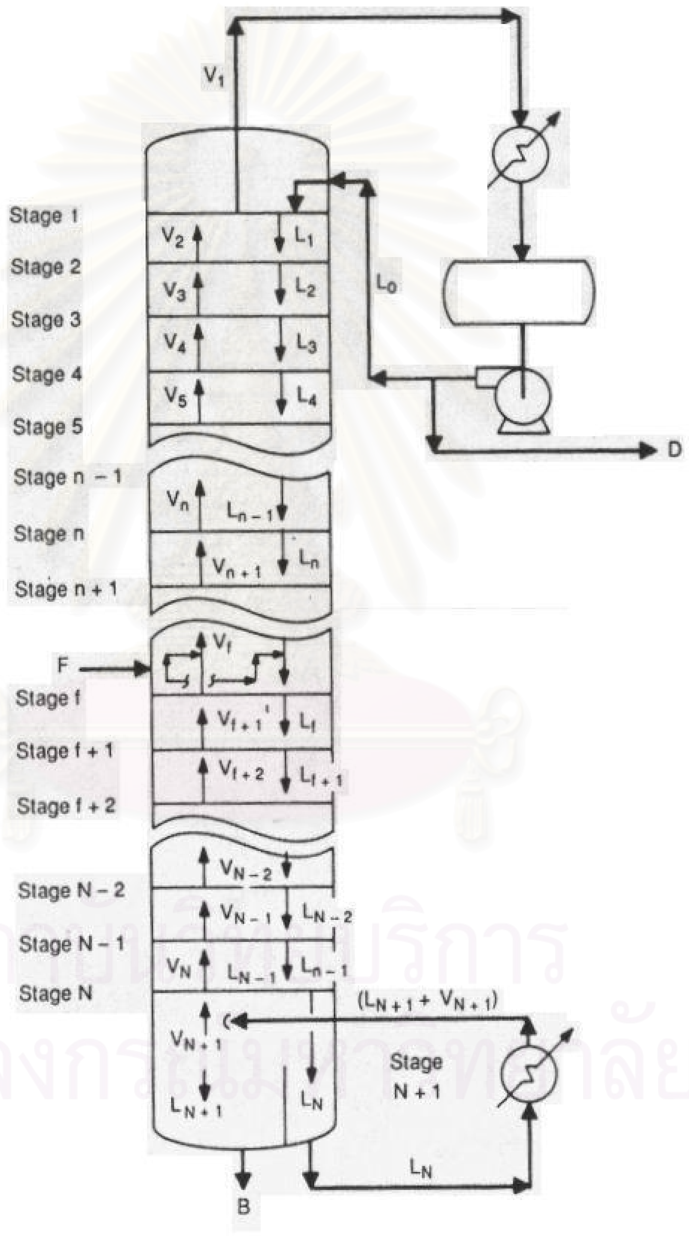


Figure 2.3 Open-loop distillation.

(Source: Henry Z. Kister, Distillation-design-, New York: McGraw-Hill Inc, 1992) [11]

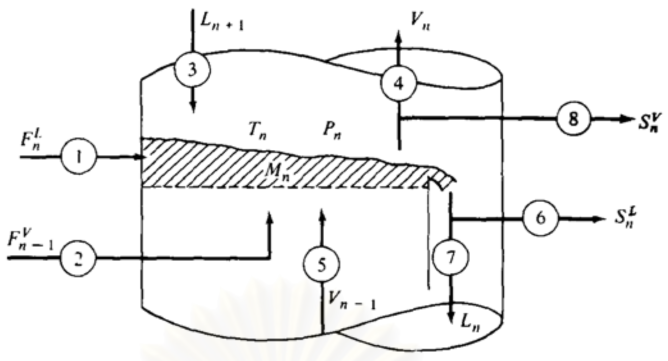


Figure 2.4 nth tray of multicomponent column.

Table 2.1 Streams on nth tray

Number	Flow rate	Composition	Temperature
1	F_n^L	x_{nj}^F	T_n^F
2	F_{n-1}^V	$y_{n-1,j}^F$	T_{n-1}^F
3	L_{n+1}	$x_{n+1,j}$	T_{n+1}
4	V_n	y_{nj}	T_n
5	V_{n-1}	$y_{n-1,j}$	T_{n-1}
6	S_n^L	x_{nj}	T_n
7	L_n	x_{nj}	T_n
8	S_n^V	y_{nj}	T_n

(Source: W. L. Luyben, Process modeling, simulation, and control for chemical engineering, New York: McGraw -Hill Book Company, 1974 [11]

In this system, the C-components mixture is separated by N-stages distillation column. The distillation calculation can be described by the set of equations.

Total continuity

$$\frac{dM_n}{dt} = L_{n+1} + F_n^L + F_{n-1}^V + V_{n-1} - V_n - L_n - S_n^L - S_n^V \tag{2-13}$$

Component continuity

$$\frac{dM_n x_{nj}}{dt} = L_{n+1}x_{n+1} + F_n^L x_{nj}^F + F_{n-1}^V y_{n-1,j}^F + V_{n-1}y_{n-1,j}$$

$$-V_n y_{nj} - L_n x_n - S_n^L x_{nj} - S_n^V y_{nj} \tag{2-14}$$

Phase equilibrium

$$y_{nj} = f\left(x_{nj}, P_n, T_n\right) \tag{2-15}$$

Component summation

$$\sum_{j=1}^{C} x_{nj} = 1 \tag{2-16}$$

$$\sum_{j=1}^{C} y_{nj} = 1 \tag{2-17}$$

Energy balance

$$\frac{dM_n h_n}{dt} = L_{n+1} h_{n+1} + F_n^L h_{nj}^F + F_{n-1}^V H_{n-1,j}^F + V_{n-1} H_{n-1,j}$$

$$-V_n H_{nj} - L_n h_n - S_n^L h_{nj} - S_n^V H_{nj} \tag{2-18}$$

where n, j are the indices of the nth tray and jth component.

There are altogether N(2C+3) equations. The unknowns are internal vapor rates (N), internal liquid rates (N), tray temperatures (N), liquid mole fractions (CN), and vapor mole fractions (CN). The total number of variables is N(2C+3) unknowns.

2.1.3 Vapor Pressure

In the ideal system, Raoult's and Dalton's laws are applied and vapor-liquid equilibrium can be calculated from vapor pressures. Vapor pressure is represented by Antoine's equation.

$$\log P = A - \frac{B}{T + C} \tag{2-19}$$

where A, B, and C are Antoine's constants. If A and B are unknown. Antoine's constant can be calculated by equations 2-20, 2-21.

$$B = \frac{\left(T_{top} + C\right)\left(T_{bottom} + C\right)\log\left(P_{top} / P_{bottom}\right)}{\left(T_{top} - T_{bottom}\right)} \tag{2-20}$$

$$A = \log P_{bottom} - \frac{B}{\left(T_{bottom} + C\right)} \tag{2-21}$$

2.1.4 Equilibrium Ratios

For many systems the equilibrium ratio K_j can be used:

$$K_j = \frac{y_j}{x_j} \tag{2-22}$$

The real value of K_j lies in the fact that charts of $K = f(T, P)$ are available. The De Priester charts are reproduced here as Figure 2.5 and Figure 2.6. They are for hydrocarbon systems only, which is somewhat restrictive.

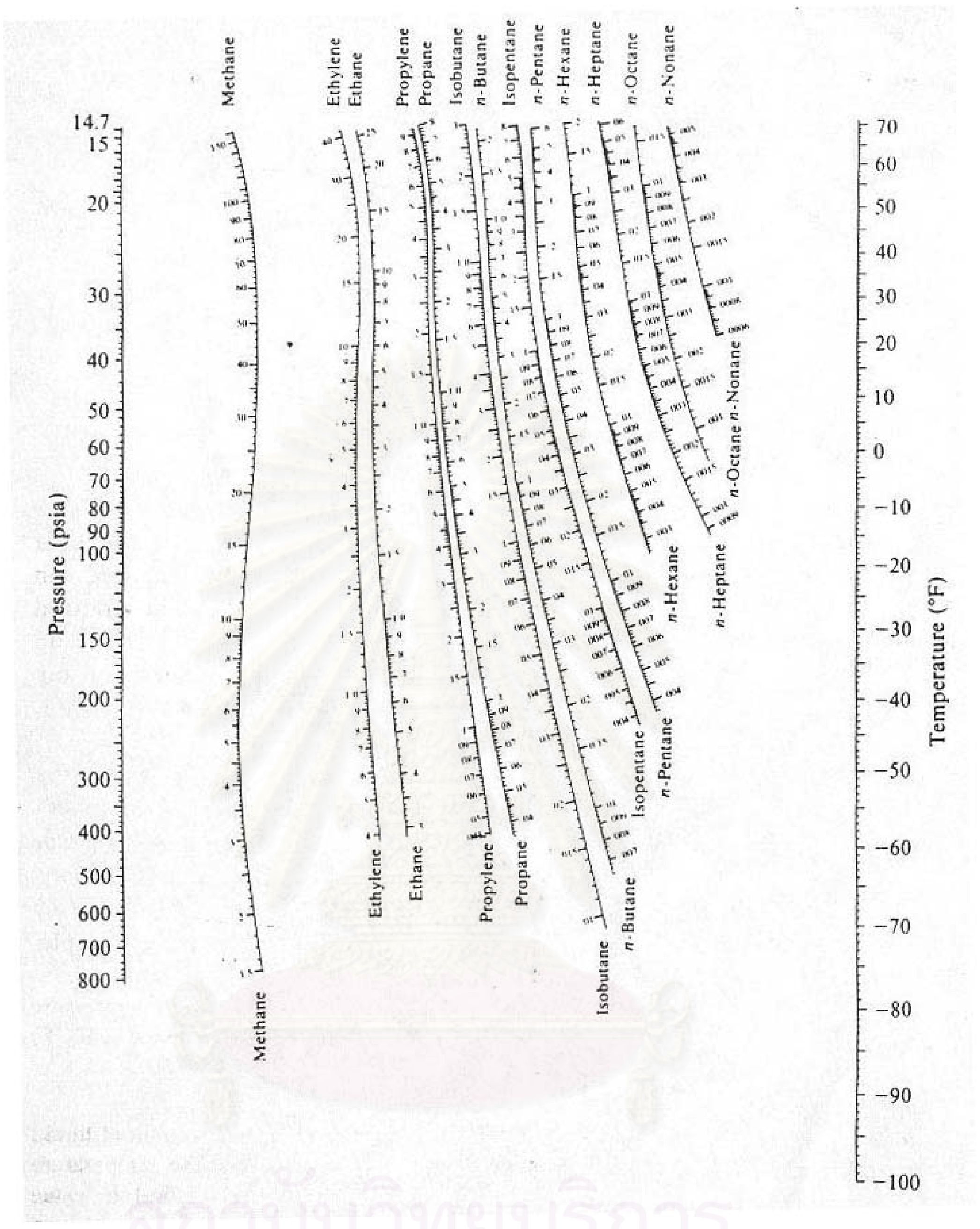


Figure 2.5 DePriester chart – low temperature range.

(Source: J. M. Smith and H. C. Van Ness, Introduction to chemical engineering thermodynamics, 4th Edition, New York: McGraw-Hill Book Company, 1987) [13]

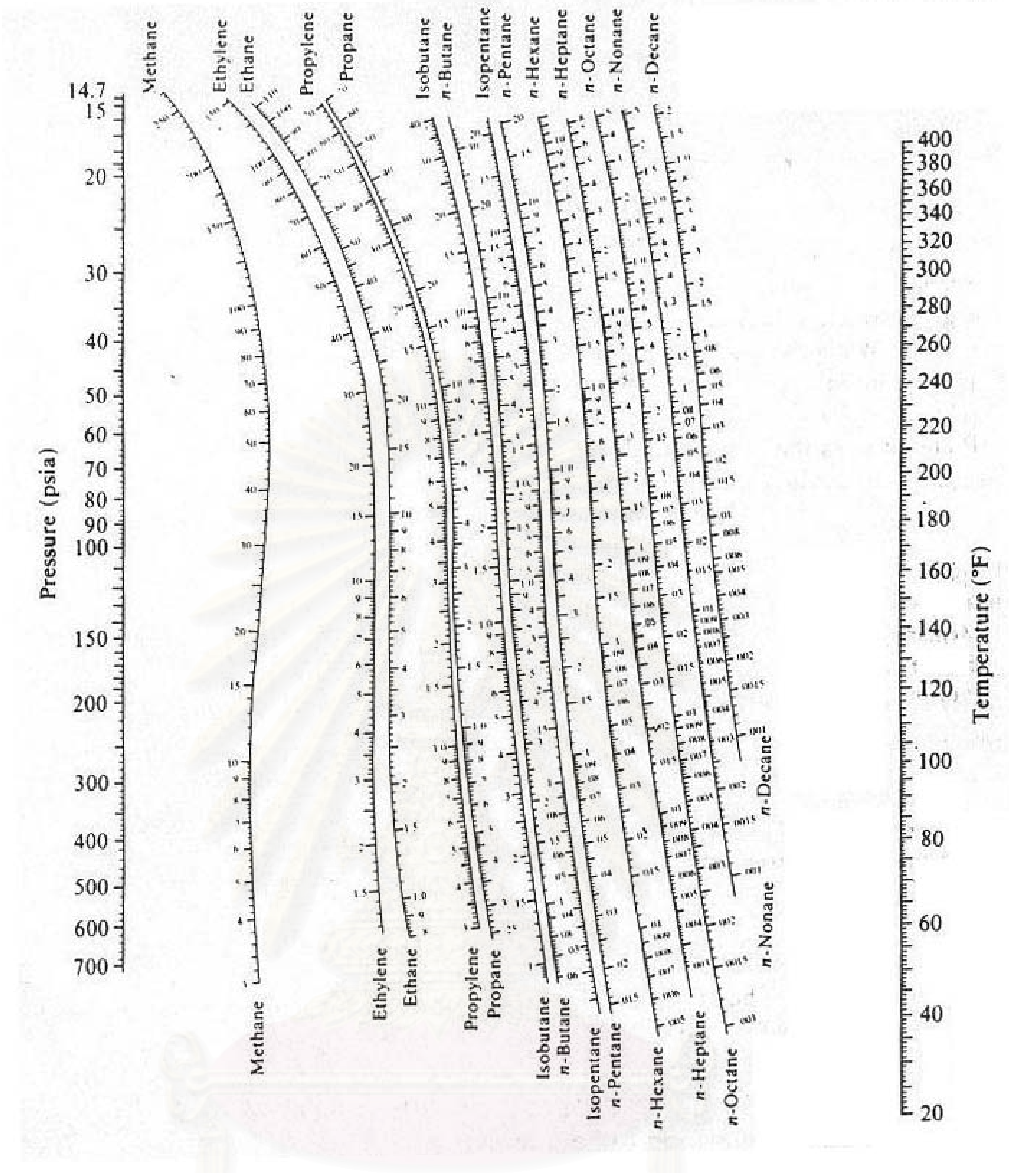


Figure 2.6 DePriester chart – high temperature range.

(Source: J. M. Smith and H. C. Van Ness, Introduction to chemical engineering thermodynamics, 4th Edition, New York: McGraw-Hill Book Company, 1987) [13]

2.1.5 The Bubble-Point (BP) Method

The BP method uses a form of the phase equilibrium equation and component summation equation to calculate the stage temperatures. By definition a saturated-liquid stream is at the boiling point (or bubble point). The first and tiniest of bubbles formed has a composition different from the liquid, but the amount of material in the bubble is too small to change the composition of the liquid. Obviously, the liquid and the bubble are in equilibrium. Mathematically, the design equation for bubble point calculation is

$$\sum_{i=1}^{NC} y_i = \sum_{i=1}^{NC} K_i x_i = 1.0 \qquad (2\text{-}23)$$

2.1.6 Murphree Tray Efficiency

Murphree tray efficiency is the ratio of the change of composition on the actual stage to the change that would occur on a theoretical stage.

$$E_{nj} = \frac{y_{nj} - y_{nj-1}}{y^*_{nj} - y_{nj-1}} \qquad (2\text{-}24)$$

where y^*_{nj} is the composition of vapor in equilibrium with the liquid leaving the tray.

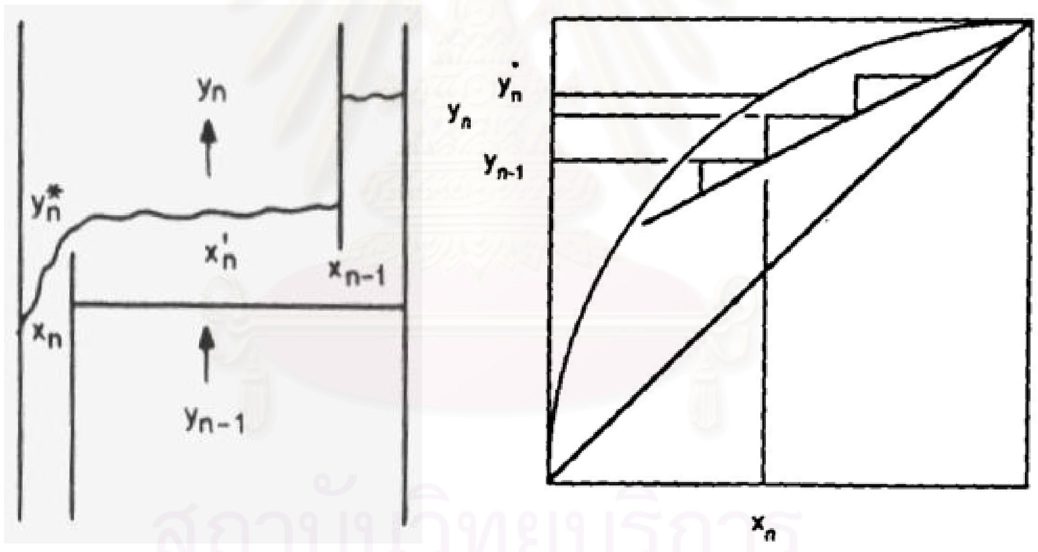


Figure 2.7 Murphree tray efficiency

(Source: W. L. Luyben, Practical distillation control, Van Nostrand Reinhold, New York, 1992) [14]

2.1.7 Enthalpy

Liquid and vapor enthalpy are functions of temperature, pressure, and composition. In energy balance, we use some simple enthalpy equations.

Liquid enthalpy

$$h = C_{PL}T \qquad (2\text{-}25)$$

where C_{PL} is molal specific heat of liquid at constant pressure.

Vapor enthalpy

$$H = C_{PV}T + \lambda_v \qquad (2\text{-}26)$$

where C_{PV} is molal specific heat of vapor at constant pressure.

λ_v is heat of vaporization.

2.1.8 Liquid Holdup

Francis weir formula is used to calculate the liquid hold up on a tray.

$$L = C\rho l_W h_{ow}^{3/2} \qquad (2\text{-}27)$$

where L is liquid rate.

C is a constant for a unit conversion.

ρ is liquid density.

l_w is weir length.

h_{ow} is liquid height over weir.

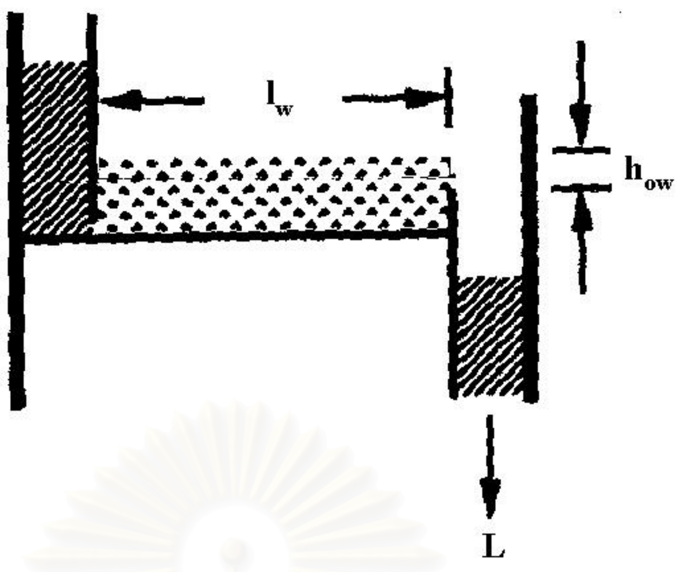


Figure 2.8 Francis weir

2.2 Parallel Computing

2.2.1 Parallel Performance

The performance of parallel application is often measured in terms of speed up that known as Amdahl's Law. The achievable speedup is shown in equation 2-28.

$$S(N) = \frac{T(1)}{T(N)} \tag{2-28}$$

where T(1) is the processing time of the program when run on one processor, and T(N) is the time taken to solve the problem using N processors. We get the good performance when speedup is above one.

Gustafson-Barsis Law suggests another speedup performance that is shown in equation 2-29.

$$S(N) = N - (N-1)\alpha \tag{2-29}$$

where α is the fraction of the program that must be run in sequence. Gustafson-Barsis interpreted the Amdahl's Law that parallelism can be used to increase the parallel size of the problem. If one processor is used, it must compute both the serial part and the parallel parts, and the parallel parts take N times as long to run on a single processor. If N parallel processors are used, the problem is scaled up so that N parallel processors execute the serial and parallel parts of the program as shown in Figure 2.9.

The parallel efficiency measures the contribution of each processor to the parallel solution when N processors are employed. The parallel efficiency is defined as

$$E(N)=\frac{S(N)}{N} \tag{2-30}$$

where E(N) is equal to the average efficiency per processor when the problem is run with N parallel processors.

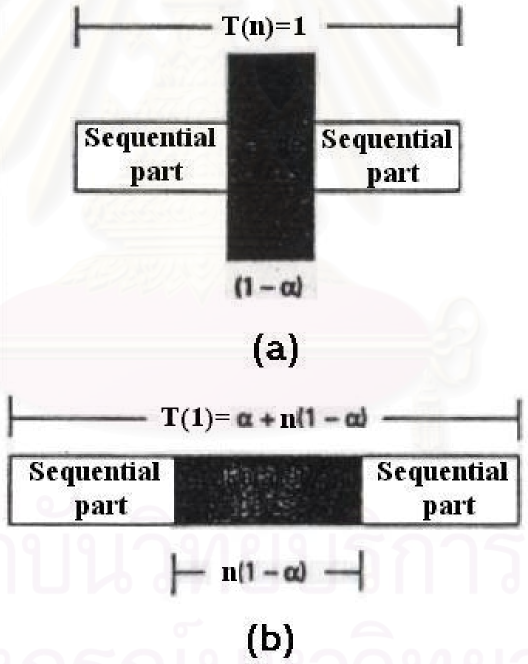


Figure 2.9 Gustafson-Barsis Law of speed up. Notice that (a) time for parallel processors (b) time for a single processor

2.2.2 Message-Passing

The Message-passing model poses a set of processes that have only local memory but are able to communicate with other processes by sending and receiving messages. It is a defining feature of the message-passing model that data transfer from the local memory of one process to that of the others requires operations to be performed by both processes.

The message-passing interface (MPI) is a standard protocol for writing message-passing programs. It was developed during 1993 and 1994 by an international group of application scientists, computer vendors, and software writers called the MPI Forum. The goal of MPI is to provide a standard library of routines for writing portable and efficient message-passing programs. MPI is not a language; it is a specification of a library of routines that can be called from C and FORTRAN 77 programs. MPI provides a rich collection of point-to-point communication routines and collective operations for data movement, global computation, and synchronization. MPI also defines a number of important features such as derived data types and communication contexts. Several functional implementations of the MPI specification are currently available. Both free public domain and commercial implementation already exist for tightly coupled parallel computers and clusters of workstations.

An MPI application can be visualized as a collection of concurrent communicating tasks. A program includes code written by the application programmer that is linked with a function library provided by the MPI software implementation. Each task is assigned a unique rank within a certain context: an integer number between 0 and n-1 for an MPI application consisting of n tasks. These ranks are used by MPI tasks to identify each other in sending and receiving messages, to execute collective operations, and to cooperate in general. MPI tasks can run on the same processor or on different processors concurrently. For an application program, sending a message to a task on the same or another machine is a transparent operation. MPI automatically selects the most efficient communication mechanism available on a particular machine or between machines. The use of ranks makes all cooperative operations independent of the physical location of the participants.

As in most computing systems, the environment in which an MPI application will run may require some set up before the MPI routines are called in an application. MPI provides the following initialization routine.

MPI_Init(&argc, &argv)

It must be called before any other MPI routine is used. All MPI tasks that participate in the computation are available after MPI is initialized. At the end of an MPI application, the programmer must call the function MPI_Finalize() to clean up the MPI state. Once this function is activated, no other MPI function can be called. It is important to ensure that all pending communication is finished before this function is called. Tasks in MPI are allowed to belong to named groups. A group in MPI is an object that can be accessed via a handle of the predefined type MPI_Group. Task groups provide contexts through which MPI operations can be restricted to only the members of a particular group. The members of a group are assigned unique identifiers, so called ranks, with in the group. A group is an ordered set of ranks that is contiguous and starts from zero. An important requirement in all message-passing systems is to guarantee a safe communication space in which unrelated messages are separated from one another, for example, library message can be sent and received without interference from other messages generated in the system.

In MPI, where is no virtual machine, using just a message tag is not enough to safely distinguish library messages from user messages. The concept of communicator is introduced in MPI to achieve this safe communication requirement. A communicator can be thought of as a binding of a communication context to a group of tasks. A communicator is an object that can be accessed via a handle of type MPI_COMM. Communicators can be classified into intracommunicators, for operations within a single group of tasks, and intercommunicators, for operations between different groups of tasks. When an MPI application starts, all tasks are associated to a “world” communicator. When a new context is needed, the program makes a synchronizing call to derive the new context from an existing one. MPI provides the predefined communicator MPI_COMM_WORLD as the default communicator. Once MPI_Init() is called, this default communicator defines a single context including the set of all MPI tasks available for the computation. The communicator MPI_COMM_WORLD has the same value in all processes and cannot be changed during the lifetime of a task. MPI also provides the predefined communicator MPI_COMM_SELF, which includes only the calling process itself. The tasks involved in a communicator are assigned consecutive integer identifiers between zero and the size of communicator's group minus one. These identifiers, which called ranks, are used to distinguish the different tasks within the same group. A task can find out its rank within a communicator by calling the function MPI_Comm_rank(). The size of the group associated with a communicator can be determined by calling the function MPI_Comm_size(). This function takes an existing communicator and returns the size of its corresponding group.

A communication among MPI tasks is based on the message-passing paradigm. MPI utilizes a rich set of functions for sending and receiving messages. Communication between two tasks involves the following components: Sender, Receiver, Message data, Message tag (which helps multiple messages between two tasks to be handled in order), and Communicator (which provides a context for communication). The basic functions to send and receive message in MPI are the blocking send and blocking receive. There are several variations of functions that facilitate different kinds of communication modes. MPI supports the standard send, blocking receive, buffered send, synchronous send, and ready send modes. In standard send mode, the sender will block until its message has been safely copied into either a matching receive buffer or a temporary system buffer. It is up to the MPI implementation to decide whether or not a message should be buffered. Once the send call returns, the send buffer can be overwritten and reused for other purposes by the sender. The following function is the standard send in MPI.

MPI_Send(buf, count, data_type, to_whom, tag, communicator)

This function will send the message stored starting at address buf to the task whose rank is given as to_whom. The message consists of count elements, each of which is of type data_type. The message tag is given as tag. Both the sender and the receiver must be part of the same communicator. If the buffer is used to store the outgoing message, the sender will continue without having to wait for a matching receive to be posted. The standard receive function in MPI is the blocking receive. A call to this function will not return until it receives the message it is expecting in its buffer. The following function is the blocking receive in MPI.

MPI_Recv(buf, count, data_type, from_whom, tag, communicator, status)

This receive will select a message with a matching sender (from_whom) and a matching message tag (tag) for receipt into the buffer (buf). Additional status information will be returned in (status). The status field is useful particularly when the source and/or the tag of the received message in not known to the receiver, as a result of using wild cards. The status is normally a structure consisting of two fields, MPI_SOURCE and MPI_TAG, for the rank of the sender and the tag of the received message, respectively. Again, the sender and the receiver should be participants in communicator. Using buffered send mode, message buffer is guaranteed that a buffered send may return whether or not a matching receive call has posted. Once the message information is buffered, the send call will return and the send buffer becomes reusable. The format of the buffered send is the same as in the standard send. The only difference is the addition of the letter B to the name of the send function, as MPI_Bsend(). Synchronous communication can be accomplished if both the sender and receiver block until the send and receive calls are posted and the communication is complete. Since the standard receive is already blocking, we just need a blocking send to be able to accomplish synchronous communication. MPI also provides the function MPI_Ssend() for this purpose. It has the same format as the standard send and it can start without having to wait for a matching receive call to be posted. However, it will not be complete until a matching receive has been posted and the receiver has started to receive the message. A send in the ready mode can be started only after a matching receive has been posted. The function MPI_Rsend() is provided for this purpose. The completion of the ready send, however, does not depend on the status at the receiving end.

Basic MPI data types include all commonly encountered C and FORTRAN native types. The definition allows for interoperation in a collection of heterogeneous machines. MPI is able to provide any bit ordering or other transformations required in the environment. MPI also provides the concept of derived data types. Derived data types consist basically of a sequence of data types and integer offsets for elements in the sequence. This concept allows the exchange of complex information elements without incurring needless overhead.

Table 2.2 Basic MPI data types for FORTRAN

MPI data type	FORTRAN data type
MPI_BYTE	
MPI_CHARACTER	CHARACTER
MPI_COMPLEX	COMPLEX
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_INTEGER	INTEGER
MPI_LOGICAL	LOGICAL
MPI_PACKED	
MPI_REAL	REAL

(Source: H. El-Rewini and T. G. Lewis, Distributed and parallel computing, Greenwich: Manning Publications Co., 1998) [15]

Table 2.3 Basic MPI data types for C

MPI data type	C data type
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	double
MPI_FLOAT	float
MPI_INT	int
MPI_LONG	long
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_SHORT	short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

(Source: H. El-Rewini and T. G. Lewis, Distributed and parallel computing, Greenwich: Manning Publications Co., 1998) [15]

Synchronization constructs are used to force a certain order of execution-among the activities of parallel tasks. In some cases, parallel tasks are required to synchronize with each other at a given point during the execution. Members of a group may need to wait at a synchronization point until all tasks reach the same point. Synchronization in MPI can be achieved using message-passing and barrier operations. Tasks in a group can synchronize at a synchronization point using a barrier. No task can proceed beyond the barrier until all tasks have checked in at that barrier. The group may include all tasks or only a subset of the tasks, depending on the communicator. The construct MPI_Barrier() takes a communicator as input as follows.

MPI_Barrier(communicator)

Barrier synchronization is achieved by having all tasks in the communicator's group call the function MPI_Barrier(). A task waits at the barrier until all tasks referenced by the communicator reach the barrier. A call to MPI_Barrier() returns after all the communicator's group members have executed their calls to this function. MPI supports a broad variety of data movement collective functions. The basic operations supported are broadcast, scatter, and gather. In a broadcast, one process sends the same message to every member in the group.

MPI_Bcast(buffer, n, data_type, root, communicator)

All members of the communicator's group, using the same arguments for the root and communicator, must call this function. The contents of the root's buffer will be copied to the buffers of all tasks (Figure 2.10).

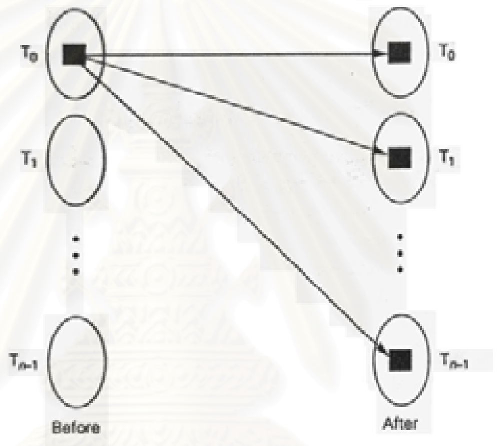


Figure 2.10 A broadcast from task T_0

(Source: H. El-Rewini and T. G. Lewis, Distributed and parallel computing, Greenwich: Manning Publications Co., 1998) [15]

A scatter operation allows one process to send a different message to each member.

MPI_Scatter(sbuf, n, stype, rbuf, m, rtype, rt, communicator)

In scatter, the send buffer at the root is divided into a number of segments, each of size n. The first n elements in the root's send buffer are copied into the receive buffer of the first member in the group. The second n elements in the root's send buffer are copied into the receiver buffer of the second member, and so on (Figure 2.11(a)).

In gather, which is the dual operation of scatter, one process will receive a message from each member in the group.

MPI_Gather(sbuf, n, stype, rbuf, m, rtype, rt, communicator)

This function, each task sends the contents of its send buffer to the root task. The root receives the messages and stores them in rank order. The send buffer of the first member in the group is copied into the send m location in the receive buffer of the root, and so on (Figure 2.11(b)). The parameters and their meanings are shown in Table 2.4.

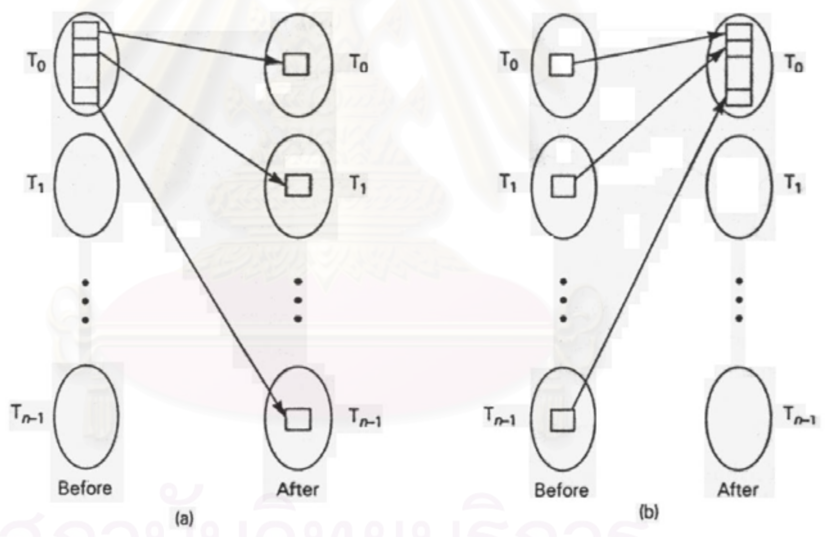


Figure 2.11 (a) Scatter from the task T_0 (b) Gather at the root task T_0

(Source: H. El-Rewini and T. G. Lewis, Distributed and parallel computing, Greenwich: Manning Publications Co., 1998) [15]

Table 2.4 The scatter and the gather parameters and meanings.

Parameter	Meaning
Sbuf	Starting address of the send buffer
N	Number of elements sent to each task by the root
Stype	Type of each element in the send buffer
Rbuf	Starting address of the receive buffer
M	Number of data elements in the receive buffer
Rtype	Type of each element in the receive buffer
Rt	Rank of the sending task or receiving task
Commnuicator	Communicator

(Source: H. El-Rewini and T. G. Lewis, Distributed and parallel computing, Greenwich: Manning Publications Co., 1998) [15]

2.3 Numerical Methods

2.3.1 Newton-Raphson Method

The Newton-Raphson method is one of the most powerful methods for solving one-dimensional or multi-dimensional and systems of nonlinear equations. This method is a powerful numerical method for thermodynamic calculation, such as the BP method. The analytical derivative is readily available for these calculations. Figure 2.12 presents the Newton-Raphson method graphically.

The method requires an initial estimate of the solution x_1. An analytical expression gives the derivative of the objective function at the estimate. The derivative is the tangent to the curve at this point. The next estimate x_2 is computed where the tangent intersects the x-axis at $f = 0$. These convergence methods require initial estimates for the solution. During the dynamic simulation, the solution at the previous time step is used for these estimates. This is very close to the solution at the current time step. The algebraic equations are solved very quickly, in only a few iterations.

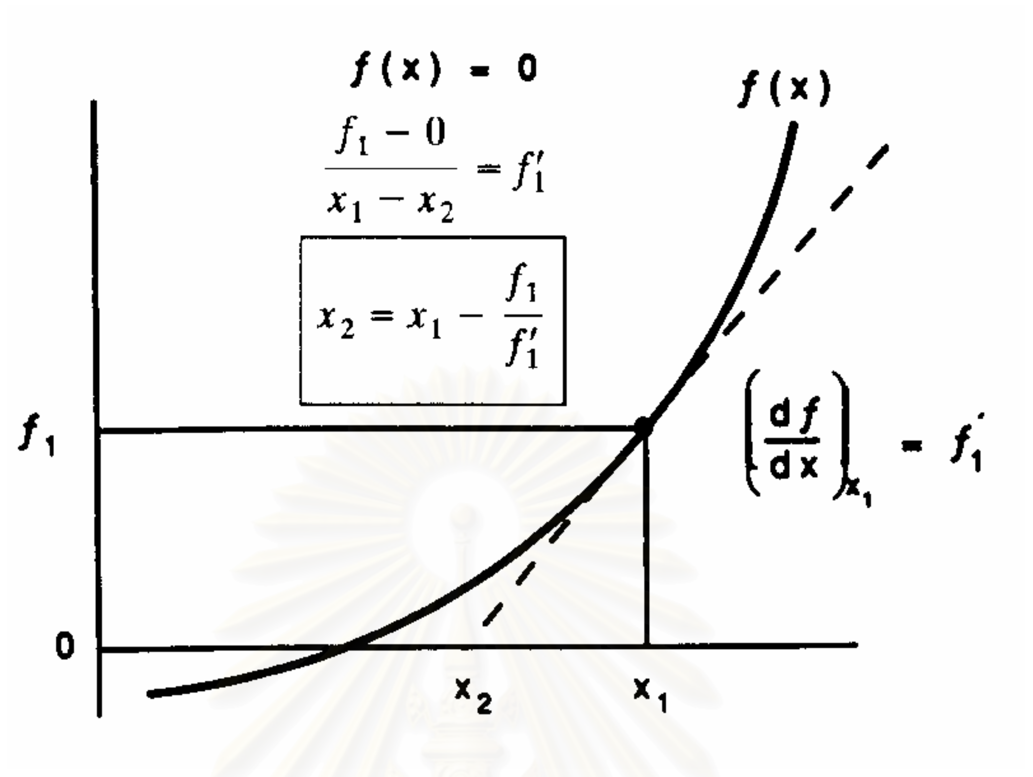

$f(x) = 0$
$\frac{f_1 - 0}{x_1 - x_2} = f_1'$
$x_2 = x_1 - \frac{f_1}{f_1'}$
$f(x)$
f_1
$\left(\frac{df}{dx}\right)_{x_1} = f_1'$
0
x_2
x_1

Figure 2.12 Newton-Raphson convergence.

(Source: W. L. Luyben, Practical Distillation Control, New York: Van Nostrand Reinhold, 1992) [14]

2.3.2 Euler Integration Method

The simplest possible numerical integration scheme is Euler integration, illustrated graphically in Figure 2.13. Assume the form of ODE be

$$\frac{dx}{dt} = f(x,t) \tag{2-31}$$

where f is a nonlinear function. An initial condition for x is needed.

$$x_{(0)} = x_0 \qquad \text{at } t = 0 \tag{2-32}$$

Now if it is moved forward in time by a small step Δt to $t = \Delta t$, it can be got an estimate of the new value of x at $t = \Delta t$, $x_{(\Delta t)}$ from a linear extrapolation using the initial time rate of change of x. The new value of x is approximately equal to the old value of x plus the product of the derivative of x times the step size.

$$x_{(\Delta t)} = x_{(0)} + \left(\frac{dx}{dt}\right)_{t=0} \Delta t \tag{2-33}$$

$$x_1 = x_0 + f(x_0, 0)\Delta t \qquad (2\text{-}34)$$

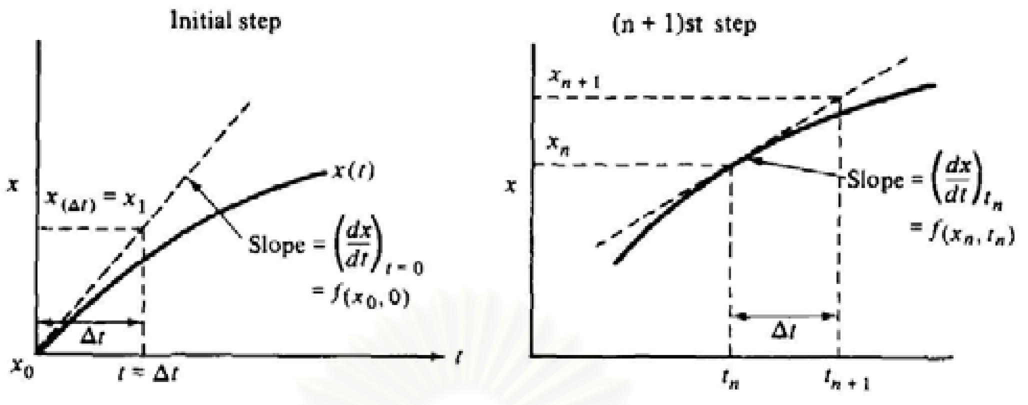

Initial step
x
$x_{(\Delta t)} = x_1$
x(t)
Slope = $\left(\frac{dx}{dt}\right)_{t=0}$ = $f(x_0, 0)$
Δt
x_0
$t = \Delta t$
t
(n + 1)st step
x_{n+1}
x_n
x
Slope = $\left(\frac{dx}{dt}\right)_{t_n}$ = $f(x_n, t_n)$
Δt
t_n
t_{n+1}

Figure 2.13 Graphical representation of Euler integration.

(Source: W. L. Luyben, Process modeling simulation and control for chemical engineering, New York: McGraw-Hill Book Company, 1974.) [12]

CHAPTER III

IMPLEMENTATION

3.1 Binary Distillation Simulation Implementation

First, the parallel algorithm will be demonstrated with the simplified binary distillation column. Its model was described in previous chapter. The following procedure as shown in Figure 3.1 can be used to solve the equilibrium-stage model. The procedure starts from the bottom tray and proceeds up through the column. At each instant in time, all holdups and all liquid compositions were known. The calculation steps were proceeded.

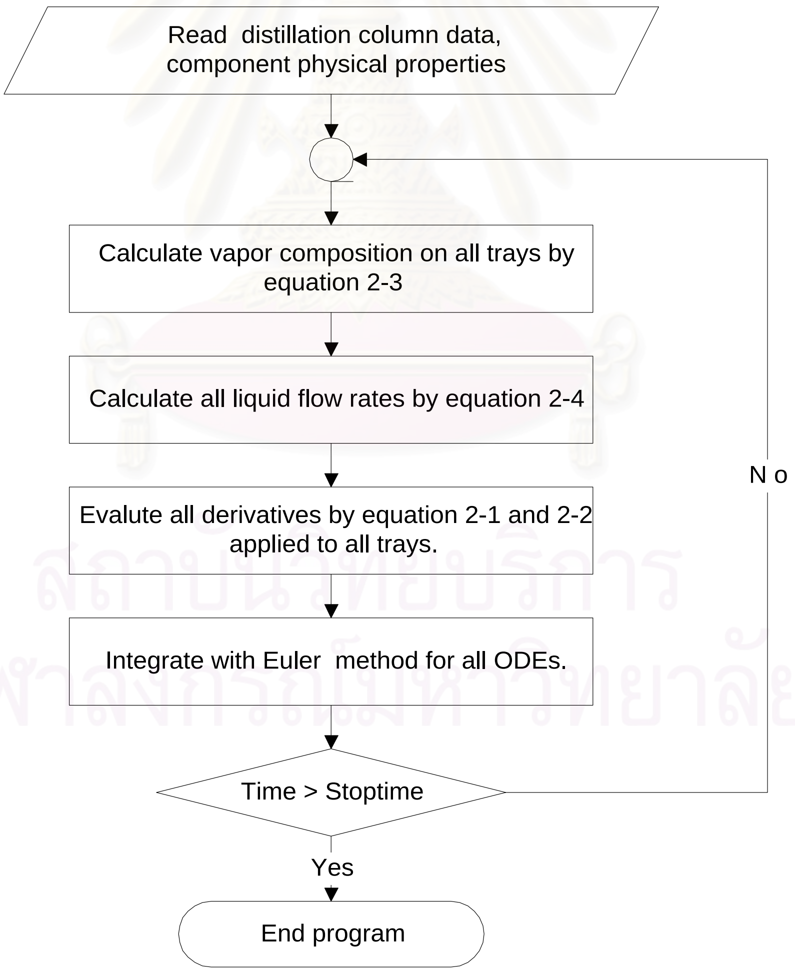

Read distillation column data, component physical properties
Calculate vapor composition on all trays by equation 2-3
Calculate all liquid flow rates by equation 2-4
Evalute all derivatives by equation 2-1 and 2-2 applied to all trays.
Integrate with Euler method for all ODEs.
Time > Stoptime
N o
Yes
End program

Figure 3.1 Algorithm for solving binary distillation simulation

3.2 Multicomponent Distillation Simulation Implementation

The general multicomponent model was described in previous chapter. The extension of the simple ideal binary system considered in the preceding section to a nonideal multicomponent column is not difficult. The only changes that have to be made to the basic structure of the solution algorithm are:

1. More ODEs must be added per tray. Each component can be written one component balance per tray. There are NC equations for NC components on each tray.
2. One energy balance per tray.
3. The Bubble point method will be used for obtaining the solution.

The simulated column is assumed to have the following equipment configurations and conditions:

1. There is one feed plate onto which vapor feed and liquid feed are introduced.
2. Pressure is constant and known on each tray. It varies linearly up the column from the base to the top.
3. Coolant and steam dynamics in the condenser and reboiler are negligible.
4. Distillate vapor and liquid products taken off the reflux drum and are in equilibrium. Dynamics of the vapor space in the reflux drum and throughout the column are negligible.
5. Liquid hydraulics are calculated from the Francis weir formula.
6. Volumetric liquid holdups in the reflux drum and column base are held perfectly constant by changing the flow rates of bottom product and liquid distillate product.

7. Dynamic changes in internal energies on the trays are much faster than the composition or total holdup changes, so the energy equation on each tray (Equation 2-18) is just algebraic.

8. Reflux and heat input to the reboiler is simply held constant.

The following procedure can be used to solve the stage multicomponent model. The procedure can be proceeded the same manner as that for binary column. The algorithm is shown in Figure 3.2.

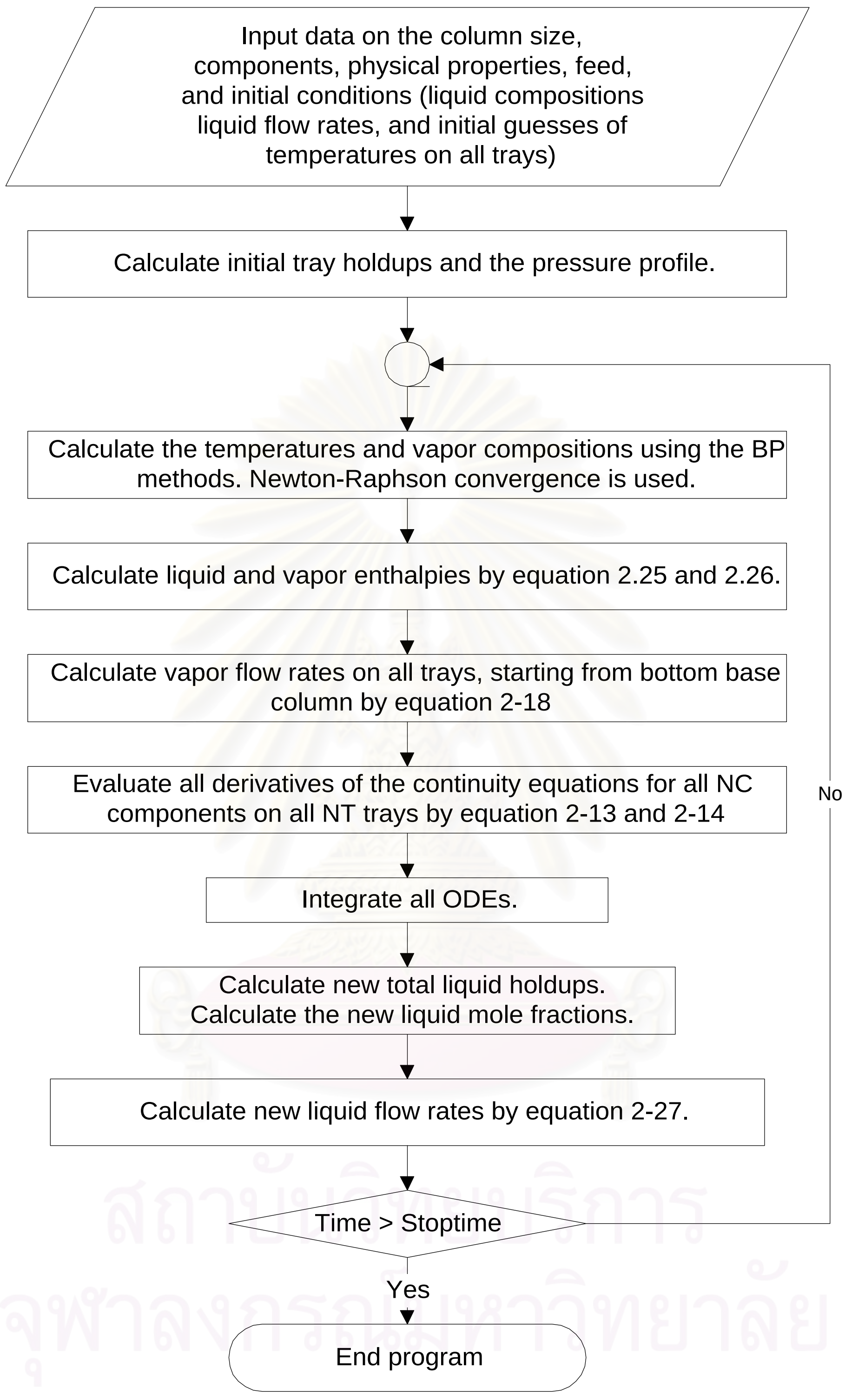

Input data on the column size, components, physical properties, feed, and initial conditions (liquid compositions liquid flow rates, and initial guesses of temperatures on all trays)
Calculate initial tray holdups and the pressure profile.
Calculate the temperatures and vapor compositions using the BP methods. Newton-Raphson convergence is used.
Calculate liquid and vapor enthalpies by equation 2.25 and 2.26.
Calculate vapor flow rates on all trays, starting from bottom base column by equation 2-18
Evaluate all derivatives of the continuity equations for all NC components on all NT trays by equation 2-13 and 2-14
Integrate all ODEs.
Calculate new total liquid holdups.
Calculate the new liquid mole fractions.
Calculate new liquid flow rates by equation 2-27.
Time > Stoptime
No
Yes
End program

Figure 3.2 Algorithm for solving dynamic multicomponent distillation simulation

3.3 Partition Algorithm for Distillation Model

The distillation column has NT trays, and contains NC components. Each tray has a set of equations as described in section 2.1.2. The Nth tray only connects with (N-1)th and (N+1)th tray. All sets of equations for all trays can be illustrated in the block band diagonal matrix (Figure 3.3).

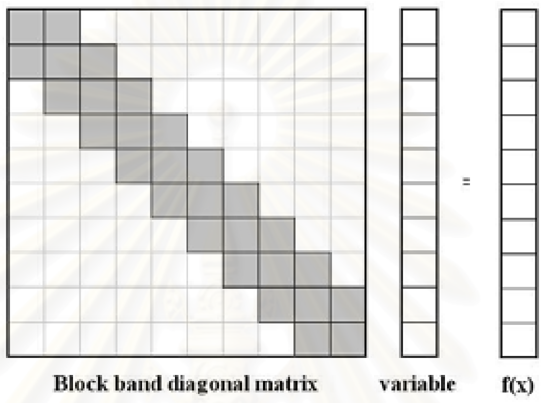


Figure 3.3 Block band diagonal matrix

In order to reduce memory usage, we reduce the block band diagonal matrix into spline matrix. According to the number of trays and that of processors, the tasks can be balanced among processors as shown in Figure 3.4.

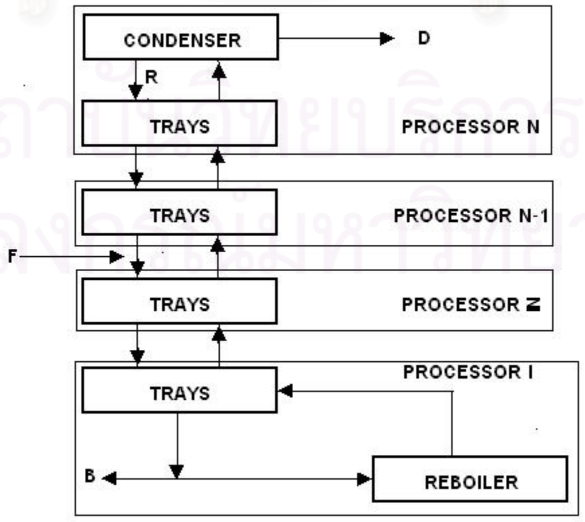


Figure 3.4 Task balance for NT trays on NP processors

The coordination algorithm is used for mapping on multicomputers. The node rank is an integer between 0 and n-1 for an MPI application consisting of n nodes. The basic structure of the coordination algorithm is as follows:

1. Calculate task size for each node. The task size is equal to the number of trays divided by the number of nodes, rounded down.

2. Calculate tray range for each node. First tray for each node is equal to task rank multiplied by task size. Last tray for each node is calculated from task rank plus one and multiplied by task size. If node rank is equal to zero, then include reboiler into task size. And if node rank is n-1, then the node will include the condenser into its task size and last tray is NT.

Each processor computes only a number of specified trays. For each time horizon, the data will be shared and exchanged among processors which represent subsystems. The solution obtained for the trays at the boundary for each subsystem will be exchanged with adjacent subsystem as shown in Figure 3.5.

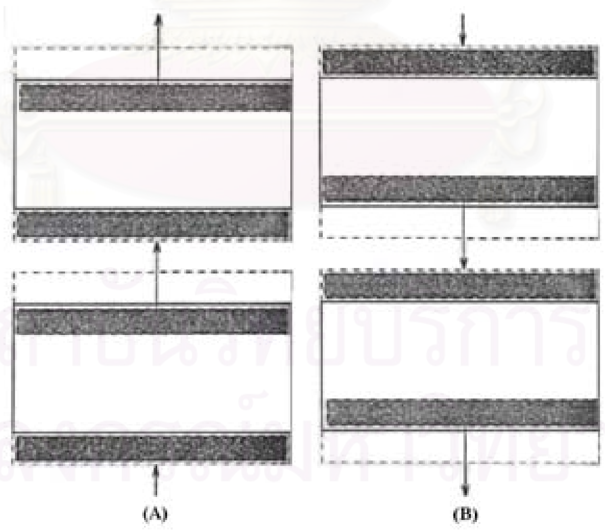


Figure 3.5 Two step processor transfer data. Ghost point areas are shown in dashed boxes; data to be moved are shaded.

(Source: William Gropp, Ewing Lusk and Anthony Skjellum, Using MPI Portable Parallel Programming with the message-Passing Interface, London: The MIT Press, 1994) [17]

In parallel computing, we reduces a number of communications by transfer all data at the end of each time horizon. For each subsystem, the serial calculation were carried out without waiting for the solution from other processors in each time horizon integration. It uses the old solution from last time horizon instead of the updated solution from the same time horizon.

The algorithm on message passing multicomputers using paradigm is shown in Figure 3.6. Each subsystem and the root do use the same code. However, the root also performs the output routine.

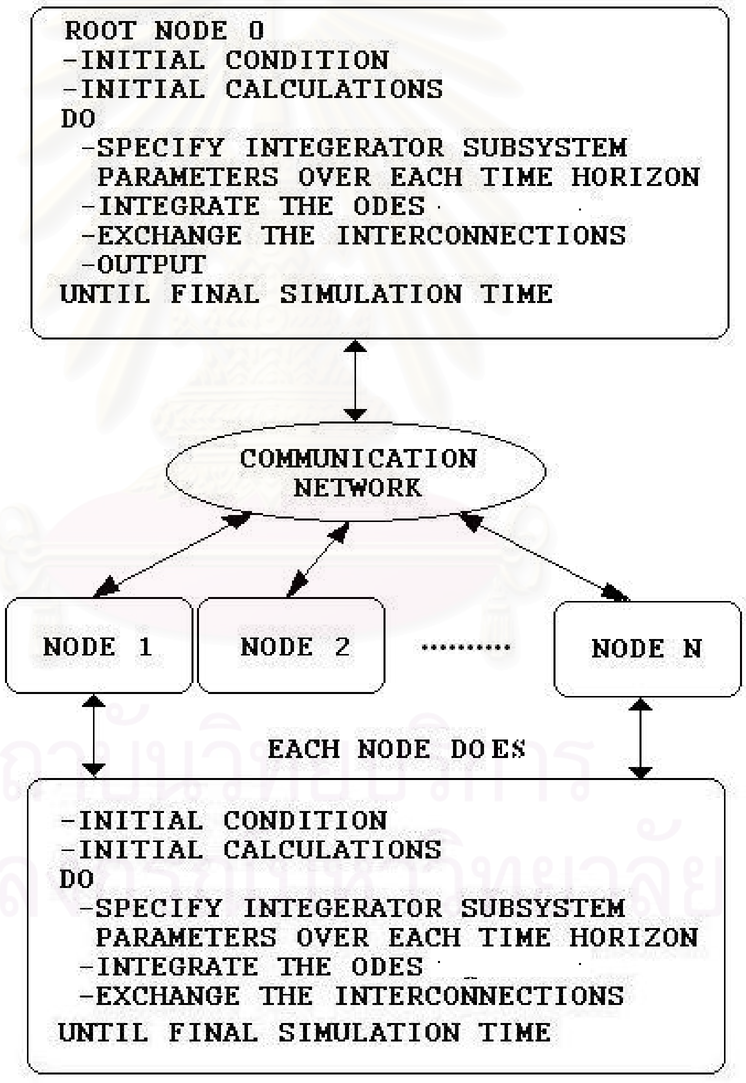

ROOT NODE 0
-INITIAL CONDITION
-INITIAL CALCULATIONS
DO
-SPECIFY INTEGERATOR SUBSYSTEM PARAMETERS OVER EACH TIME HORIZON
-INTEGRATE THE ODES
-EXCHANGE THE INTERCONNECTIONS
-OUTPUT
UNTIL FINAL SIMULATION TIME
COMMUNICATION NETWORK
NODE 1
NODE 2
....
NODE N
EACH NODE DOES
-INITIAL CONDITION
-INITIAL CALCULATIONS
DO
-SPECIFY INTEGERATOR SUBSYSTEM PARAMETERS OVER EACH TIME HORIZON
-INTEGRATE THE ODES
-EXCHANGE THE INTERCONNECTIONS
UNTIL FINAL SIMULATION TIME

Figure 3.6 Paradigm for parallel simulation

3.4 Input Data Required

3.4.1 Binary Distillation Column

1. Number of plates in the column (not including condenser and reboiler), NT
2. Feed tray, NF
3. Liquid holdup in column base and reflux drum (moles), MB and MD
4. Liquid holdup on each tray (moles), M
5. Product flow rate in column base and reflux rate (mol/h),V and R
6. Feed flow rate (lb mol/h), F
7. Relative volatility, α
8. Hydraulic time constant, β
9. Initial condition each tray for composition, X
10. Controller constant in column base and reflux drum, KCD, KCB, TAUD, and TAUB
11. Feed disturbance, Z
12. Time step size (Hours), DELTA

3.4.2 Multicomponent Distillation Column

1. Number of plates in the column (not including condenser and reboiler), NT
2. Number of components, NC
3. Murphree vapor-phase tray efficiency, EFF
4. Feed tray, NF
5. Time step size (Hours), DELTA
6. Time step show result (Hours), STEP
7. Stop time for end program (Hours), STOP
8. Liquid feed data i.e. flow rate (lb mol/h), FL, temperature (^{o}F), TFL, and composition, XF

9. Vapor feed data i.e. flow rate (lb mol/h), FV, temperature (^{o}F), TFV, and composition, YF
10. Weir height in stripping and rectifying section (inches), WHS and WHR
11. Weir length in stripping and rectifying section (inches), WLS and WLR
12. Column diameter in stripping and rectifying section (inches), DS and DR
13. Volumetric holdup in column base and reflux drum (ft^{3}), MVB and MVD
14. Pressure in column base and reflux drum (psia), PB and PD
15. Vapor and liquid distillate product flow rate (lb mol/h), DV and DL
16. Reboiler heat input (10^{6} Btu/h), QR
17. Reflux flow rate (lb mol/h), R
18. Component properties i.e. name, molecular weight (lb$_{m}$/lb mol), MW, density, (DENS), heat of vaporization at normal boiling point (Btu/lb$_{m}$^{o}F), HVAP, boiling point temperature (^{o}F), BPT, heat capacity of vapor (Btu/lb$_{m}$^{o}F), HCAPV, heat capacity of liquid (Btu/lb$_{m}$^{o}F), HCAPL, Antoine constant (or calculate Antoine constant from equation 2-20 and 2-21, and temperature and pressure find from Figure 2.5 and 2.6 at equilibrium stage)
19. Initial condition for each tray (i.e. temperature (^{o}F), T, liquid flow rate (lb mol/h), LO, and liquid composition, X)

3.5 Hardware System

- Three personal computers, CPU: AMD Duron 600 MHz, RAM: 128 MB., OS: Linux Slackware 7.1, Lan card: SMC 1211TX, Hard disk: 4 GB.
- Surecom 5 Port Ethernet Mini HUB
- Each unit is connected with 10 baseT wire and joint to an Ethernet hub as shown in Figure 3.7.

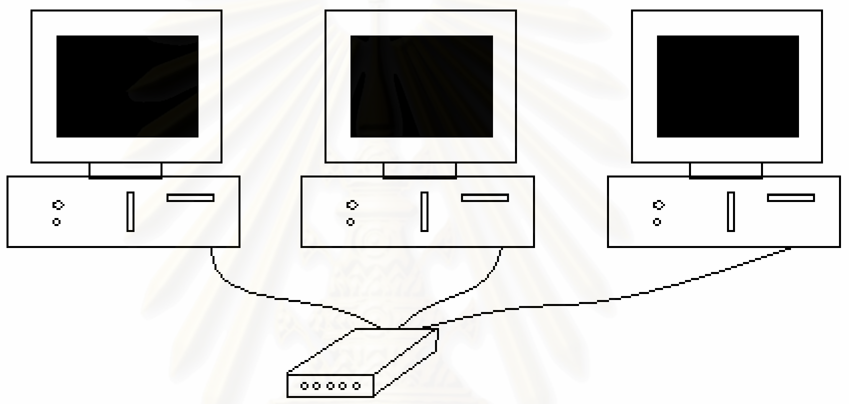


Figure 3.7 Multicomputers and network connection

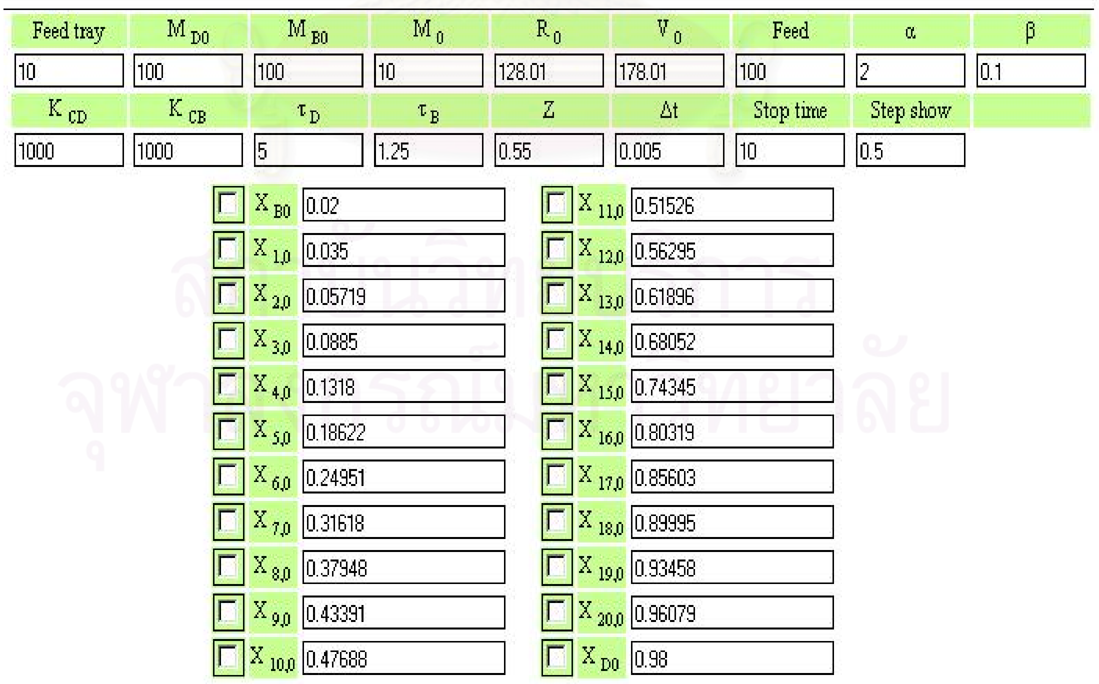
CHAPTER IV

SIMULATION RESULTS

The program developed by FORTRAN 77 language can be used to solve the dynamic distillation model. The performance test of simulation is shown and is discussed in this chapter. The parallel solutions were compared with the serial solution (William L. Luyben, 1992) [12]. Several study cases are tested.

4.1 Initial Condition

Case I: Dynamic binary distillation column (William L. Luyben, 1992) [12]. This case is the basic dynamic distillation column. The column is the conventional column (one feed and two products at distillate and bottom). The distillation column has 20 trays and 87 equations. The system has a step change in feed composition from 0.50 to 0.55 at time equal zero. The initial condition descriptions are shown in previous chapter. The Initial condition is shown as Figure 4.1.

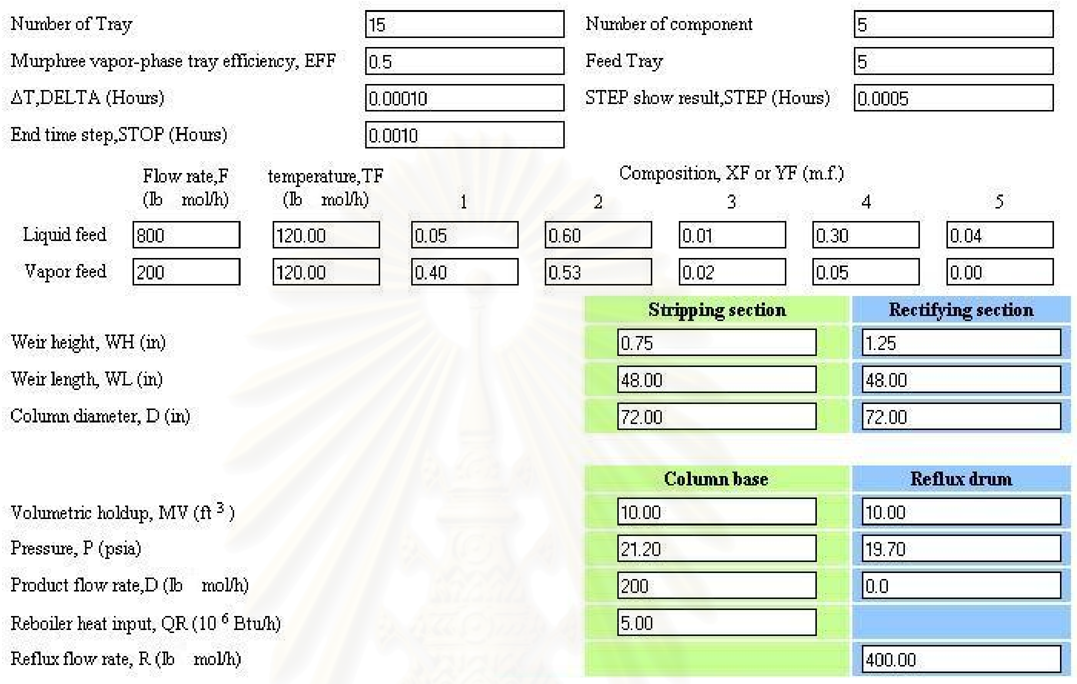


Feed tray	M_{D0}	M_{B0}	M_0	R_0	V_0	Feed	α	β
10	100	100	10	128.01	178.01	100	2	0.1
K_{CD}	K_{CB}	τ_D	τ_B	Z	Δt	Stop time	Step show	
1000	1000	5	1.25	0.55	0.005	10	0.5	

☐	X_{B0}	0.02	☐	$X_{11,0}$	0.51526
☐	$X_{1,0}$	0.035	☐	$X_{12,0}$	0.56295
☐	$X_{2,0}$	0.05719	☐	$X_{13,0}$	0.61896
☐	$X_{3,0}$	0.0885	☐	$X_{14,0}$	0.68052
☐	$X_{4,0}$	0.1318	☐	$X_{15,0}$	0.74345
☐	$X_{5,0}$	0.18622	☐	$X_{16,0}$	0.80319
☐	$X_{6,0}$	0.24951	☐	$X_{17,0}$	0.85603
☐	$X_{7,0}$	0.31618	☐	$X_{18,0}$	0.89995
☐	$X_{8,0}$	0.37948	☐	$X_{19,0}$	0.93458
☐	$X_{9,0}$	0.43391	☐	$X_{20,0}$	0.96079
☐	$X_{10,0}$	0.47688	☐	X_{D0}	0.98

Figure 4.1 Dynamic binary distillation initial condition

Case II: Dynamic multicomponent distillation column (William L. Luyben, 1992) [12]. The distillation column has 15 trays and 5 components and 221 equations. The initial condition descriptions are shown in previous chapter. The Initial condition is shown as Figure 4.2.



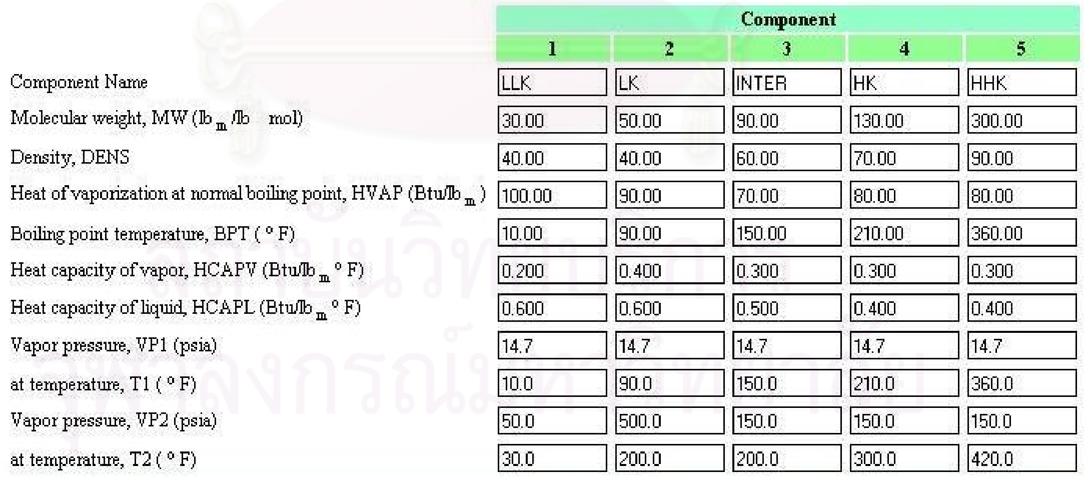
Number of Tray	15	Number of component	5
Murphree vapor-phase tray efficiency, EFF	0.5	Feed Tray	5
ΔT,DELTA (Hours)	0.00010	STEP show result,STEP (Hours)	0.0005
End time step,STOP (Hours)	0.0010		

	Flow rate,F (lb mol/h)	temperature,TF (lb mol/h)	Composition, XF or YF (m.f.) 1	2	3	4	5
Liquid feed	800	120.00	0.05	0.60	0.01	0.30	0.04
Vapor feed	200	120.00	0.40	0.53	0.02	0.05	0.00

	Stripping section	Rectifying section
Weir height, WH (in)	0.75	1.25
Weir length, WL (in)	48.00	48.00
Column diameter, D (in)	72.00	72.00

	Column base	Reflux drum
Volumetric holdup, MV (ft ^{3})	10.00	10.00
Pressure, P (psia)	21.20	19.70
Product flow rate,D (lb mol/h)	200	0.0
Reboiler heat input, QR (10 ^{6} Btu/h)	5.00	
Reflux flow rate, R (lb mol/h)		400.00

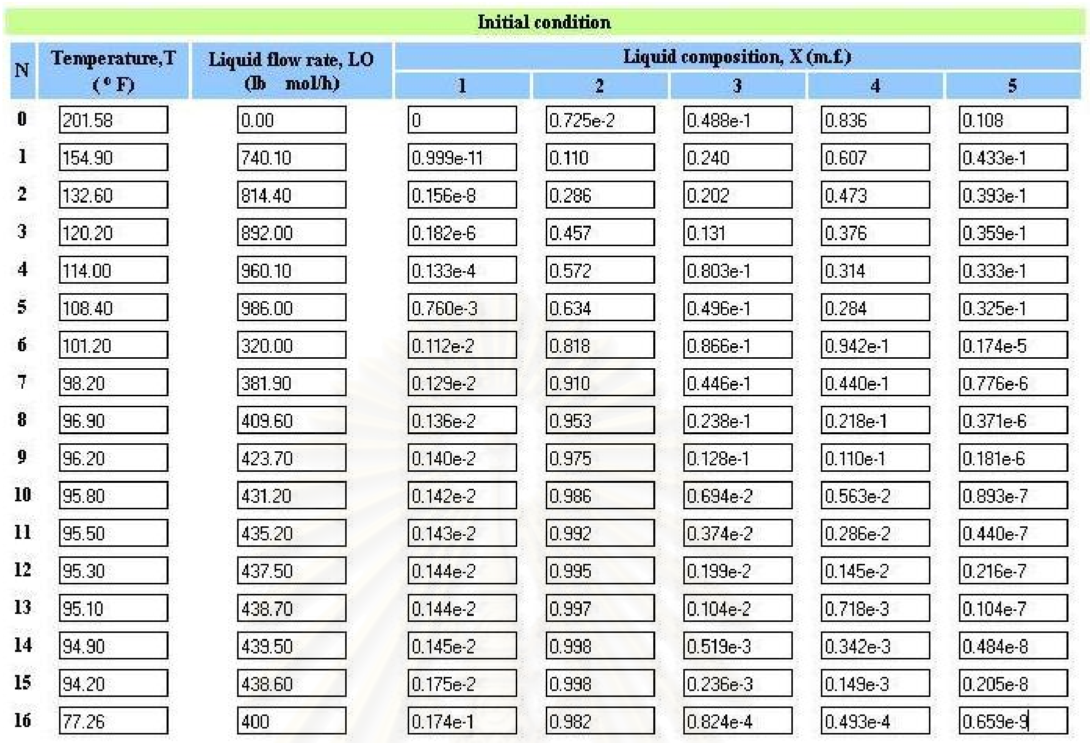
(a)



	Component 1	2	3	4	5
Component Name	LLK	LK	INTER	HK	HHK
Molecular weight, MW (lb $_m$ /lb mol)	30.00	50.00	90.00	130.00	300.00
Density, DENS	40.00	40.00	60.00	70.00	90.00
Heat of vaporization at normal boiling point, HVAP (Btu/lb $_m$)	100.00	90.00	70.00	80.00	80.00
Boiling point temperature, BPT ($^\circ$ F)	10.00	90.00	150.00	210.00	360.00
Heat capacity of vapor, HCAPV (Btu/lb $_m$ $^\circ$ F)	0.200	0.400	0.300	0.300	0.300
Heat capacity of liquid, HCAPL (Btu/lb $_m$ $^\circ$ F)	0.600	0.600	0.500	0.400	0.400
Vapor pressure, VP1 (psia)	14.7	14.7	14.7	14.7	14.7
at temperature, T1 ($^\circ$ F)	10.0	90.0	150.0	210.0	360.0
Vapor pressure, VP2 (psia)	50.0	500.0	150.0	150.0	150.0
at temperature, T2 ($^\circ$ F)	30.0	200.0	200.0	300.0	420.0

(b)

Figure 4.2 Dynamic multicomponent distillation initial condition (a) column properties (b) component properties (c) initial condition trays. Zeroth trays means reboiler. Last tray means condenser.



Initial condition							
N	Temperature,T (° F)	Liquid flow rate, LO (lb mol/h)	Liquid composition, X (m.f.)				
			1	2	3	4	5
0	201.58	0.00	0	0.725e-2	0.488e-1	0.836	0.108
1	154.90	740.10	0.999e-11	0.110	0.240	0.607	0.433e-1
2	132.60	814.40	0.156e-8	0.286	0.202	0.473	0.393e-1
3	120.20	892.00	0.182e-6	0.457	0.131	0.376	0.359e-1
4	114.00	960.10	0.133e-4	0.572	0.803e-1	0.314	0.333e-1
5	108.40	986.00	0.760e-3	0.634	0.496e-1	0.284	0.325e-1
6	101.20	320.00	0.112e-2	0.818	0.866e-1	0.942e-1	0.174e-5
7	98.20	381.90	0.129e-2	0.910	0.446e-1	0.440e-1	0.776e-6
8	96.90	409.60	0.136e-2	0.953	0.238e-1	0.218e-1	0.371e-6
9	96.20	423.70	0.140e-2	0.975	0.128e-1	0.110e-1	0.181e-6
10	95.80	431.20	0.142e-2	0.986	0.694e-2	0.563e-2	0.893e-7
11	95.50	435.20	0.143e-2	0.992	0.374e-2	0.286e-2	0.440e-7
12	95.30	437.50	0.144e-2	0.995	0.199e-2	0.145e-2	0.216e-7
13	95.10	438.70	0.144e-2	0.997	0.104e-2	0.718e-3	0.104e-7
14	94.90	439.50	0.145e-2	0.998	0.519e-3	0.342e-3	0.484e-8
15	94.20	438.60	0.175e-2	0.998	0.236e-3	0.149e-3	0.205e-8
16	77.26	400	0.174e-1	0.982	0.824e-4	0.493e-4	0.659e-9

(c)

Figure 4.2 (Continued)

Case III: Increase the number of trays and the size of data transfer for output between subsystems. The initial conditions are similar to the previous case except the number of trays, the feed tray, and the tray conditions. The new initial conditions were estimated by using linear interpolation between the upper tray and lower tray conditions.

$$f(x) = f(x_0) + (x - x_0)\frac{f(x_1) - f(x_0)}{x_1 - x_0} \qquad (4\text{-}1)$$

Where x is the tray number. f is the tray condition. The subscript zero means lower tray. The subscript one means upper tray. After linear interpolation, the tray number with decimal point was round up to integer number. The feed tray was changed with the total number of tray in the column as shown in Table 4.1. The distillation columns have the number of equation as shown in Table 4.2.

Table 4.1 The relationship between the number of trays and the feed tray.

The number of trays	NT	15	60	120	240	360	480
The feed tray	$NF = (5-1)\left(\frac{NT}{15}\right)+1$	5	17	33	65	97	129

Table 4.2 The relationship between the number of trays and the number of equations.

The number of trays	NT	15	60	120	240	360	480
The number of equations	$NE = (NT+2)(2NC+3)$	221	806	1,586	3,146	4,706	6,266

These case studies were divided into two parts. First, the performance was studied when full solution transfer for output at $\Delta t = 10^{-4}$ time step and integrated time horizon from 0.0 hours to 2x10^{-4} hours. Second part studies the performance when partial solution transfer for output at $\Delta t = 10^{-4}$ time step and integrated time horizon from 0.0 hours to 0.02 hours.

The load balance is measured as

$$\text{imbalance} = \frac{\text{CPU}_{\max} - \text{CPU}_{\text{avg}}}{\text{CPU}_{\text{avg}}} \tag{4-2}$$

where $\text{CPU}_{\max}$ and CPU_{avg} refer to the maximum and average CPU times. In general, a perfectly load balanced partitioning will produce an imbalance of zero. In this case study, load imbalance is nearly zero.

4.2 Results

Case I: Dynamic binary distillation

Examine the accuracy of the solutions obtained from serial and parallel computation. The parallel computation was carried out with two processors configuration test unit. The results are shown in Table 4.3.

Table 4.3 Comparison of compositions from serial and parallel computation for dynamic binary distillation column

TIME (hours)	Serial			Parallel nodes=2		
	XB	X10	XD	XB	X10	XD
0.0	0.02000	0.47688	0.98000	0.02000	0.47688	0.98000
5.0	0.01950	0.52057	0.98018	0.01951	0.52061	0.98019
10.0	0.01989	0.52266	0.98024	0.01989	0.52268	0.98024
15.0	0.01997	0.52299	0.98017	0.01997	0.52300	0.98017
20.0	0.01998	0.52299	0.98011	0.01998	0.52299	0.98011
25.0	0.01999	0.52295	0.98007	0.01999	0.52296	0.98007
30.0	0.01999	0.52293	0.98004	0.01999	0.52293	0.98004
35.0	0.02000	0.52291	0.98003	0.02000	0.52291	0.98003
40.0	0.02000	0.5229	0.98002	0.02000	0.52290	0.98002
45.0	0.02000	0.52289	0.98001	0.02000	0.52289	0.98001
50.0	0.02000	0.52289	0.98001	0.02000	0.52289	0.98001

At the beginning of the simulation, the solutions by parallel calculation have slightly error. However, the solutions reach steady state at the same time and same solution. Figure 4.3 show the percentage error between serial and parallel simulation of various variables (i.e. composition at bottom, 10th tray, distillate, and control parameters). The error is monotonically decreased with time.

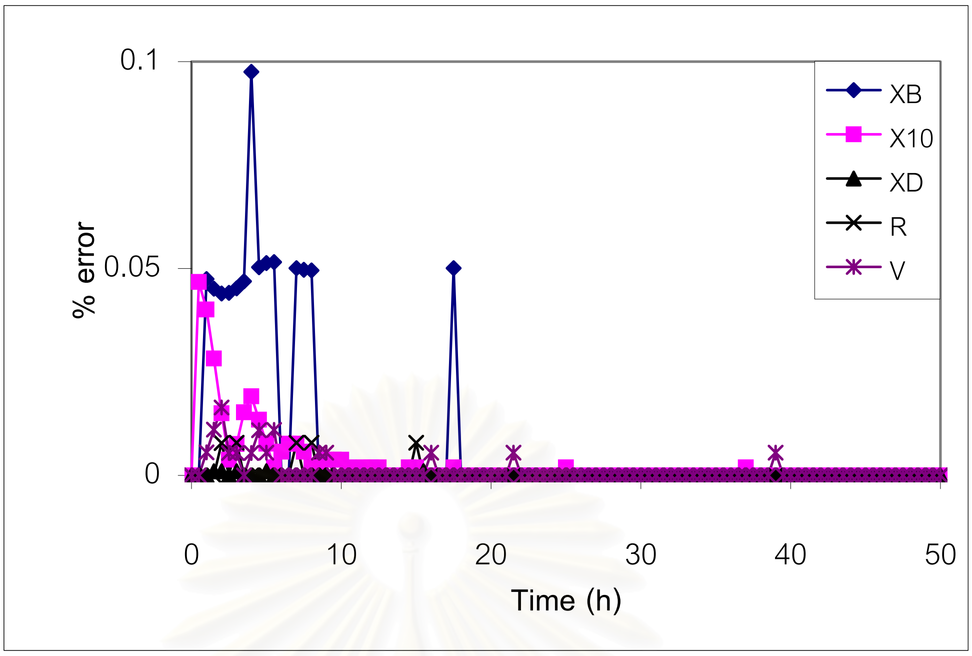


Figure 4.3 Percentage error between the serial and parallel simulation two processors in dynamic binary distillation

During the experiment, it was observed that the average CPU time with parallel simulation dramatically increases compared with serial simulation, but the speedup and efficiency decreases. Table 4.4 shows the performances of serial and parallel simulation.

Table 4.4 Comparison the performance between serial and parallel simulation (2 nodes) of dynamic binary distillation.

The performance	Serial	Parallel (nodes=2)
Average CPU time (seconds)	0.20	26.15
Speedup	1.00	7.64E-03
Parallel efficiency	1.00	3.82E-03

Case II: Dynamic simulation of multicomponent distillation column

Examine the accuracy of the solution of dynamic multicomponent distillation model by comparing dynamic response of distillate and bottom products obtained from serial and parallel simulation (two processors and three processors). The results were shown in Figure 4.4 to Figure 4.9. It was observed that the profiles of both distillate and bottom products obtained from parallel calculation were different from that obtained from serial calculation. Moreover, the system of three nodes has shown larger error than that of two nodes, especially the distillate temperature and composition.

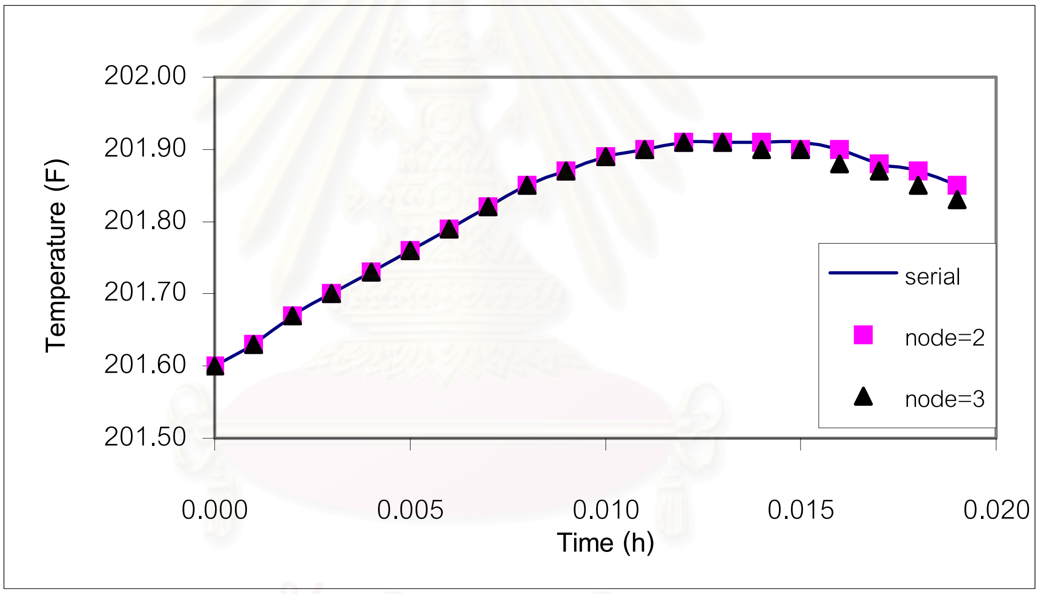


Figure 4.4 Transient profiles of bottom product temperature on the 15-trays column

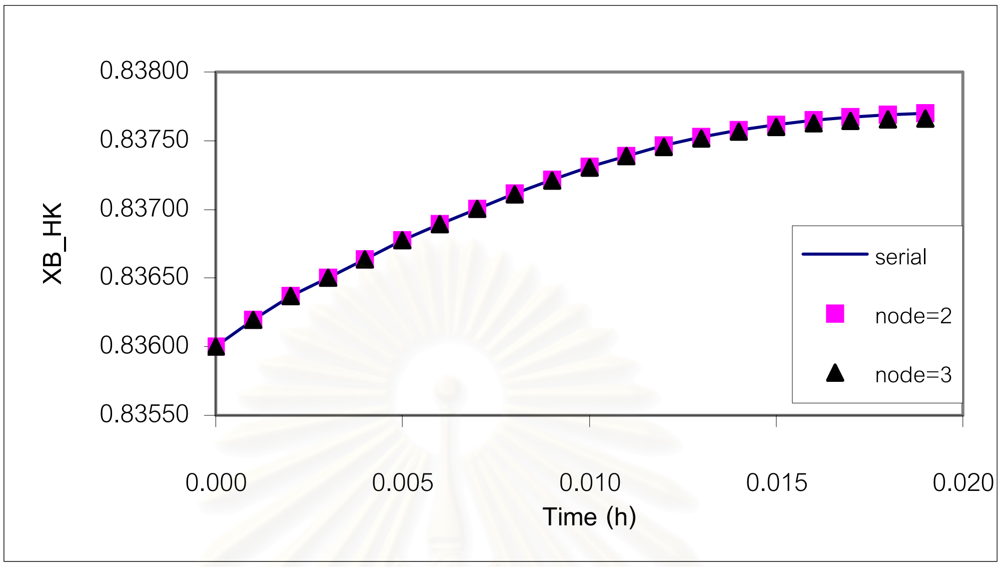


Figure 4.5 Transient profiles of liquid composition of the heavy-key composition in the bottom product on the 15-trays column

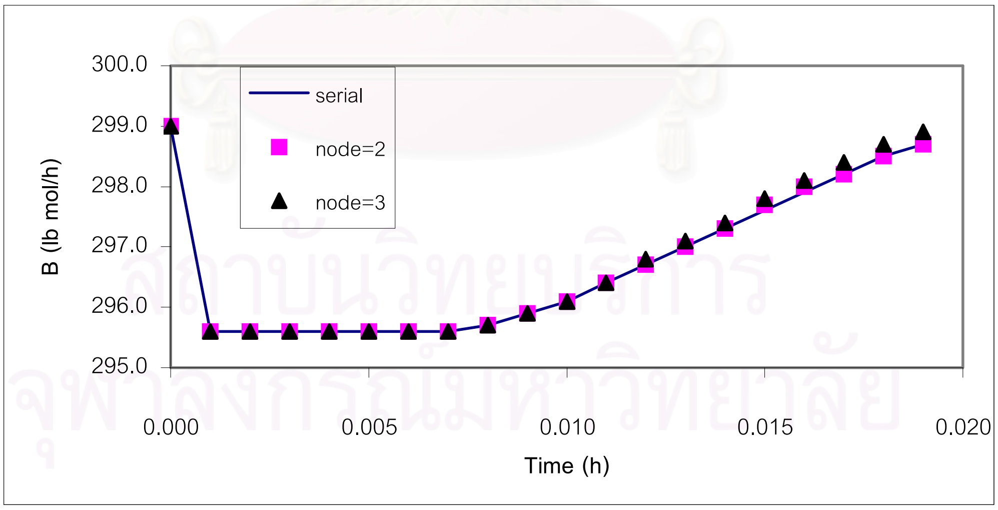


Figure 4.6 Transient profiles of bottom product rate on the 15-trays column

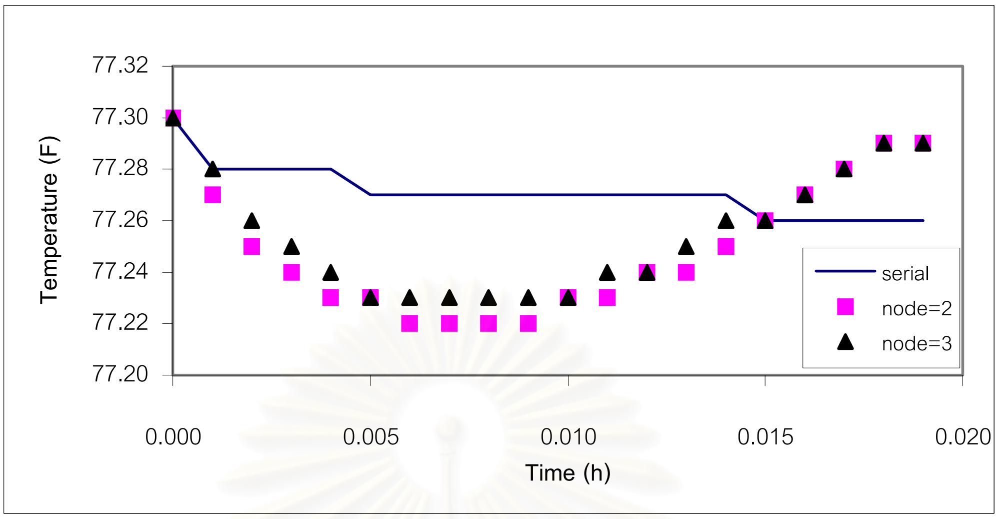


Figure 4.7 Transient profiles of distillate product temperature on the 15-trays column

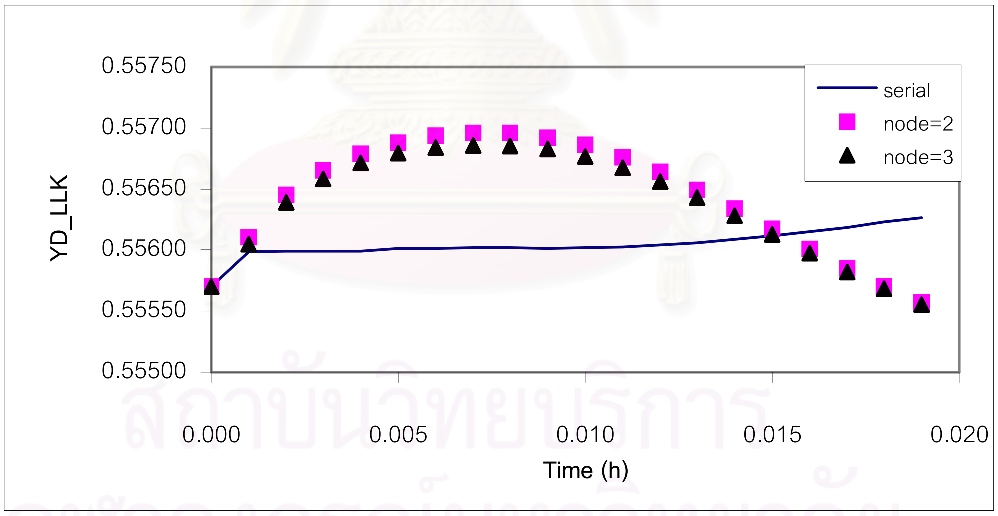


Figure 4.8 Transient profiles of vapor composition of the light-key composition in the distillate product on the 15-trays column

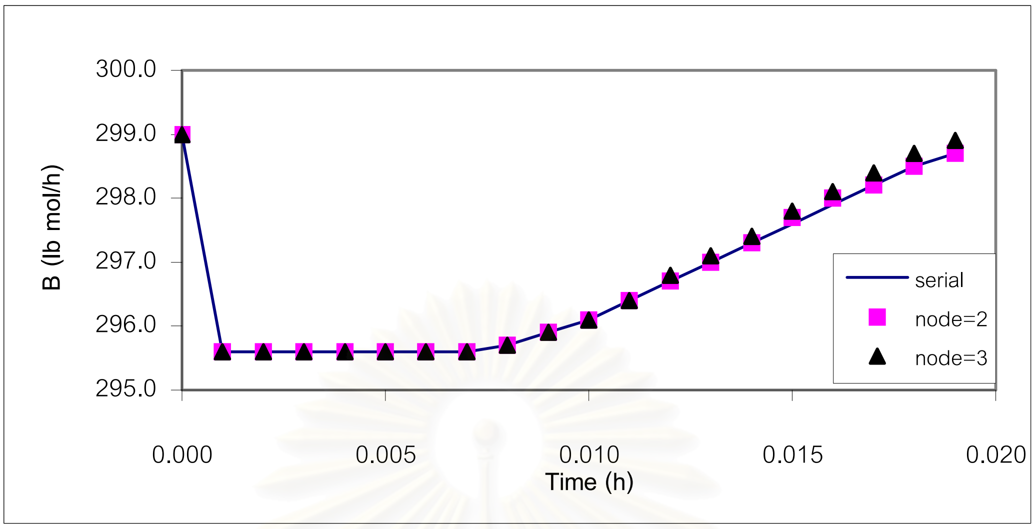


Figure 4.9 Transient profiles of distillate product rate on the 15-trays column

The average CPU time in parallel simulation is higher than the average CPU time in serial simulation. In other words, the former has lower speedup and lower efficiency. The load imbalance is nearly zero. Table 4.5 shows the performances between serial and parallel simulation. The parallel CPU time is larger than the serial CPU time, because the computational time is smaller than communication time.

Table 4.5 Comparison the performance between a serial simulation and a parallel simulation (two and three processors) on 15 trays and 5 components distillation.

The performance	Serial	Parallel (nodes=2)	Parallel (nodes=3)
Average CPU time (seconds)	0.04405	0.14842	0.24374
Speedup	1.00000	0.29679	0.18073
Parallel efficiency	1.00000	0.14840	0.06024
Load imbalance		0.00904	0.01100

Case III: Test performance

Examine the performance of the multicomponent distillation simulation.

Part I, The system is conducted in full data transfer for output. All solutions for monitoring were sent back to root processor. The large numbers of data increase as the numbers of tray increase. Comparison the CPU time between serial and parallel simulation was shown in Figure 4.10. With lower number of trays, the parallel computing time is larger than the serial one. With higher number of trays, serial computing time becomes longer than the two-nodes parallel time, but not for three-nodes parallel time. Comparison the speedup of parallel simulation was shown in Figure 4.11. Two nodes speedup is more than one when the number of trays is larger than one hundred. On the other hand, the system with three-nodes, its speedup is less then one. Comparison the parallel efficiency was shown in Figure 4.12. Parallel efficiencies for parallel computing systems are much less than one. Two-nodes system has higher efficiency than the three-nodes system.

Part II, The system is the partial data transfer for output. Only distillate and bottom solutions were sent back to root processor. The numbers of data are not increased as the number of trays increased. Comparison the CPU time between serial and parallel simulation was shown in Figure 4.13. With lower number of trays, serial time is faster than parallel time. On the other hand, with higher number of trays, parallel time is faster than serial time. The system with three-nodes, its parallel time is faster than that with two-nodes. Comparison the speedup of parallel simulation was shown in Figure 4.14. The speedup of the two-nodes system is more than one when the number of trays in the column are higher than eighty. While that of the three-nodes system is more than one, when the number of trays in the column are higher than one hundred. However, the larger the number of trays, the three-nodes system becomes more powerful than the two-nodes system. Comparison of the parallel efficiencies were shown in Figure 4.15. Parallel efficiencies were less than one for both of parallel units. Two-nodes system is more efficient than the three-nodes system.

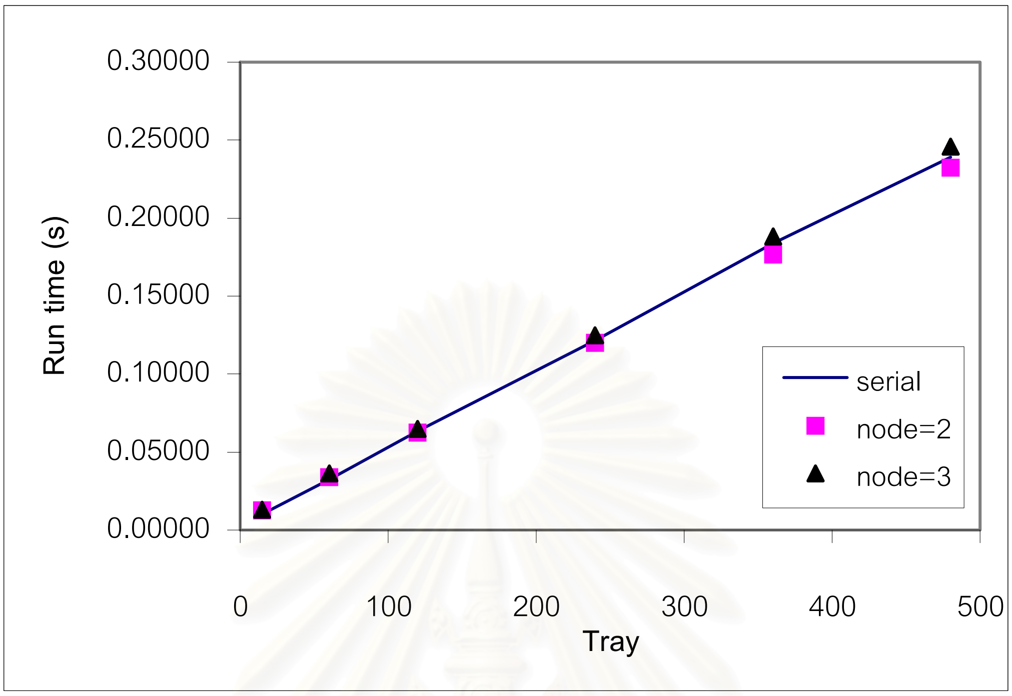


Figure 4.10 CPU time of full data transfer for output.

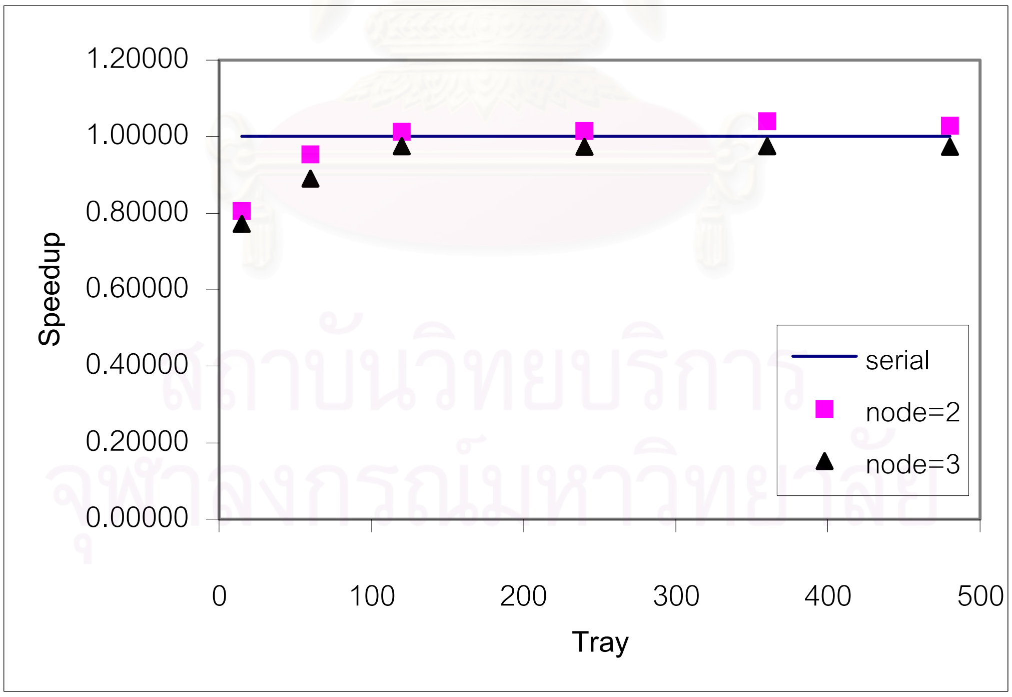


Figure 4.11 Speedup of full data transfer for output.

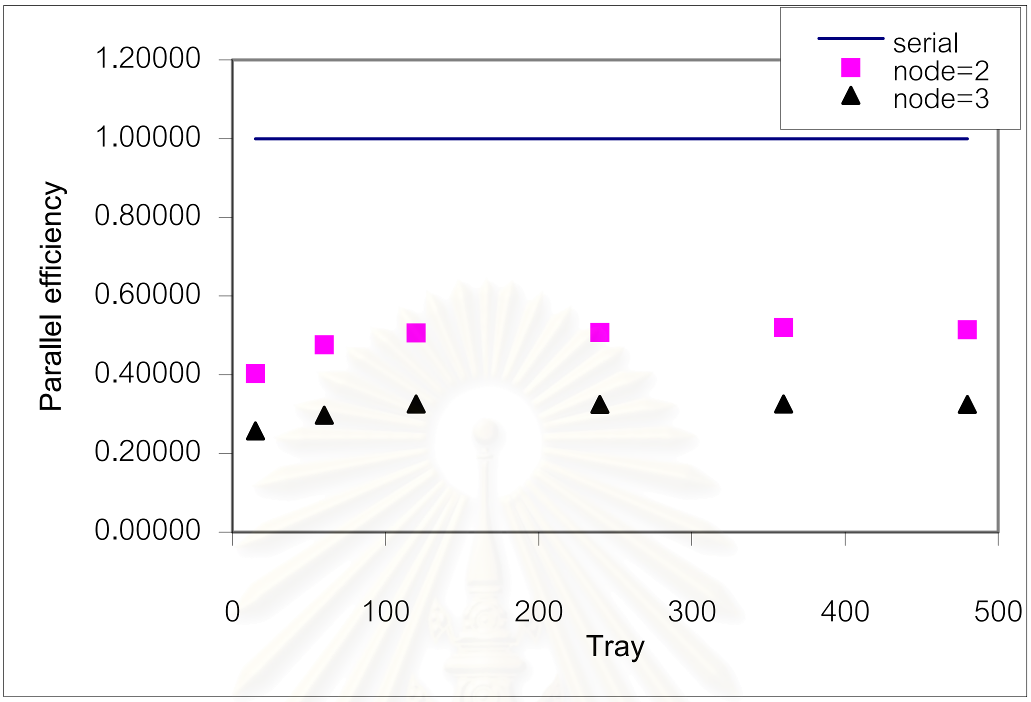


Figure 4.12 Parallel efficiency of full data transfer for output.

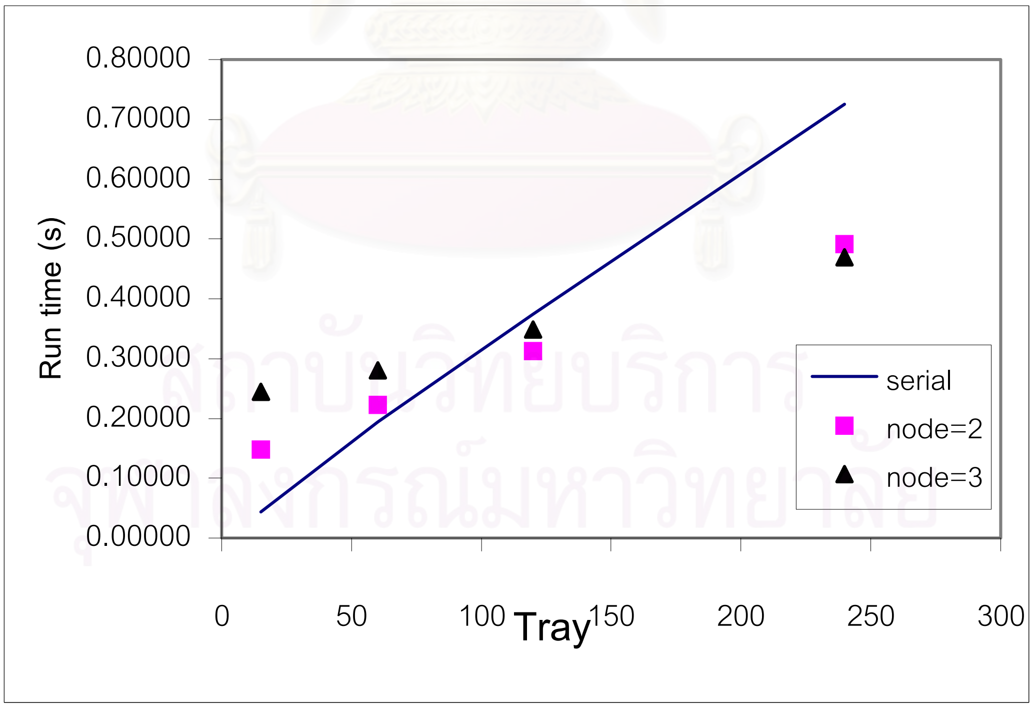


Figure 4.13 CPU time of partial data transfer for output.

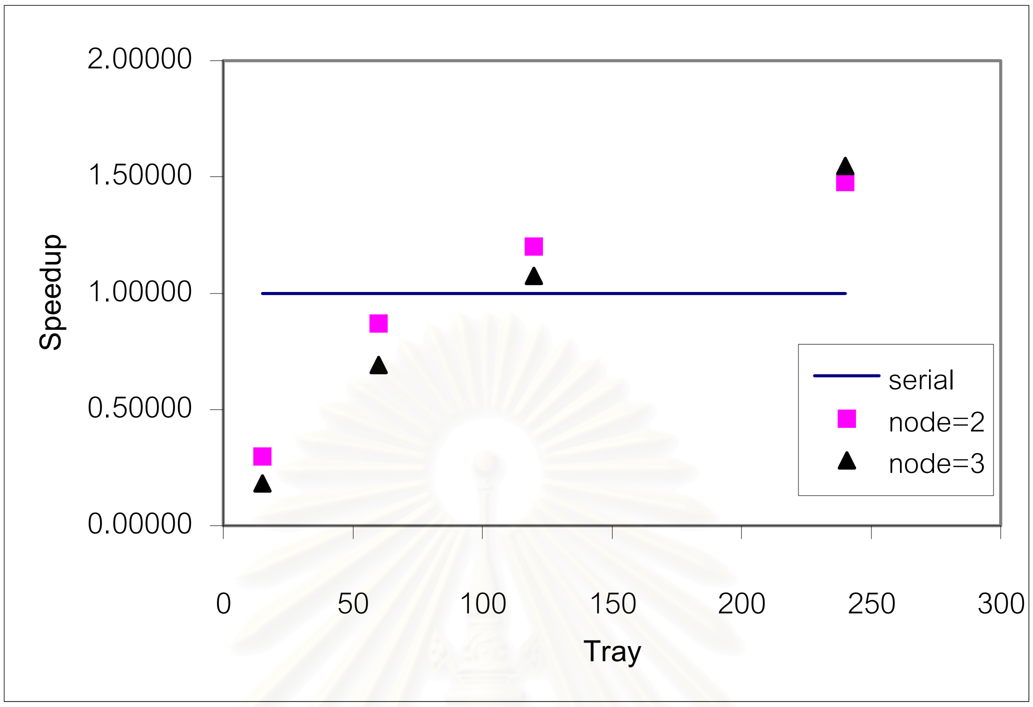


Figure 4.14 Speedup of partial data transfer for output.

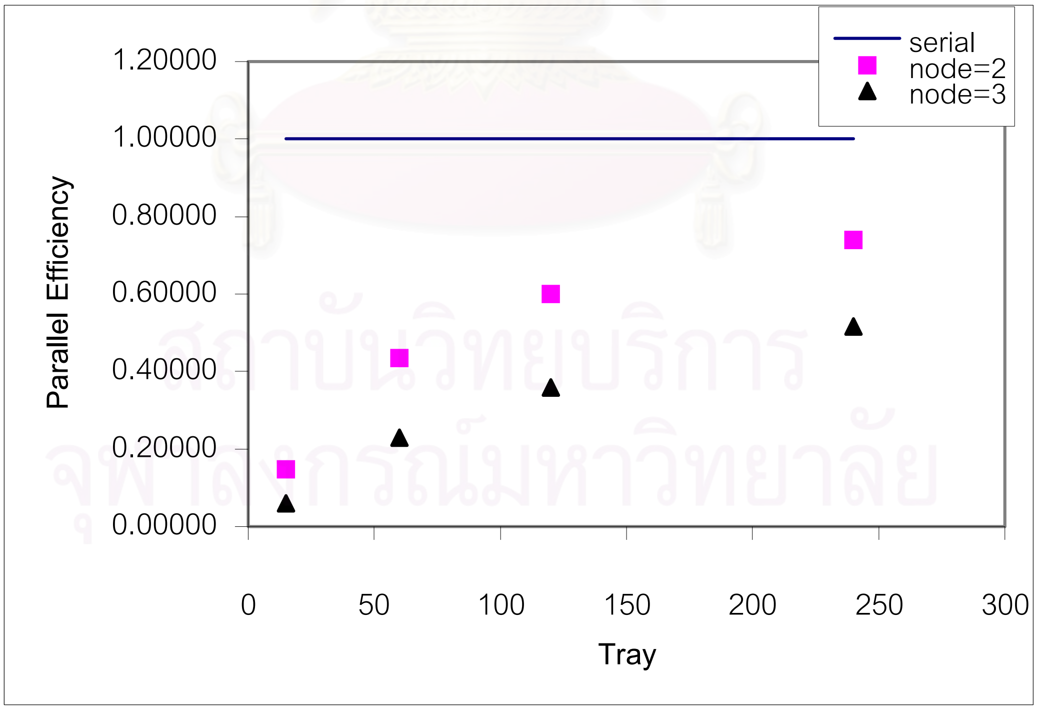


Figure 4.15 Parallel efficiency of partial data transfer for output.

Table 4.6 shows the load imbalance. This algorithm gives good load balance.

Table 4.6 CPU time and load balance for the parallel runs on different numbers of trays.

Trays	Nodes	CPU_{max}	CPU_{avg}	$CPU_{max} - CPU_{avg}$	Load imbalance
15	2	0.149757	0.148415	0.001342	0.009042
	3	0.246415	0.243735	0.002680	0.010996
60	2	0.223933	0.222570	0.001363	0.006126
	3	0.283441	0.280753	0.002688	0.009575
120	2	0.31376	0.312443	0.001317	0.004215
	3	0.351174	0.348916	0.002258	0.006472
240	2	0.492588	0.491218	0.001370	0.002789
	3	0.472761	0.468885	0.003876	0.008266

Examine the accuracy of the solution of dynamic multicomponent distillation model, comparison of the percentage error of parallel dynamic responses of distillate and bottom products between two processors and three processors was shown in Figure 4.16 to Figure 4.21. It was observed that the error profiles of distillate product was higher than those of bottom product. When the number of trays is large, the two-nodes and three-nodes systems had the same trend of the error profiles. The errors on the bottom products were within the acceptable range. However, the errors on the distillate product were rather high, especially when the column containing the large number of trays. The reason of the large deviation was that the data was not updated often enough and the process was large. Therefore, the solution can move away much Further.

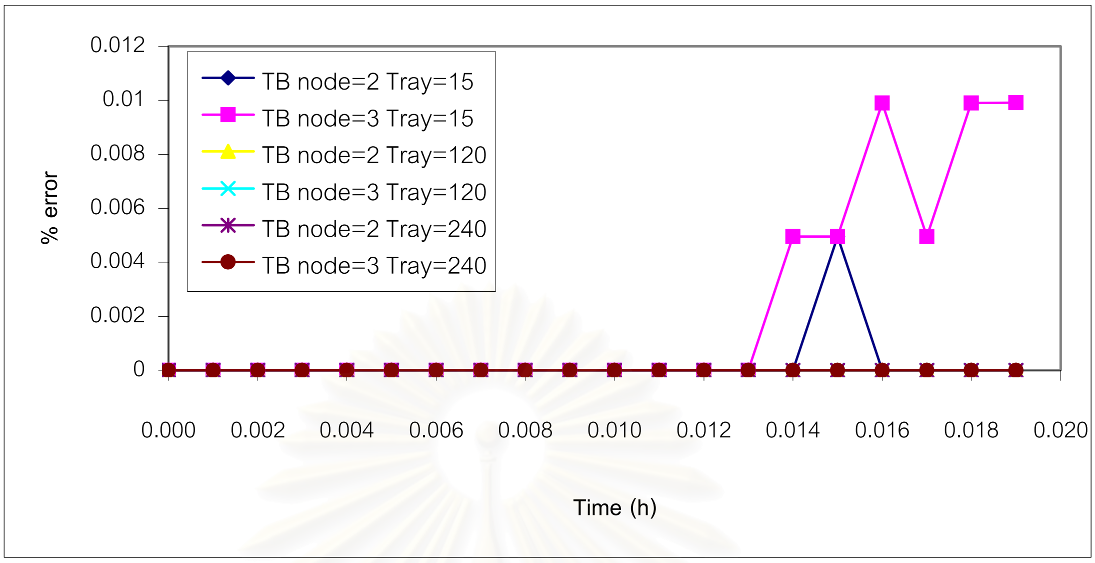


Figure 4.16 Percentage error of transient profiles of bottom product temperature in multicomponent distillation

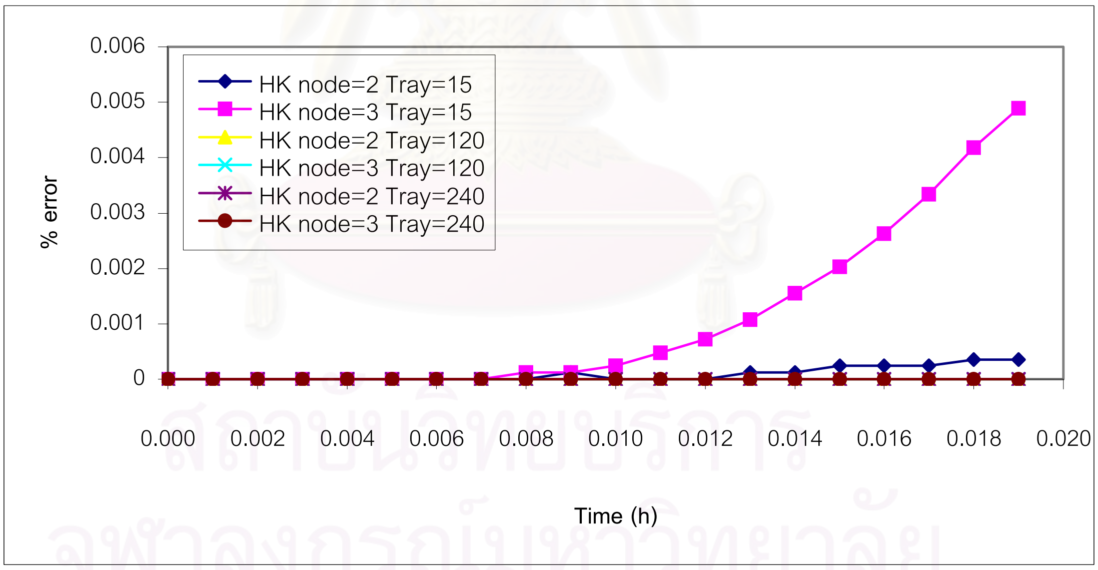


Figure 4.17 Percentage error of transient profiles of liquid composition of the heavy-key composition in the bottom product in multicomponent distillation

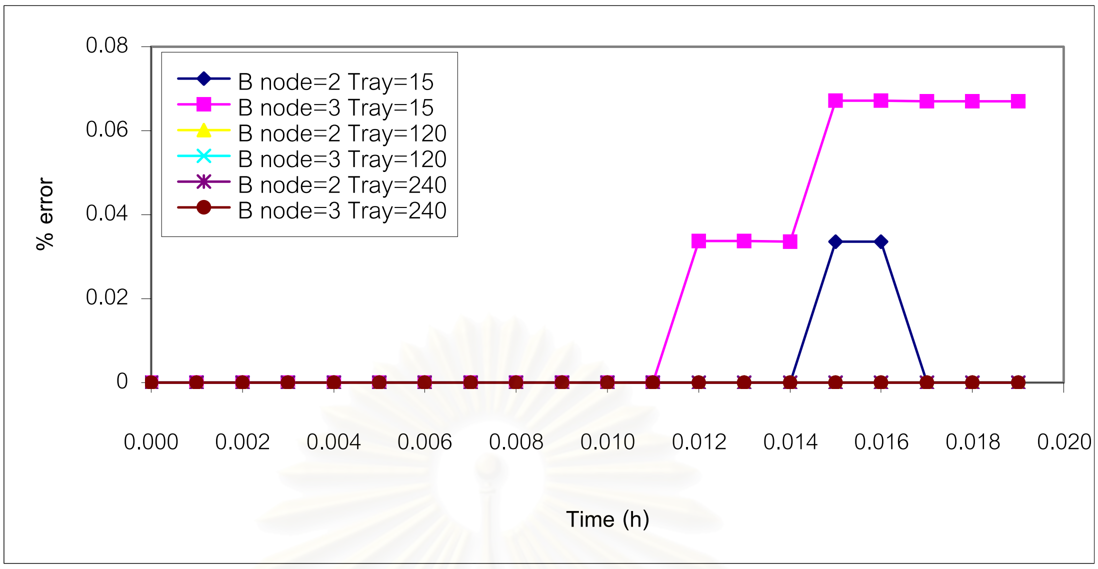


Figure 4.18 Percentage error of transient profiles of bottom product rate in multicomponent distillation

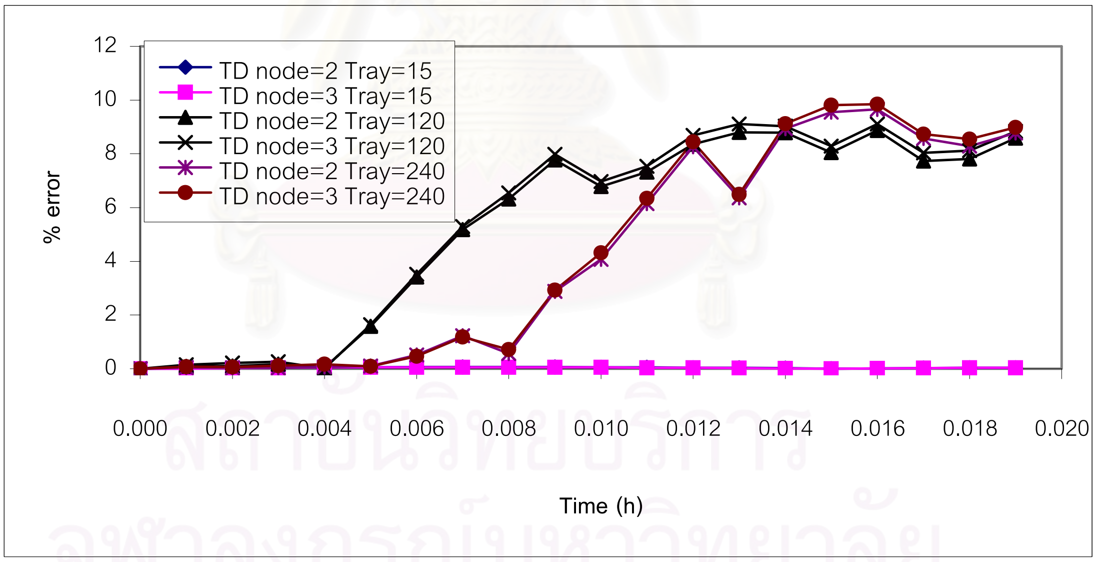


Figure 4.19 Percentage error of transient profiles of distillate product temperature in multicomponent distillation

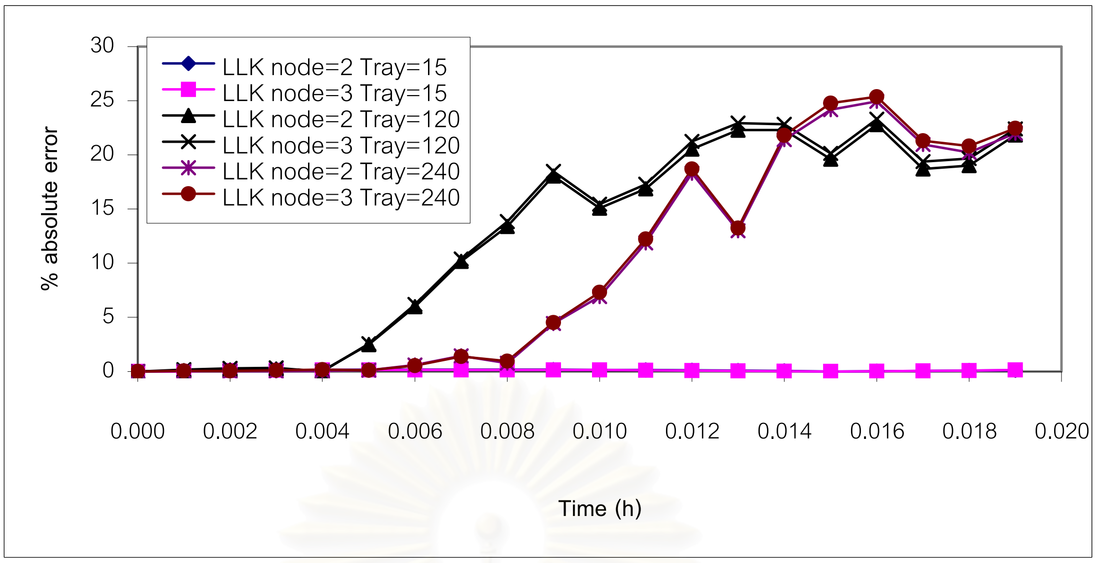


Figure 4.20 Percentage error of transient profiles vapor composition of the light-key composition in the distillate product in multicomponent distillation

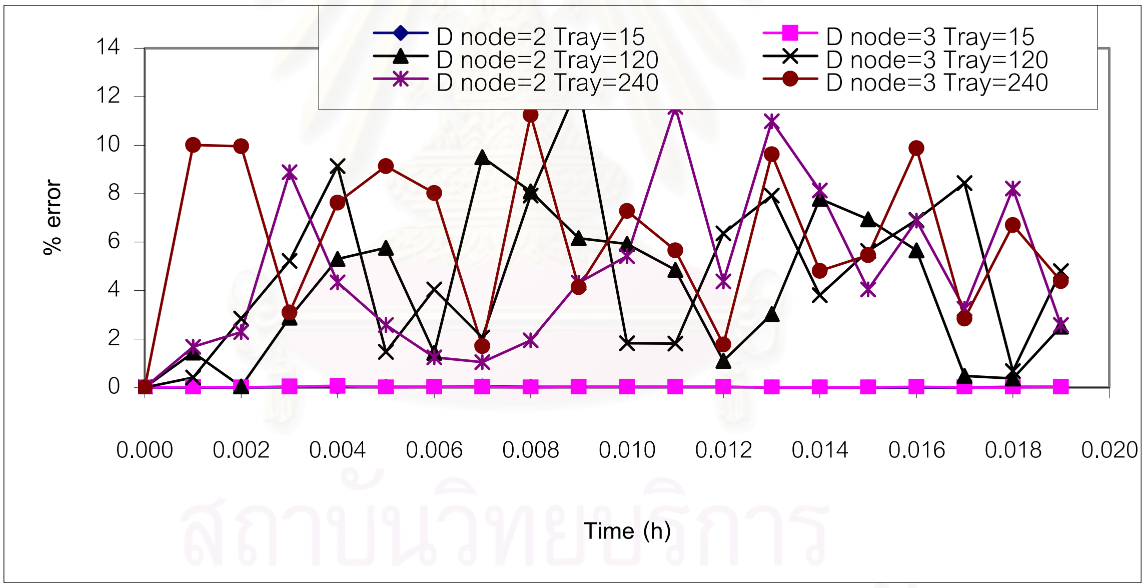


Figure 4.21 Percentage error of transient profiles of bottom product rate in multicomponent distillation

Examine the runtime of the dynamic multicomponent distillation model, comparison of the full data transfer for output on two processors between computation time and communication time was shown in Figure 4.22.

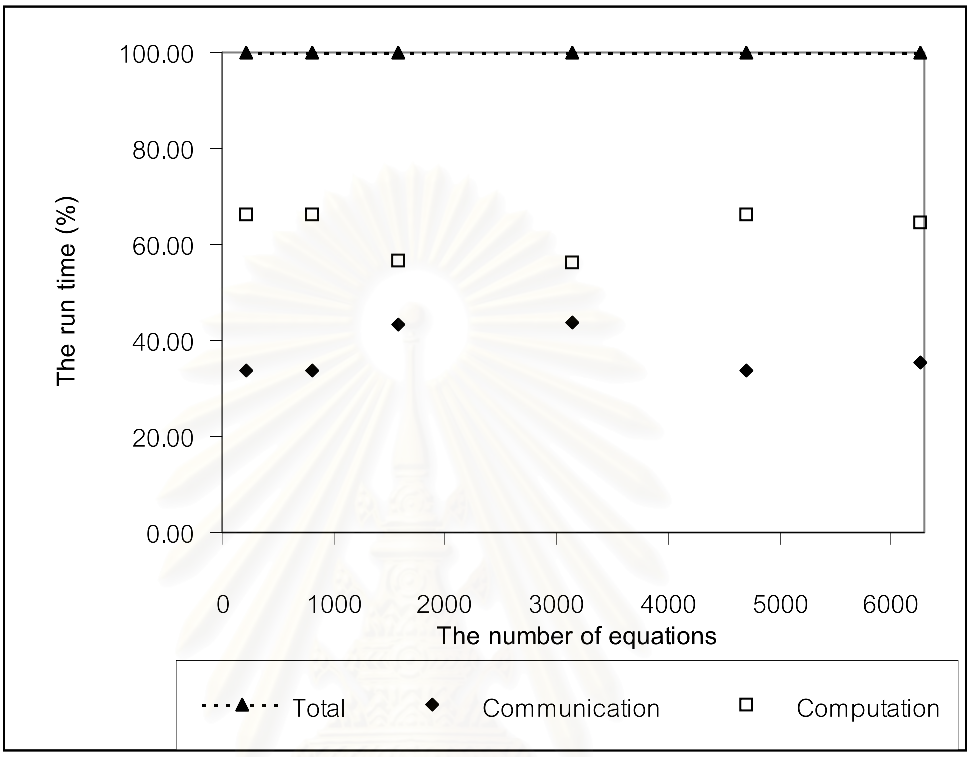


Figure 4.22 Comparison the runtime of the full data transfer for output on two processors.

CHAPTER V

CONCLUSIONS

5.1 Dynamic Binary Distillation

In case I, the dynamic binary distillation column model on distributed computing consumes more CPU time. It indicates that task was not large enough to employ parallel computation. The tasks were increased by adding the number of components. It was observed that more error was generated in the beginning of the computation compared with serial calculation. However, both approaches converge to the same solution at the same time. The partition algorithm has minimum load imbalance. The overhead has affected on CPU time higher than on computation time. It indicates that the size of the problem is smaller than the parallel computation.

5.2 Dynamic Multicomponent Distillation

In case II, the parallel unit run for dynamic multicomponent distillation gives better performance than that for binary system, but task was still not large enough when the column having a small number of trays. In the multicomponent system, a new algorithm was used to find the solution. The simulation was more complicated than for the binary system. In this case study, the distillate solution was more deviate from the serial solution than the bottom product. The error also increases as the number of trays is increased. The parallel performance was better than the binary system. The partition algorithm gives good load balance. The small speedup indicates that the size of the problem was smaller than parallel computing.

In case III, the problem was divided in two cases: the size of the problem and the size of the transfer data for output. The large size of the problem has good performance than the small size of problem. The error also increases as the size of equation is increased. The small size of the solution transfer for output show better performance than the large size of the solution transfer for output. The speedup of maximum solution transfer for output gives performance nearly one. It indicated that, for the case of full data transfer for output, the serial simulation is the appropriate approach.

These algorithms were not adequate, according to parallel efficiency since, for all case studies, parallel efficiencies were below than one. The partition algorithm has good load balance. The algorithm of Murphree tray efficiency of multicomponent distillation was modified for parallel calculation. It decreased CPU time but gives accumulation error instead.

5.3 Suggestion and Future Development

This multicomponent distillation model was used the ideal equation of state. In the future, the nonideal equation of state (e.g. Generic – Redlich – Kwong (GRK), Soave – Redliche – Kwong (SRK), Peng – Robinson (PR) and others) may be applied. These nonideal equations of states require more computation since the equations are complicated. The multicomponent distillation model is open-loop system. Reflux rate and heat input to the reboiler are the manipulated variables. In this simulation, they are simply held constant. If the close-loop response is desired, the model can be changed to use reflux rate to hold a temperature or a composition in the top of the column and to use heat input to hold a temperature or a composition in the bottom of the column. There are two degrees of freedom, so two variables can be specified. This work used the BP method solving MESH equations. There are many algorithms to solve MESH equations. Instead of using Euler integration, Runge-Kutta integration can be used.

REFERENCES

1. Phil Merkey, Beowulf Introduction & Overview [Online], 1998, Available from: http://dune.mcs.kent.edu/~farrell/equip/beowolf/intro.html [2001, December 17]

2. Secchi A.R ., Morari M., and Biscaia E.C., THE WAVE-FORM RELAXATION METHOD IN THE CONCURRENT DYNAMIC PROCESS SIMULATION, Computers & Chemical Engineering Vol. 17, No. 7, July 1993, pp. 683-704.

3. Mallya J.U., Zitney S.E., Choudhary S., and Stadtherr M.A., Parallel frontal solver for large-scale process simulation and optimization, AICHE JOURNAL Vol. 43, No. 4, April 1997, pp. 1032-1040.

4. Nabil A.J., Brice C., and Costas K., A multirate parallel-modular algorithm for dynamic process simulation using distributed memory multicomputers, Computers and Chemical Engineering Vol. 23, 1999, pp. 733-761.

5. Camarda K.V., and Stadtherr M.A., Matrix ordering strategies for process engineering: graph partitioning algorithms for parallel computation, Computers & Chemical Engineering Vol. 23, No. 8, August 1999, pp. 1063-1073.

6. Borchardt J., Newton-Type decomposition methods in large-scale dynamic process simulation, Computers & Chemical Engineering Vol. 25, No. 7-8, August 2001, pp. 951-961.

7. Marakis J.G., Chamico J. Brenner G., and Durst F., Parallel ray tracing radiative heat transfer – Application in a distributed computing environment, International Journal of Numerical Methods for Heat & Fluid Flow Vol. 11, No. 7, 2001, pp. 663-681.

8. Carlsson P., Distributed optimization with a two-dimensional drying model of a board, built up by sapwood and heartwood, HOLZFORSCHUNG Vol. 55, No. 4, 2001, pp. 426-432.

9. Anido L., Santos J., Caeiro M. Rodriguez J., Femandez M.J., and Llamas M., Moving the business logic tier to the client. Cost-effective distributed computing for the WWW, Software-Practice & Experience Vol. 31, No. 14, November 2001, pp. 1331-1350.

10. Lalis S., and Karipidis A., JaWS: An open market-based framework for distributed computing over the Internet, Grid computing – Grid 2000, Proceedings, 2001, pp. 36-46.

11. Henry Z. Kister, Distillation design , New York: McGraw-Hill Inc, 1992.

12. W. L. Luyben, Process modeling simulation and control for chemical engineering, New York: McGraw-Hill Book Company, 1974.

13. J. M. Smith and H. C. Van Ness, Introduction to chemical engineering thermodynamics, 4th Edition, New York,:McGraw-Hill Book Company, 1987.

14. W. L. Luyben, Practical distillation control, New York: Van Nostrand Reinhold, 1992.

15. H. El-Rewini and T. G. Lewis, Distributed and parallel computing, Greenwich: Manning Publications Co., 1998.

16. William Gropp, Ewing Lusk and Anthony Skjellum, Using MPI Portable Parallel Programming with the message-Passing Interface, London: The MIT Press, 1994

17. Dennis Wright, Basic Programs for Chemical Engineers, New york: Van Nostrand Reinhold, 1986.

18. Cricket L., Jerry P., Russ J., Bryan B., and Adrian N., Managing Internet Information Services, CA: O'Reilly & Associates, Inc., 1994.

19. W. Richard Stevens, UNIX Network Programming Volume 1, 2nd Edition, NJ: Prentice-Hall Interational, Inc., 1998.

20. Ian Foster, Designing and Building Parallel Programs, USA: Addison-Wesley Publishing Company, Inc., 1995.

21. W. Richard Stevens, UNIX Network Programming, New Jersey: Prentice-Hall, Inc., 1990.

22. Philip A. Schweitzer, editor, Handbook of Separation Techniques for Chemical Engineers, USA: McGraw-Hill, Inc., 1979.

23. Charles D. Holland and Athanasios I. Liapis, Computer Methods for Solving Dynamic Separation Problems, USA: McGraw-Hill, Inc., 1983.

24. S. Lakshmivarahan and Sudarshan K. Dhall, Analysis and Design of Parallel Algorithms Arithmetic and Matrix Problems, Singapore: McGraw-Hill, Inc., 1990.

25. George Coulouris, Jean Dollimore and Tim Kindberg, Distributed Systems Concepts and Design, 2nd Edition, USA: Addison-Wesley Publishing Company Inc., 1994.

26. Perry R.H. and D. Green, editor, Perry's Chemical Engineers' Handbook, 6th Edition, New York: McGraw-Hill Book Co., 1984.

27. Warin Iamteerapaiboon. Dynamics Compartmental Model for Multicomponent Distillation. Graduate School, Chulalongkorn University, 1994.

28. Kallaya Klaithong, A Comparative Study of Thermodynamic Models for Dynamic Simulation of Continuous Distillation, Graduate School, Chulalongkorn University, 1994.

29. Eric W. Weisstein, Percentage Error -- from MathWorld [Online], 1999, Available from: http://mathworld.wolfram.com/PercentageError.html [2002,April 28]

APPENDICES

APPENDIX A

LINUX SETUP

Linux Specifiction

- Recommendation : Linux Slackware 7.1, Harddisk 240 MB, Memory 64 MB
- Choose install compiler (C, C++, and/or Fortran) , Network (telnet, ftp, rlogin) and Xwindow

Setup Linux

- Setup /etc/hosts add your hostname to list.
- Setup path add "." to /etc/export file.
- On login check rlogin to another computer. If can't add file .rlogin to home directory.

Comment

- On Linux Redhat you must open service rlogin.
- About network file system (NFS), It make comfortable to run MPI. But It uses many resource for networking.

APPENDIX B

MPI INSTALLATION

Installation

- Copy mpich.tar.gz to /usr/local .
- Extract mpich.tar.gz by command "tar -xvf mpich.tar.gz" or "tar -xzf mpich.tar.gz".
- In directory “mpich”, use command "configure" or "./configure" and wait for a minutes. When stop, it create new file "./config.status".
- When got file "config.status". Using command "make" or "make PREFIX=/usr/local/mpich install".
- If no problem, Editing file “/usr/local/mpich/util/machines/machines.Linux” adds hostname.
- Checking file “/etc/hosts”, “/etc/hosts.equiv” and “~(home directory)/.rshosts” have hostname in group.

Compile & run

- Before running copy executable program to each computer to run. if use NFS not copy it.
- The C language uses command "mpicc" for compile and "mpirun" for run.
- The Fortran 77 language uses command "mpif77" for compile and "mpirun" for run.
- example : "mpicc cpi.c -o cpi", "mpif77 fpi.f -o fpi", "mpirun -np 2 cpi", "mpirun -np 4 fpi" or "mpirun -np 3 fpi -machinesfile hostlist"
- For Linux Redhat edits additional file “/etc/inetd.conf”.

Sample Error

- Checking command “rlogin”. If can not login then can not run.
- Can running program, but show message “permission denied” or “file not found”. Checking each computer has “+x” on file permission.
- Checking communication time between each computer.

APPENDIX C

LIST OF PARALLEL BINARY DISTILLATION PROGRAM

```
      PROGRAM MAIN
      INCLUDE "MPIF.H"
      DIMENSION X(20),Y(20),L(20),LO(20),M(20)
      DIMENSION MX(20),MDOT(20),MXDOT(20)
      REAL L,LO,M,MX,MDOT,MXDOT,MDO,MO,MBO,KCD,KCB
      REAL Z
      REAL BUFFER(100)
      INTEGER BLOCK,START,ISTOP,MID,ED,ST
      INTEGER MYID,MASTER,NUMPROCS,IERR,STAT
     +(MPI_STATUS_SIZE)
      EQUIL(XX)=ALPHA*XX/(1.+(ALPHA-1.)*XX)
      DATA NT,NF,MDO,MBO,MO,RO,VO,F,BETA,ALPHA
     +/20,10,100.,100.,10.,128.01,178.01,100.,0.1,2./
      DATA XB,X,XD
     +/.02,.035,.05719,.08885,.1318,.18622,.24951,
     +.31618,.37948,.43391,.47688,.51526,.56295,.61896,.68052,
     +.74345,.80319,.85603,.89995,.93458,.96079,.98/
      DATA KCD,KCB,TAUD,TAUB,DELTA,TIME,TPRINT,ERINTD,ERINTB/
     +1000.,1000.,5.,1.25,.005,4*0./
      Z=0.55
      MASTER=0
      CALL MPI_INIT(IERR)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYID,IERR)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NUMPROCS,IERR)

      ST=MPI_WTIME()

      BLOCK=NT/NUMPROCS
      START=BLOCK*MYID+1
      ISTOP=BLOCK*(MYID+1)
      MID=10/BLOCK
      IF(MYID.EQ.NUMPROCS-1)THEN
        ISTOP=NT
      END IF
      WRITE(*,*) MYID,BLOCK,START,ISTOP,MID
      COUNT=0
      IF(MYID.EQ.MASTER)THEN
        WRITE(6,1) Z,F
1       FORMAT(7X,'Z= ',F10.5,' F = ',F10.2)
      END IF
      DO 3 N=START,ISTOP
      M(N)=MO
      MX(N)=M(N)*X(N)
      LO(N)=RO+F
      IF(N.GT.NF)THEN
          LO(N)=RO
      END IF
3     CONTINUE

      IF(MYID.EQ.MASTER)THEN
        WRITE(6,2)
2       FORMAT(6X,' TIME     XB     X10     XD       R       V')
      END IF
100   DO 20 N=START,ISTOP
      Y(N)=EQUIL(X(N))
      L(N)=LO(N)+(M(N)-MO)/BETA
20    CONTINUE
      IF(MYID.EQ.MASTER)THEN
```

```
          YB=EQUIL(XB)
          ERRB=.02-XB
          V=VO-KCB *(ERRB+ERINTB/TAUB)
          B=L(1)-V
        END IF
        CALL MPI_BCAST(B,1,MPI_REAL,MASTER,MPI_COMM_WORLD,IERR)
        CALL MPI_BCAST(V,1,MPI_REAL,MASTER,MPI_COMM_WORLD,IERR)
        IF(MYID.EQ.NUMPROCS-1)THEN
          ERRD=.98-XD
          R=RO+KCD *(ERRD+ERINTD/TAUD)
          D=V-R
        END IF
          CALL MPI_BCAST(R,1,MPI_REAL,NUMPROCS-1,MPI_COMM_WORLD,IERR)
          CALL MPI_BCAST(D,1,MPI_REAL,NUMPROCS-1,MPI_COMM_WORLD,IERR)
        IF(R.LT.0.) GO TO 500
        IF(V.LT.0.) GO TO 500
        IF(D.LT.0.) GO TO 500
        IF(B.LT.0.) GO TO 500
        IF(MYID.NE.NUMPROCS-1)THEN
          CALL MPI_SEND(Y(ISTOP),1,MPI_REAL,MYID+1,0,
     +    MPI_COMM_WORLD,IERR)
        END IF
        IF(MYID.NE.MASTER)THEN
          CALL MPI_RECV(Y(START-1),1,MPI_REAL,MYID-1,0,
     +    MPI_COMM_WORLD,STAT,IERR)
          BUFFER(1)=L(START)
          BUFFER(2)=X(START)
          CALL MPI_SEND(BUFFER,2,MPI_REAL,MYID-1,1,
     +    MPI_COMM_WORLD,IERR)
        END IF
        IF(MYID.NE.NUMPROCS-1)THEN
          CALL MPI_RECV(BUFFER,2,MPI_REAL,MYID+1,1,
     +    MPI_COMM_WORLD,STAT,IERR)
          L(ISTOP+1)=BUFFER(1)
          X(ISTOP+1)=BUFFER(2)
        END IF
        IF(MYID.EQ.MASTER)THEN
          XBDOT=(L(1)*X(1)-V*YB-B*XB)/MBO
          MDOT(1)=L(2)-L(1)
          MXDOT(1)=V*(YB-Y(1))+L(2)*X(2)- L(1)*X(1)
        END IF
        IF(MYID.EQ.MASTER)THEN
          ST=2
        ELSE
          ST=START
        END IF
        IF(MYID.EQ.NUMPROCS-1)THEN
          ED=NT-1
        ELSE
          ED=ISTOP
        END IF
        DO 30 N=ST,ED
        MDOT(N)=L(N+1)- L(N)
        MXDOT(N)=V*(Y(N-1)-Y(N))+ L(N+1)*X(N+1)- L(N)*X(N)
        IF(N.EQ.NF)THEN
          MDOT(N)=MDOT(N)+F
          MXDOT(N)=MXDOT(N)+F*Z
        END IF
30      CONTINUE
        IF(MYID.EQ.NUMPROCS-1)THEN
          MDOT(NT)=R-L(NT)
          MXDOT(NT)=V*(Y(NT-1)-Y(NT))+R*XD-L(NT)*X(NT)
          XDDOT=V*(Y(NT)-XD)/MDO
        END IF
        IF(TIME.LT.TPRINT) GO TO 50
        CALL MPI_BCAST(X(10),1,MPI_REAL,MID,
     +      MPI_COMM_WORLD,IERR)
        CALL MPI_BCAST(XD,1,MPI_REAL,NUMPROCS-1,
     +      MPI_COMM_WORLD,IERR)
```

```
      IF(MYID.EQ.MASTER)THEN
        WRITE(6,41) TIME,XB,X(5),X(10),XD,R,V
41      FORMAT(7X,F5.1,4F9.5,2F9.2)
      END IF
      TPRINT=TPRINT+.5
50    CONTINUE
      TIME=TIME+DELTA
      IF(MYID.EQ.MASTER)THEN
        XB=XB+DELTA*XBDOT
        ERINTB=ERINTB+ERRB*DELTA
      END IF
      SUM=0
      DO 60 N=START,ISTOP
      M(N)=M(N)+MDOT(N)*DELTA
      MX(N)=MX(N)+MXDOT(N)*DELTA
      X(N)=MX(N)/M(N)
      IF(X(N).LT.0.) GO TO 500
      IF(X(N).GT.1.) GO TO 500
60    CONTINUE
      IF(MYID.EQ.NUMPROCS-1)THEN
        XD=XD+XDDOT*DELTA
        ERINTD=ERINTD+ERRD*DELTA
      END IF
      IF(TIME.LE.100.) GO TO 100
        CALL MPI_FINALIZE(IERR)

        ED=MPI_WTIME()
        WRITE(*,*) MYID,ED,ST,ED-ST

      STOP
 500  WRITE(6,501) R,V,D,B
 501  FORMAT(7X,'LEVEL TOO LOW OR ',F10.2,F10.2,F10.2,F10.2)
      WRITE(*,*)

      CALL MPI_FINALIZE(IERR)
      STOP
      END
```

APPENDIX D

LIST OF PARALLEL MULTICOMPONENT DISTILLATION PROGRAM

Input initial condition program by Active Server Pages (ASP)

- Program requirement

This ASP web page requires IIS version 4.0 or PWS or ASP host.

- List of input initial condition program

```
<%Function subs(A)%>
<sub><%=A%></sub>
<%end function%>

<%Function sup(A)%>
<sup><%=A%></sup>
<%end function%>

<%
dim
Name(20),DENS(20),MW(20),HVAP(20),BPT(20),HCAPV(20),HCAPL(20),VP1(20)
,VP2(20),T1(20),T2(20)
dim XF(20),YF(20)
dim T0(200),LO(200),XO(1000)
NC=request.form("NC")
NT=request.form("NT")
NF=request.form("NF")
WHS=request.form("WHS")
WHR=request.form("WHR")
DS=request.form("DS")
DR=request.form("DR")
WLS=request.form("WLS")
WLR=request.form("WLR")
MVB=request.form("MVB")
MVD=request.form("MVD")
FL=request.form("FL")
TFL=request.form("TFL")
FV=request.form("FV")
TFV=request.form("TFV")
PD=request.form("PD")
PB=request.form("PB")
QR=request.form("QR")
R=request.form("R")
DV=request.form("DV")
EFF=request.form("EFF")
STEP=request.form("STEP")
STOP1=request.form("STOP1")
DELTA=request.form("DELTA")
Data1=request.form("Data1")
for i=1 to NC
Name(i)=request.form("Name" & i)
MW(i)=request.form("MW" & i)
DENS(i)=request.form("DENS" & i)
HVAP(i)=request.form("HVAP" & i)
BPT(i)=request.form("BPT" & i)
HCAPV(i)=request.form("HCAPV" & i)
HCAPL(i)=request.form("HCAPL" & i)
VP1(i)=request.form("VP1" & i)
VP2(i)=request.form("VP2" & i)
```

```
T1(i)=request.form("T1" & i)
T2(i)=request.form("T2" & i)
XF(i)=request.form("XF" & i)
YF(i)=request.form("YF" & i)
next
for j=0 to NT+1
T0(j)=request.form("T0" & j)
LO(j)=request.form("LO" & j)
for i=1 to NC
XO(j*NC+i)=request.form("XO" & (j*NC+i))
next
next
%>
<html>
<head><title>Distillation Model</title></head>
<script language="Javascript">
function gencode() {
  alert('Gen code');
  f1=form1.data1
  f1.value=""
  f1.value+=form1.NC.value+'\n'
  f1.value+=form1.NT.value+'\n'
 f1.value+=form1.NT.value+','+form1.NF.value+','+form1.WHS.value+','+
form1.WHR.value+','+form1.DS.value+','+form1.DR.value+','+form1.WLS.v
alue+','+form1.WLR.value+','+form1.MVB.value+','+form1.MVD.value+'\
n'
  <%For i=1 to NC-1%>
  f1.value+=form1.Name<%=i%>.value+','
  <%next%>  f1.value+=form1.Name<%=NC%>.value+'\n'
  <%For i=1 to NC-1%>
  f1.value+=form1.HVAP<%=i%>.value+','
  <%next%>  f1.value+=form1.HVAP<%=NC%>.value+'\n'
  <%For i=1 to NC-1%>
  f1.value+=form1.HCAPV<%=i%>.value+','
  <%next%>  f1.value+=form1.HCAPV<%=NC%>.value+'\n'
  <%For i=1 to NC-1%>
  f1.value+=form1.HCAPL<%=i%>.value+','
  <%next%>  f1.value+=form1.HCAPL<%=NC%>.value+'\n'
  <%For i=1 to NC-1%>
  f1.value+=form1.VP1<%=i%>.value+','
  <%next%>  f1.value+=form1.VP1<%=NC%>.value+'\n'
  <%For i=1 to NC-1%>
  f1.value+=form1.T1<%=i%>.value+','
  <%next%>  f1.value+=form1.T1<%=NC%>.value+'\n'
  <%For i=1 to NC-1%>
  f1.value+=form1.VP2<%=i%>.value+','
  <%next%>  f1.value+=form1.VP2<%=NC%>.value+'\n'
  <%For i=1 to NC-1%>
  f1.value+=form1.T2<%=i%>.value+','
  <%next%>  f1.value+=form1.T2<%=NC%>.value+'\n'
  f1.value+=form1.TFL.value+','+form1.FL.value+','
  <%For i=1 to NC-1%>
  f1.value+=form1.XF<%=i%>.value+','
  <%next%>  f1.value+=form1.XF<%=NC%>.value+'\n'
  f1.value+=form1.TFV.value+','+form1.FV.value+','
  <%For i=1 to NC-1%>
  f1.value+=form1.YF<%=i%>.value+','
  <%next%>  f1.value+=form1.YF<%=NC%>.value+'\n'
  f1.value+=form1.PD.value+','+form1.PB.value+','+form1.QR.value+','
+form1.R.value+','+form1.DV.value+','+form1.EFF.value+'\n'
  <%For j=0 to NT%>
  f1.value+=form1.T0<%=j%>.value+','
  <%next%>  f1.value+=form1.T0<%=(NT+1)%>.value+'\n'
  <%For j=0 to NT%>
  f1.value+=form1.LO<%=j%>.value+','
  <%next%>  f1.value+=form1.LO<%=(NT+1)%>.value+'\n'
  f1.value+=(form1.NC.value-(-2))+'\n'
  <%For i=1 to NC-1%>
  f1.value+=form1.XO<%=i%>.value+','
  <%next%>  f1.value+=form1.XO<%=NC%>.value+',\n'
  <%For i=1 to NC%>
  <%For j=1 to NT%>
  f1.value+=form1.XO<%=j*NC+i%>.value+','
  <%next%>  f1.value+='\n'
  <%next%>
  <%For i=1 to NC-1%>
  f1.value+=form1.XO<%=(NT+1)*NC+i%>.value+','
  <%next%>  f1.value+=form1.XO<%=(NT+1)*NC+NC%>.value+'\n'
  <%For i=1 to NC%>
  f1.value+=form1.MW<%=i%>.value+'\n'
  <%next%>
  <%For i=1 to NC%>
  f1.value+=form1.DENS<%=i%>.value+'\n'
  <%next%>
  <%For i=1 to NC%>
  f1.value+=form1.BPT<%=i%>.value+'\n'
  <%next%>
  f1.value+=form1.STEP.value+'\n'
  f1.value+=form1.STOP1.value+'\n'
  f1.value+=form1.DELTA.value+'\n'
  f1.value+='0.\n0.\n822.\n'
}
function clearcode() {
  f1=form1.data1
  f1.value=""
}
</script>
<style type="text/css">
@import url(main.css);
select, input {border-RIGHT: black 1px double; border-TOP: black 1px
double; FONT-SIZE: 8pt; border-LEFT: black 1px double; border-BOTTOM:
black 1px double}
</style>
<body bgcolor=#FFFFFF leftmargin="0" topmargin="0">
<div align="center"><table width="700"><tr><td>
<%if len(request.form("data1"))=0 then
' check data in textarea%>

<table border="0" width="100%">
<form action="gencode2.asp" method="post" name="form1">
<tr>
 <td><font size="2">
 Number of Tray
 </font></td>
 <td><font size="2">
 <input type="text" name="NT" value="<%=NT%>">
 </font></td>
 <td><font size="2">
 Number of component
 </font></td>
 <td><font size="2">
 <input type="text" name="NC" value="<%=NC%>">
 </font></td>
</tr>
<%if len(NC)>0 and len(NT)>0 then
'check number of component and tray%>
<tr>
 <td><font size="2">
 Murphree vapor-phase tray efficiency, EFF
 </font></td>
 <td><font size="2">
 <input type="text" name="EFF" value="<%=EFF%>">
 </font></td>
 <td><font size="2">
 Feed Tray
 </font></td>
 <td><font size="2">
 <input type="text" name="NF" value="<%=NF%>">
 </font></td>
</tr>
<tr>
 <td><font size="2">
 &Delta;T,DELTA (Hours)
 </font></td>
 <td><font size="2">
 <input type="text" name="DELTA" value="<%=DELTA%>">
 </font></td>
 <td><font size="2">
 STEP show result,STEP (Hours)
 </font></td>
 <td><font size="2">
 <input type="text" name="STEP" value="<%=STEP%>">
 </font></td>
</tr>
<tr>
 <td><font size="2">
 End time step,STOP (Hours)
 </font></td>
 <td><font size="2">
 <input type="text" name="STOP1" value="<%=STOP1%>">
 </font></td>
 <td><font size="2">
 &nbsp;
 </font></td>
 <td><font size="2">
 &nbsp;
 </font></td>
</tr>


<tr><td colspan="4">
<table border="0" width="100%">
<tr>
<td rowspan="2"><div align="center"><font size="2">
&nbsp;
</font></div></td>
<td rowspan="2"><div align="center"><font size="2">
Flow rate,F <br>(lb mol/h)
</font></div></td>
<td rowspan="2"><div align="center"><font size="2">
temperature,TF <br>(lb mol/h)
</font></div></td>
<td colspan="<%=NC%>"><div align="center"><font size="2">
Composition, XF or YF (m.f.)
</font></div></td>
</tr>
<tr>
<%For i=1 to NC%>
<td><div align="center"><font size="2">
<%=i%>
</font></div></td>
<%next%>
</tr>
<tr>
<td><div align="center"><font size="2">
Liquid feed
</font></div></td>
<td><div align="center"><font size="2">
<input type="text" name="FL" value="<%=FL%>" size="10">
</font></div></td>
<td><div align="center"><font size="2">
<input type="text" name="TFL" value="<%=TFL%>" size="10">
</font></div></td>
<%For i=1 to NC%>
<td><div align="center"><font size="2">
<input type="text" name="XF<%=i%>" value="<%=XF(i)%>" size="10">
```

```
</font></div></td>
<%next%>
</tr>
<tr>
<td><div align="center"><font size="2">
Vapor feed
</font></div></td>
<td><div align="center"><font size="2">
<input type="text" name="FV" value="<%=FV%>" size="10">
</font></div></td>
<td><div align="center"><font size="2">
<input type="text" name="TFV" value="<%=TFV%>" size="10">
</font></div></td>
<%For i=1 to NC%>
<td><div align="center"><font size="2">
<input type="text" name="YF<%=i%>" value="<%=YF(i)%>" size="10">
</font></div></td>
<%next%>
</tr>
</table>
</td></tr>

<tr>
 <td colspan="2"><font size="2">
 &nbsp;
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2"><b>
 Stripping section
 </b</font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2"><b>
 Rectifying section
 </b></font></div></td>
</tr>
<tr>
 <td colspan="2"><font size="2">
 Weir height, WH (in)
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2">
 <input type="text" name="WHS" value="<%=WHS%>">
 </font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2">
 <input type="text" name="WHR" value="<%=WHR%>">
 </font></div></td>
</tr>
<tr>
 <td colspan="2"><font size="2">
 Weir length, WL (in)
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2">
 <input type="text" name="WLS" value="<%=WLS%>">
 </font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2">
 <input type="text" name="WLR" value="<%=WLR%>">
 </font></div></td>
</tr>
<tr>
 <td colspan="2"><font size="2">
 Column diameter, D (in)
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2">
 <input type="text" name="DS" value="<%=DS%>">
 </font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2">
 <input type="text" name="DR" value="<%=DR%>">
 </font></div></td>
</tr>

<tr> <td colspan="4"><font size="2"> &nbsp; </font></td></tr>
```

```
<tr>
 <td colspan="2"><font size="2">
 &nbsp;
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2"><b>
 Column base
 </b</font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2"><b>
 Reflux drum
 </b></font></div></td>
</tr>
<tr>
 <td colspan="2"><font size="2">
 Volumetric holdup, MV (ft<%sup("3")%>)
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2">
 <input type="text" name="MVB" value="<%=MVB%>">
 </font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2">
 <input type="text" name="MVD" value="<%=MVD%>">
 </font></div></td>
</tr>
<tr>
 <td colspan="2"><font size="2">
 Pressure, P (psia)
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2">
 <input type="text" name="PB" value="<%=PB%>">
 </font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2">
 <input type="text" name="PD" value="<%=PD%>">
 </font></div></td>
</tr>
<tr>
 <td colspan="2"><font size="2">
 Product flow rate,D (lb mol/h)
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2">
 <input type="text" name="DL" value="<%=DL%>">
 </font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2">
 <input type="text" name="DV" value="<%=DV%>">
 </font></div></td>
</tr>
<tr>
 <td colspan="2"><font size="2">
 Reboiler heat input, QR (10<%sup("6")%> Btu/h)
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2">
 <input type="text" name="QR" value="<%=QR%>">
 </font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2">
 &nbsp;
 </font></div></td>
</tr>
<tr>
 <td colspan="2"><font size="2">
 Reflux flow rate, R (lb mol/h)
 </font></td>
 <td bgcolor="#CCFF99"><div align="center"><font size="2">
 &nbsp;
 </font></div></td>
 <td bgcolor="#99CCFF"><div align="center"><font size="2">
 <input type="text" name="R" value="<%=R%>">
 </font></div></td>
</tr>

<tr><td colspan="4">
<table border="0" width="100%">
```

```
<tr>
<td><font size="2">
&nbsp;
</font></td>
 <td colspan="<%=NC%>" bgcolor="#99FFCC"><div align="center"><font
size="2"><b>
  Component
 </b></font></div></td>
</tr>
<tr>
<td><font size="2">
&nbsp;
</font></td>
<%For i=1 to NC%>
 <td bgcolor="#99FF99"><div align="center"><font size="2"><b>
  <%=i%>
 </b></font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
Component Name
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="Name<%=i%>" value="<%=Name(i)%>"
size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
Molecular weight, MW (lb<%subs("m")%>/lb mol)
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="MW<%=i%>" value="<%=MW(i)%>" size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
Density, DENS
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="DENS<%=i%>" value="<%=DENS(i)%>"
size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
Heat of vaporization at normal boiling point, HVAP
(Btu/lb<%subs("m")%>)
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="HVAP<%=i%>" value="<%=HVAP(i)%>"
size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
Boiling point temperature, BPT (<%sup("o")%>F)
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="BPT<%=i%>" value="<%=BPT(i)%>" size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
Heat capacity of vapor, HCAPV (Btu/lb<%subs("m")%> <%sup("o")%>F)
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="HCAPV<%=i%>" value="<%=HCAPV(i)%>"
size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
Heat capacity of liquid, HCAPL (Btu/lb<%subs("m")%> <%sup("o")%>F)
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="HCAPL<%=i%>" value="<%=HCAPL(i)%>"
size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
Vapor pressure, VP1 (psia)
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="VP1<%=i%>" value="<%=VP1(i)%>" size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
at temperature, T1 (<%sup("o")%>F)
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="T1<%=i%>" value="<%=T1(i)%>" size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
Vapor pressure, VP2 (psia)
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="VP2<%=i%>" value="<%=VP2(i)%>" size="10">
 </font></div></td>
<%next%>
</tr>
<tr>
<td><font size="2">
at temperature, T2 (<%sup("o")%>F)
</font></td>
<%For i=1 to NC%>
 <td><div align="center"><font size="2">
 <input type="text" name="T2<%=i%>" value="<%=T2(i)%>" size="10">
 </font></div></td>
<%next%>
</tr>
</table>
</td></tr>
```

```
<tr><td colspan="4" bgcolor="#CCFF99"><font size="2"><b>
<center>Initial condition</center>
</b></font></td></tr>
<tr><td colspan="4">
<table border="0" width="100%">
<tr bgcolor="#99CCFF">
<td rowspan="2"><div align="center"><font size="2"><b>
N
</b></font></div></td>
<td rowspan="2"><div align="center"><font size="2"><b>
Temperature,T <br>(<%sup("o")%>F)
</b></font></div></td>
<td rowspan="2"><div align="center"><font size="2"><b>
Liquid flow rate, LO <br>(lb mol/h)
</b></font></div></td>
<td colspan="<%=NC%>"><div align="center"><font size="2"><b>
Liquid composition, X (m.f.)
</b></font></div></td>
</tr>
<tr bgcolor="#99CCFF">
<%For i=1 to NC%>
<td><div align="center"><font size="2"><b>
<%=i%>
</b></font></div></td>
<%next%>
</tr>
<%For J=0 to NT+1%>
<tr>
<td><div align="center"><font size="2"><b>
<%=J%>
</b></font></div></td>
<td><div align="center"><font size="2"><b>
<input type="text" name="T0<%=J%>" value="<%=T0(j)%>" size="10">
</b></font></div></td>
<td><div align="center"><font size="2"><b>
<input type="text" name="LO<%=J%>" value="<%=LO(j)%>" size="10">
</b></font></div></td>
<%For i=1 to NC%>
<td><div align="center"><font size="2"><b>
<input type="text" name="XO<%=(J*NC+i)%>" value="<%=XO(j*NC+i)%>"
size="10">
</b></font></div></td>
<%next%>
</tr>
<%next%>
</table>
</td></tr>
<tr>
 <td colspan="4"><div align="center"><font size="2">
 <input type="button" value="TRANSFER DATA" onclick="gencode()">
 <input type="button" value="CLEAR DATA" onclick="clearcode()">
 </font></div></td>
</tr>
<tr>
 <td colspan="4"><div align="center"><font size="2">
 <textarea cols="160" rows="10" name="data1"><%=Data1%></textarea>
 </font></div></td>
</tr>
<%end if%>
<tr> <td colspan="4"><font size="2"> &nbsp; </font></td></tr>
<tr>
 <td colspan="4"><div align="center"><font size="2">
 <input type="submit" value="Generate code">
 <input type="reset" value="Clear">
 </font></div></td>
</tr>
</form>
</table>
<%else%>
```

```
 <textarea cols="160" rows="40" name="data1"><%=Data1%></textarea>
<%end if%>
</td></tr></table>
</div>
</body>
</html>
```

- Demo program

1. Start input number of tray and number of component in input box. And click "generate code" button.

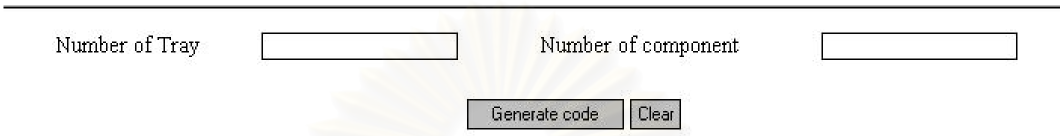
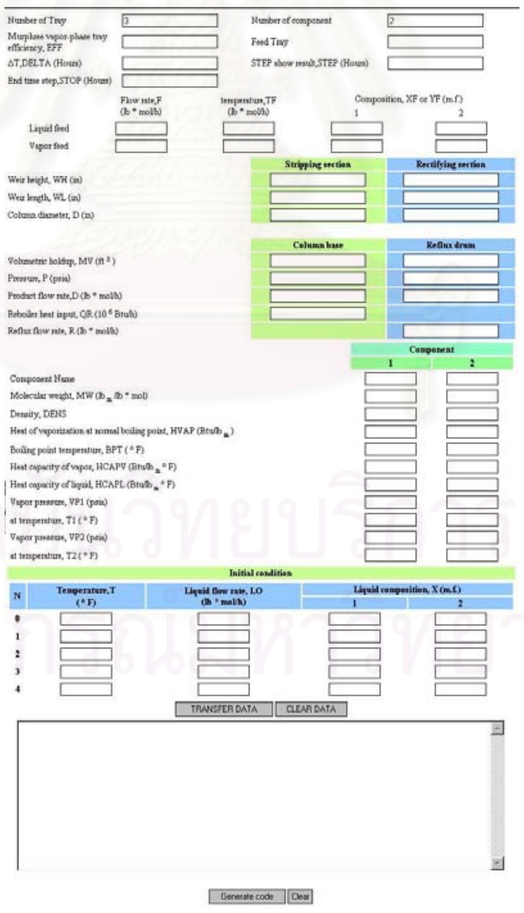


Figure D.1 First page input initial condition program

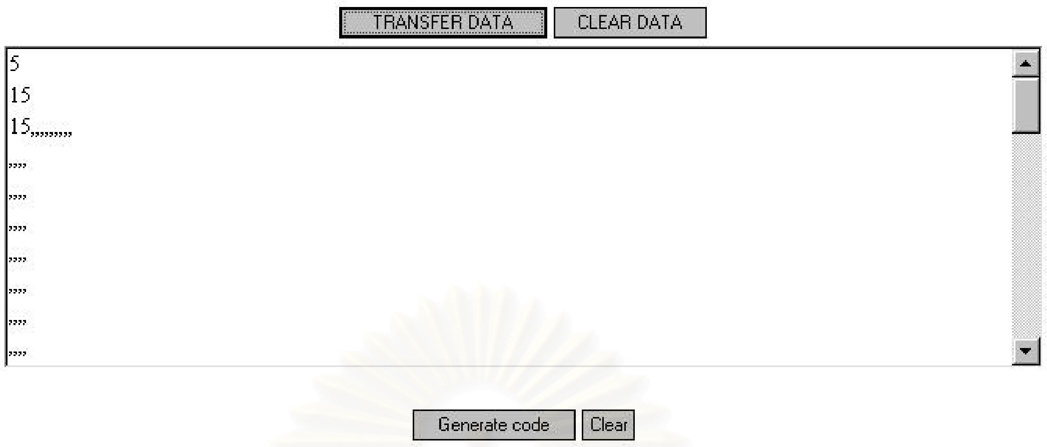
2. Input column condition, component physical properties and initial steady state tray condition. And click "TRANSFER DATA" button. All initial condition will show the initial condition in below rectangle box. Copy the data to file.



Number of Tray
Number of component
Murphree vapor phase tray efficiency, EFF
Feed Tray
ΔT,DELTA (Hours)
STEP show result,STEP (Hours)
End time step,STOP (Hours)
Flow rate,F (lb * mol/h)
temperature,TF (lb * mol/h)
Composition, XF or YF (m.f)
1
2
Liquid feed
Vapor feed
Stripping section
Rectifying section
Weir height, WH (in)
Weir length, WL (in)
Column diameter, D (in)
Column base
Reflux drum
Volumetric holdup, MV (ft ³)
Pressure, P (psia)
Product flow rate,D (lb * mol/h)
Reboiler heat input, QR (10 ⁶ Btu/h)
Reflux flow rate, R (lb * mol/h)
Component
1
2
Component Name
Molecular weight, MW (lb m /lb * mol)
Density, DENS
Heat of vaporization at normal boiling point, HVAP (Btu/lb m)
Boiling point temperature, BPT (° F)
Heat capacity of vapor, HCAPV (Btu/lb m ° F)
Heat capacity of liquid, HCAPL (Btu/lb m ° F)
Vapor pressure, VP1 (psia)
at temperature, T1 (° F)
Vapor pressure, VP2 (psia)
at temperature, T2 (° F)
Initial condition
N
Temperature,T (° F)
Liquid flow rate, LO (lb * mol/h)
Liquid composition, X (m.f.)
1
2
0
1
2
3
4
TRANSFER DATA
CLEAR DATA
Generate code
Clear

Number of Tray: 3
Number of component: 2

Figure D.2 Second page input initial condition program



```
5
15
15,,,,,,,,,
,,,,
,,,,
,,,,
,,,,
,,,,
,,,,
,,,,
```

Figure D.3 The output initial condition

<u>Generate parallel program by PASCAL language</u>

- Program requirement

DOS 6.0., initial condition file and file “P8-1.txt”

- List of generate parallel program

```
program parallel_gencode;
var f1,f2,st : string;
    t1,t2 : text;
    nc,nt,i,j : integer;
begin
    write('input file output: '); readln(f1);
    assign(t1,f1);
    rewrite(t1);
    write('input file input: '); readln(f2);
    assign(t2,f2);
    reset(t2);
    writeln(t1,'C SIMPLE MODEL on MPI');
    writeln(t1,'        program main');
    writeln(t1,'        include "mpif.h"');
    writeln(t1,'        real buffer(100)');
    writeln(t1,'        integer block,start,istop,done,count(3)');
    writeln(t1,'        double precision st,ed');
    writeln(t1,'        integer myid,master,numprocs,ierr,stat
(MPI_STATUS_SIZE)');
    writeln(t1,'        REAL MW,LO,MVB,MVD,MWA,MV,LV,M,L,MB,MD');
    writeln(t1,'  CHARACTER*6 NAME');
    readln(t2,nc);
    readln(t2,nt);
    writeln(t1,'  COMMON NC,MW(',nc,'),DENS(',nc,'),C1(',nc,'),C2
(',nc,'),C3(',nc,'),');
    writeln(t1,'     +  BPT(',nc,'),AVP(',nc,'),BVP(',nc,')');
    writeln(t1,'  DIMENSION LV(',nt,'),L(',nt,'),P(',nt,'),XF
',nc,'),YF(',nc,'),DXD(',nc,'),YAV(',nc,'),');
    writeln(t1,'     +  YY(',nc,'),HL(',nt,'),HV(',nt,'),V(',nt,'),DM
(',nt,'),DXM(',nt,',',nc
    ,'),XM(',nt,',',nc,'),DXB(',nc,')');
    writeln(t1,'  DIMENSION NAME(',nc,'),T(',nt,'),XB(',nc,'),X
(',nt,',',nc,'),Y(',nt,',',nc,'),LO(',nt,')');
    writeln(t1,'     +  ,XD(',nc,'),YB(',nc,'),YD(',nc,'),XX
(',nc,'),MV(',nt,'),M(',nt,'),HCAPV(',nc,'),HCAPL(',nc,')');
```

```
    writeln(t1,'     +  ,VP1(',nc,'),VP2(',nc,'),T1(',nc,'),T2
(',nc,'),HVAP(',nc,')');
    writeln(t1,'  DATA NT,NF,WHS,WHR,DS,DR,WLS,WLR,MVB,MVD/');
    readln(t2,st);
    writeln(t1,'     +  ',st,'/');
    readln(t2,st);
    writeln(t1,'  DATA NAME/',st,'/');
    readln(t2,st);
    writeln(t1,'  DATA HVAP/',st,'/');
    readln(t2,st);
    writeln(t1,'  DATA HCAPV/',st,'/');
    readln(t2,st);
    writeln(t1,'  DATA HCAPL/',st,'/');
    readln(t2,st);
    writeln(t1,'  DATA VP1/',st,'/');
    readln(t2,st);
    writeln(t1,'  DATA T1/',st,'/');
    readln(t2,st);
    writeln(t1,'  DATA VP2/',st,'/');
    readln(t2,st);
    writeln(t1,'  DATA T2/',st,'/');
    writeln(t1,'  DATA TFL,FL,XF/');
    readln(t2,st);
    writeln(t1,'     +  ',st,'/');
    writeln(t1,'  DATA TFV,FV,YF/');
    readln(t2,st);
    writeln(t1,'     +  ',st,'/');
    readln(t2,st);
    writeln(t1,'  DATA PD,PB,QR,R,DV,EFF/',st,'/');
    writeln(t1,'  DATA TB,T,TD/');
    readln(t2,st);
    writeln(t1,'     +  ',st,'/');
    writeln(t1,'  DATA LO/');
    readln(t2,st);
    writeln(t1,'     +  ',st,'/');
    writeln(t1,'  DATA XB,X,XD/');
    readln(t2,j);
    for i:=1 to j do
    begin
    readln(t2,st);
    writeln(t1,'     +  ',st);
    end;
    readln(t2,st);
    writeln(t1,'     +  ',st,'/');
    writeln(t1,'  NC=',nc);
    for i:=1 to nc do
    begin
    readln(t2,st);
    writeln(t1,'  MW(',i,')=',st);
    end;
    for i:=1 to nc do
    begin
    readln(t2,st);
    writeln(t1,'  DENS(',i,')=',st);
    end;
    for i:=1 to nc do
    begin
    readln(t2,st);
    writeln(t1,'  BPT(',i,')=',st);
    end;
    readln(t2,st);
    writeln(t1,'  STEP=',st); {'Step to show output'}
    readln(t2,st);
    writeln(t1,'  STOP1=',st); {'Stop loop'}
    readln(t2,st);
    writeln(t1,'  DELTA=',st); {'step time'}
    readln(t2,st);
    writeln(t1,'  TIME=',st); {'default =0'}
    readln(t2,st);
    writeln(t1,'  TPRINT=',st); {'default=0'}
    readln(t2,st);
    writeln(t1,'  V(NF)=',st); {'default=822'}

    close(t2);
    assign(t2,'P8-1.txt');
    reset(t2);
    while not(eof(t2)) do
    begin
    readln(t2,st);
    writeln(t1,st);
    end;
    close(t1);
    close(t2);
end.
```

- List of file “P8-1.txt”

```
        master=0
        done=0
        count(1)=0
        count(2)=0
        count(3)=0

C       MPI Initial

        call MPI_INIT(ierr)
        call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
        call MPI_COMM_SIZE(MPI_COMM_WORLD,numprocs,ierr)
        st=MPI_WTIME()
C       INITIAL CONDITION

        if(myid.eq.master)then
      WRITE(6,1)
1     FORMAT (' NT NF NC WHS WHR DS DR WLS WLR MVB MVD
     +      ')
      WRITE(6,3) NT,NF,NC,WHS,WHR,DS,DR,WLS,WLR,MVB,MVD
3     FORMAT(1X,3I3,1X,8F6.2)
      WRITE(6,*) 'NAME ','MW ','DENS ','HVAP ','BPT '
     +      ,'HCAPV ','HCAPL ','VP1 ','T1 ','VP2 ','T2 '
      DO 5 J=1,NC
      WRITE(6,4)NAME(J),MW(J),DENS(J),HVAP(J),BPT(J),HCAPV(J),
     +      HCAPL(J),VP1(J),T1(J),VP2(J),T2(J)
4     FORMAT(1X,A6,4F7.2,2F7.3,4F7.1)
5     CONTINUE
      WRITE(6,*) 'FL ','TFL ','XF1 ','XF2 ','XF3 ','XF4 '
     +      ,'XF5'
      WRITE(6,7) FL,TFL,(XF(J),J=1,NC)
7     FORMAT(1X,2F10.0,5E10.2)
      WRITE(6,*) 'FV ','TFV ','YF1 ','YF2 ','YF3 ','YF4 '
     +      ,'YF5'
      WRITE(6,7) FV,TFV,(YF(J),J=1,NC)
      WRITE(6,*) 'PD ','PB ','QR ','R ','DV ','EFF '
      WRITE(6,9) PD,PB,QR,R,DV,EFF
9     FORMAT(1X,10F8.2)
c     READ(*,*) II
      WRITE(6,10)
10    FORMAT(4X,'N   TEMP    L    X1    X2    X3    X4    X5')
      BLANK=0.
      WRITE(6,11)TB,BLANK,(XB(J),J=1,NC)
11    FORMAT(5X,2F8.2,5E10.3)
      DO 12 N=1,NT
12    WRITE(6,15)N,T(N),LO(N),(X(N,J),J=1,NC)
15    FORMAT(1X,I3,1X,2F8.2,5E10.3)
      WRITE(6,11 )TD,R,(XD(J),J=1,NC)
      WRITE(6,16) DELTA
16    FORMAT(1X,' DELTA = ',F8.5)
        end if

      DO 19 J=1,NC
      AVP(J)=(T1(J)+460.)*(T2(J)+460.)*ALOG(VP2(J)/VP1(J))/(T1(J)-
T2(J))
      BVP(J)=ALOG(VP2(J))-AVP(J)/(T2(J)+460.)
      C2(J)=HCAPV(J)*MW(J)
      C3(J)=HCAPL(J)*MW(J)
      C1(J)=HVAP(J)*MW(J)+(C3(J)-C2(J))*BPT(J)
C     WRITE(*,*) AVP(J),BVP(J),C1(J),C2(J),C3(J)
 19   CONTINUE
C     READ(*,*) II
      CALL ENTH(TFL,XF,YF,HLF,HVF)

C       CALCULATE INITIAL HOLDUP

        CALL MWDENS(TB,XB,MWA,DENSA)
        MB = MVB*DENSA/MWA
      DO 20 N=1,NT
      DO 21 J=1,NC
21    XX(J) = X(N,J)
      CALL MWDENS(T(N),XX,MWA,DENSA)
      LV(N) = LO(N) * MWA/DENSA
      L(N) = LO(N)
      IF(N.GE.NF)THEN
        HFOW = (LV(N)/(999.*WLR))**.66667
        MV(N) = (HFOW+WHR/12.)*3.1416*DR*DR/(4.*144.)
      ELSE
        HFOW = (LV(N)/(999.*WLS))**.66667
        MV(N) = (HFOW+WHS/12.)*3.1416*DS*DS/(4.*144.)
      END IF
      M(N) = MV(N)*DENSA/MWA
      DO 31 J=1,NC
      XM(N,J) = M(N)*X(N,J)
31    CONTINUE

C       CALCULATE PRESSURE PROFILE

      P(N)=(PB-(N*(PB-PD))/NT)
20    CONTINUE
      CALL MWDENS(TD,XD,MWA,DENSA)
      MD=MVD*DENSA/MWA

C       MAIN LOOP FOR EACH TIME STEP

        master=myid

100   CONTINUE
        if(done.gt.0)then
        block=NT/numprocs
        start=block*myid+1
        istop=block*(myid+1)
        if(myid.eq.numprocs-1) istop=NT
        done=done+1
c         write(*,*) myid,done,block,start,istop
        else
        start=1
        istop=NT
        done=1
c         write(*,*) myid,done,block,start,istop
        end if
c         if(myid.eq.0) read(*,*) ii
        if(myid.eq.master)then
      CALL BUBPT(TB,XB,YB,PB)
      CALL ENTH(TB,XB,YB,HLB,HVB)
        end if

      DO 110 N = start,istop
      DO 111 J = 1,NC
111   XX(J)=X(N,J)
      CALL BUBPT(T(N),XX,YY,P(N))
      DO 106 J=1,NC
      IF(N.EQ.1)THEN
        Y(N,J)=YB(J)+EFF*(YY(J)-YB(J))
      ELSE IF(N.EQ.NF+1)THEN
        YAV(J)=(YF(J)*FV+Y(N-1,J)*V(N-1))/(V(N-1)+FV)
        Y(N,J)=(YY(J)-YAV(J))*EFF+YAV(J)
      ELSE
        Y(N,J) =(YY(J)-Y(N-1,J))*EFF+Y(N-1,J)
      END IF
106   YY(J)=Y(N,J)
      CALL ENTH(T(N),XX,YY,HL(N),HV(N))
110   CONTINUE
        if(myid.eq.numprocs-1 .or. done.eq.0)then
      CALL BUBPT(TD,XD,YD,PD)
      CALL ENTH(TD,XD,YD,HLD,HVD)
        end if

C       CALCULATE VAPOR RATE
        if(myid.eq.master)then
      VB = (QR*1000000.-L(1)*(HLB-HL(1)))/(HVB-HLB)
C     WRITE(*,*) VB,QR,L(1),HLB,HL(1),HVB
      B = L(1)-VB
      IF (B .LT. 0.) STOP
        end if
      DO 120 N = start,istop
      IF(N.EQ.1)THEN
        V(N)  =  (HL(N+1)*L(N+1)+HVB*VB-HL(N)*L(N))/HV(N)
      ELSE IF(N.EQ.NF)THEN
        V(N)=(HL(N+1)*L(N+1)+HV(N-1)*V(N-1)-HL(N)*L(N)+HLF*FL)
     +         /HV(N)
      ELSE IF(N.EQ.NF+1)THEN
        V(N)=(HL(N+1)*L(N+1)+HV(N-1)*V(N-1)+HVF*FV-HL(N)*L(N))
     +         /HV(N)
      ELSE IF(N.EQ.NT)THEN
        V(N)=(HLD*R+HV(N-1)*V(N-1)-HL(N)*L(N))/HV(N)
      ELSE
        V(N)=(HL(N+1)*L(N+1)+HV(N-1)*V(N-1)-HL(N)*L(N))/HV(N)
      END IF
120   CONTINUE
        if(myid.eq.numprocs-1 .or. done.eq.0)then
      DL=V(NT)-DV-R
        end if


C       EVALUATE DERIVATIVES

      DO 160 J=1,NC
        if(myid.eq.master)then
      DXB(J)=(X(1,J)*L(1)-YB(J)*VB-XB(J)*B)/MB
        end if
      DO 140 N=start,istop
      IF(N.EQ.1)THEN
        DM(N)=L(N+1)+VB-V(N)-L(N)
        DXM(N,J)=X(N+1,J)*L(N+1)+YB(J)*VB-X(N,J)*L(N)-Y(N,J)*V(N)
      ELSE IF(N.EQ.NF)THEN
        DM(N)=L(N+1)+FL+V(N-1)-L(N)-V(N)
        DXM(N,J)=X(N+1,J)*L(N+1)+Y(N-1,J)*V(N-1)-X(N,J)*L(N)-V(N)
     +         *Y(N,J)+FL*XF(J)
      ELSE IF(N.EQ.NF+1)THEN
        DM(N)=L(N+1)+FV+V(N-1)-L(N)-V(N)
        DXM(N,J)=X(N+1,J)*L(N+1)+Y(N-1,J)*V(N-1)-X(N,J)*L(N)
     +         -V(N)*Y(N,J)+FV*YF(J)
      ELSE IF(N.EQ.NT)THEN
        DM(N)=R+V(N-1)-L(N)-V(N)
        DXM(N,J)=XD(J)*R+Y(N-1,J)*V(N-1)-X(N,J)*L(N)-Y(N,J)*V(N)
      ELSE
        DM(N)=L(N+1)+V(N-1)-L(N)-V(N)
        DXM(N,J)=X(N+1,J)*L(N+1)+Y(N-1,J)*V(N-1)-X(N,J)*L(N)-V(N)
     +         *Y(N,J)
      END IF
```

```
140     CONTINUE
        if(myid.eq.numprocs-1 .or. done.eq.0)then
      DXD(J)=(V(NT)*Y(NT,J)-DV*YD(J)-(R+DL)*XD(J))/MD
        end if
160   CONTINUE

C       PRINT DATA

      IF (TIME.GT.STOP1) GO TO 400
      IF (TIME.LT.TPRINT) GO TO 210
      WRITE(6,201) myid,master,TIME
201   FORMAT (3X,2I3,1X,F5.4,' TIME  T  X1  X2  X3  X4  X5  L V P')
        if(myid.eq.master .or. TPRINT.lt.STEP)then
      WRITE(6,202) TB,(XB(J),J=1,NC),B,VB,PB
202   FORMAT(3X,F7.2,5F9.6,3F7.1)
        end if
      DO 203 N=start,istop
203   WRITE(6,204) N,T(N),(X(N,J),J=1,NC),L(N),V(N),P(N)
204   FORMAT(I3,F7.2,5F9.6,3F7.1)
        if(myid.eq.numprocs-1 .or. done.eq.0)then
      WRITE(6,205) TD,(XD(J),J=1,NC),R,PD
205   FORMAT (3X,F7.2,5F9.6,F7.1,7X,F7.1)
      WRITE(6,206) (YD(J),J=1,NC),DL
206   FORMAT(10X,5F9.6,F7.1)
        end if
      TPRINT = TPRINT+STEP
c     READ(*,*) II
210   TIME=TIME+DELTA

C       INTEGRATE EULER

      DO 220 J=1,NC
        if(myid.eq.master)then
      XB(J)=XB(J)+DXB(J)*DELTA
      IF (XB(J).LT.0.) XB(J)=0.0
      IF (XB(J).GT.1.) XB(J)=1.
        end if
      DO 225 N=start,istop
      XM(N,J)=XM(N,J)+DXM(N,J)*DELTA
      X(N,J)=XM(N,J)/M(N)
      IF (X(N,J).GT.1.) X(N,J)=1.
      IF (X(N,J).LT.0.) X(N,J)=0.0
225   CONTINUE
        if(myid.eq.numprocs-1 .or. done.eq.0)then
      XD(J)=XD(J)+DXD(J)*DELTA
      IF(XD(J).LT.0.) XD(J)=0.0
      IF(XD(J).GT.1.) XD(J)=1.
        end if
220   CONTINUE

      DO 215 N=start,istop
      M(N)=M(N)+DM(N)*DELTA

C       CALCULATE NEW LIQUID RATES

      DO 271 J=1,NC
271   XX(J)=X(N,J)
      IF(N.GE.NF)THEN
        CALL HYDRAU(M(N),T(N),XX,L(N),WHR,WLR,DR)
      ELSE
        CALL HYDRAU(M(N),T(N),XX,L(N),WHS,WLS,DS)
      END IF
215   CONTINUE

C       share data between tray & processor
        if(done.gt.1)then
        if(myid.ne.numprocs-1)then
        do 300 K=1,NC
300     buffer(K)=X(istop,K)
```

```
         buffer(NC+1)=P(istop)
         buffer(NC+2)=L(istop)
         buffer(NC+3)=V(istop)
         call MPI_SEND(buffer,NC+3,MPI_REAL,myid+1,0,
     +        MPI_COMM_WORLD,ierr)
         count(1)=count(1)+1
         end if
         if(myid.ne.master)then
         call MPI_RECV(buffer,NC+3,MPI_REAL,myid-1,0,
     +        MPI_COMM_WORLD,stat,ierr)
         do 310 K=1,NC
310      X(start-1,K)=buffer(K)
         P(start-1)=buffer(NC+1)
         L(start-1)=buffer(NC+2)
         V(start-1)=buffer(NC+3)

         do 320 K=1,NC
320      buffer(K)=X(start,K)
         buffer(NC+1)=P(start)
         buffer(NC+2)=L(start)
         buffer(NC+3)=V(start)
         call MPI_SEND(buffer,NC+3,MPI_REAL,myid-1,1,
     +        MPI_COMM_WORLD,ierr)
         count(2)=count(2)+1
         end if
         if(myid.ne.numprocs-1)then
         call MPI_RECV(buffer,NC+3,MPI_REAL,myid+1,1,
     +        MPI_COMM_WORLD,stat,ierr)
         do 330 K=1,NC
330      X(istop+1,K)=buffer(K)
         P(istop+1)=buffer(NC+1)
         L(istop+1)=buffer(NC+2)
         V(istop+1)=buffer(NC+3)
         count(3)=count(3)+1
         end if
         end if

         master=0

      GO TO 100
400      call MPI_FINALIZE(ierr)
         ed=MPI_WTIME()
         write(*,*) myid,done,ed,st,ed-st,(count(I),I=1,3)
      STOP
      END

      SUBROUTINE HYDRAU(M,T,X,L,WH,WL,DCOL)
      REAL M,L,MW,MWA
      COMMON NC,MW(5),DENS(5),C1(5),C2(5),C3(5),BPT(5),AVP(5),
     +      BVP(5)
      DIMENSION X(5)
      CALL MWDENS(T,X,MWA,DENSA)
      CONST=183.2*M*MWA/(DENSA*DCOL*DCOL)-WH/12.
      IF(CONST.LE.0.) GO TO 10
      L=DENSA*WL*999.*((183.2*M*MWA/(DENSA*DCOL*DCOL)-
     +      WH/12.)**1.5)/MWA
      RETURN
10    L=0.
      RETURN
      END

      SUBROUTINE ENTH(T,X,Y,HL,HV)
      COMMON NC,MW(5),DENS(5),C1(5),C2(5),C3(5),BPT(5),AVP(5),
     +      BVP(5)
      DIMENSION X(5),Y(5)
      HL=0.0
      HV=0.0
      DO 1 J=1,NC
      HL=HL+X(J)*C3(J)*T
      HV=HV+Y(J)*(C1(J)+C2(J)*T)
1     CONTINUE
      RETURN
      END

      SUBROUTINE MWDENS(T,X,MWA,DENSA)
      COMMON NC,MW(5),DENS(5),C1(5),C2(5),C3(5),BPT(5),AVP(5),
     +      BVP(5)
      DIMENSION X(5)
      REAL MW,MWA
      DENSA=0.0
      MWA=0.
      DO 1 J=1,NC
      MWA=X(J)*MW(J)+MWA
1     DENSA=X(J)*DENS(J) +DENSA
      T=T+0.0
      RETURN
      END

      SUBROUTINE BUBPT(T,X,Y,P)
      COMMON NC,MW(5),DENS(5),C1(5),C2(5),C3(5),BPT(5),AVP(5),
     +      BVP(5)
      DIMENSION X(5),Y(5),PS(5)
      LOOP=0
10    LOOP=LOOP+1
      IF(LOOP.GT.50) GO TO 30
      SUMY=0.0
      DO 15 J=1,NC
      PS(J)=EXP(BVP(J)+AVP(J)/(T+460.))
      Y(J)=PS(J)*X(J)/P
15    SUMY=SUMY+Y(J)
      IF(ABS(SUMY-1.).LT..00001) RETURN
      F=SUMY*P-P
      FSLOPE=0.
      TSQ=(T+460.)**2
      DO 20 J=1,NC
20    FSLOPE=FSLOPE-AVP(J)*X(J)*PS(J)/TSQ
      T=T-F/FSLOPE
      GO TO 10
30    WRITE(6,21)
21    FORMAT(1X,'TEMP LOOP')
        call MPI_FINALIZE(ierr)
      STOP
      END
```

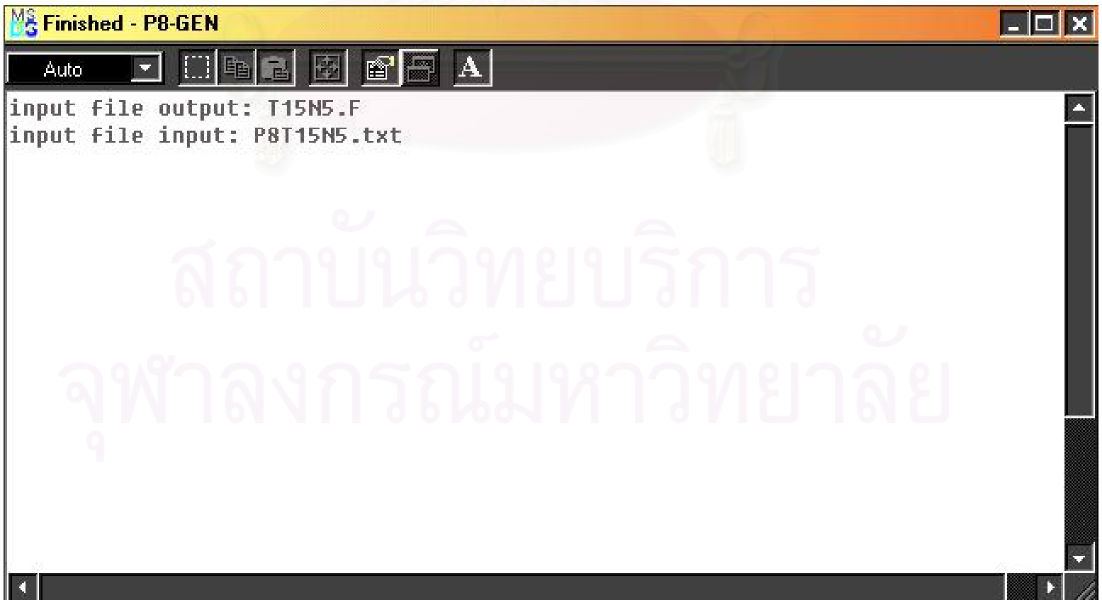
- List of example file 15 trays 5 components (P8T15N5.txt)

```
5
15
15,5,0.75,1.25,72.00,72.00,48.00,48.00,10.00,10.00
'LLK','LK','INTER','HK','HHK'
100.00,90.00,70.00,80.00,80.00
0.200,0.400,0.300,0.300,0.300
0.600,0.600,0.500,0.400,0.400
5*14.7
10.00,90.00,150.00,210.00,360.00
50.00,500.00,150.00,150.00,150.00
30.00,200.00,200.00,300.00,420.00
120.00,800.00,0.5E-01,0.6E+00,0.10E-01,0.30E+00,0.40E-01
120.00,200.00,0.40E+00,0.53E+00,0.20E-01,0.50E-01,0.00E+00
19.70,21.20,5.00,400.0,200.0,0.50
201.58,154.90,132.60,120.20,114.00,108.40,101.20,98.20,96.90,96.20,95
.80,95.50,95.30,95.10,94.90,94.20,77.26
740.10,814.40,892.00,960.10,986.00,320.00,381.90
+  ,409.60,423.70,431.20,435.20,437.50,438.70,439.50,438.60
14
0.000E+00,0.725E-02,0.488E-01,0.836E+00
,0.108E+00,0.999E-11,0.156E-08,0.182E-06,0.133E-04,0.760E-03
,0.112E-02,0.129E-02,0.136E-02,0.140E-02,0.142E-02,0.143E-02
,0.144E-02,0.144E-02,0.145E-02,0.175E-02,0.110E+00,0.286E+00
```

```
,0.457E+00,0.572E+00,0.634E+00,0.818E+00,0.910E+00,0.953E+00
,0.975E+00,0.986E+00,0.992E+00,0.995E+00,0.997E+00,0.998E+00
,0.998E+00,0.240E+00,0.202E+00,0.131E+00,0.803E-01,0.496E-01
,0.866E-01,0.446E-01,0.238E-01,0.128E-01,0.694E-02,0.374E-02
,0.199E-02,0.104E-02,0.519E-03,0.236E-03,0.607E+00,0.473E+00
,0.376E+00,0.314E+00,0.284E+00,0.942E-01,0.440E-01,0.218E-01
,0.110E-01,0.563E-02,0.286E-02,0.145E-02,0.718E-03,0.342E-03
,0.149E-03,0.433E-01,0.393E-01,0.359E-01,0.333E-01,0.325E-01
,0.174E-05,0.776E-06,0.371E-06,0.181E-06,0.893E-07,0.440E-07
,0.216E-07,0.104E-07,0.484E-08,0.205E-08,0.174E-01,0.982E+00
,0.824E-04,0.493E-04,0.659E-09
30.00
50.00
90.00
130.00
300.00
40.00
40.00
60.00
70.00
90.00
10.00
90.00
150.00
210.00
360.00
0.0001
0.0002
0.0001
0.
0.
822.
```

- Demo program

1. Input output filename in Fortran file format (“.f” or “.for”)
2. Input initial condition file.



```
input file output: T15N5.F
input file input: P8T15N5.txt
```

Figure D.4 Demo generate parallel program

VITA

Mr. Sumrid Limvongsuwan graduated high school from Samsen Wittayalai school in 1994 and received a Bachelor Degree in Chemical Engineering from the Department of Chemical Engineering, Faculty of Engineering, King Mongkut Institute of Technology Ladkrabang in 1998. After then he has subsequently studied for a requirement of the Master's Degree in Computational Science at the Department of Mathematics, Faculty of Science, Chulalongkorn University from 1998 till 2002.