УДК 681.3

Балансирование загрузки конвейера процессора с помощью векторов ветвления

Рязанцев А.И., Щербаков Е.В, Щербакова М.Е.
Технологический институт Восточноукраинского национального университета
имени В. Даля (г. Северодонецк)
kvarc@sti.lg.ua

Abstract

Processor' conveyor balancing using branching vectors Ryazantsev A.I., Shcherbakov E.V., Shcherbakova M.E. *Program mechanism of conditional vector processing is proposed for processor' conveyor balancing. Special software tools were developed for data arrays processing. While using it, every element of data vectors may be processed by its own algorithm. Developed tools provide more complete processor's conveyor load in comparison with other tools of technological data arrays processing. Mathematical operations execution using developed tools is faster in more than 2 times than execution of those operations, using similar tools with lesser possibilities.*

Одним из наиболее эффективных средств повышения производительности вычислительных систем является реализация векторных вычислений на конвейерных ЭВМ или на системах с несколькими процессорными элементами. Особенно актуальными для компьютерных систем управления и других систем обработки больших массивов данных являются вопросы эффективного использования конвейерной обработки данных, так как она присутствует во всех современных компьютерах.

Наиболее эффективным средством ускорения неветвящейся обработки последовательностей чисел фиксированного размера (векторов) является использование шаблона классов valarray [1, 2, 3] из библиотеки STL языка программирования C++. К операциям над объектами классов valarray сводятся задачи линейной алгебры, задачи статистической обработки данных, задачи решения систем обыкновенных дифференциальных уравнений и уравнений в частных производных. Главным преимуществом использования valarray является эффективность проведения одинаковых операций сразу над всеми элементами векторов, что является следствием агрессивной оптимизации, выполняемой компиляторами над этими типами данных.

Обработка массивов технологических данных в АСУ ТП

В автоматизированных системах управления технологическими процессами (АСУ ТП) также обрабатываются большие массивы однотипных данных [4]. Но, в отличие от перечисленных выше задач, обработка этих данных производится по ветвящимся алгоритмам в зависимости от условий, задаваемых статически на этапе конфигурирования АСУ ТП, или возникающих на объекте управления в произвольные моменты времени и влияющих на обработку каждого элемента данных технологических массивов в отдельности. Это делает практически невозможным программирование алгоритмов систем управления с использованием шаблонов valarray, а значит, упускается возможность существенного ускорения скорости обработки технологических данных и, как следствие, уменьшения времени реакции системы на внешние события и увеличения надёжности её функционирования.

Одними из важнейших требований, которые выдвигаются пользователями к разработчикам программ АСУ ТП, являются надёжность функционирования системы, минимальные временные характеристики при обработке массивов технологических данных фиксированной размерности и связанные с этим требования как можно меньшего времени отклика системы на внешние события, включая время реакции системы на различные запросы обслуживающего персонала [5, 6].

Примерами такой обработки массивов в АСУ ТП могут служить: получение значений технологических параметров с контролируемого объекта, выработка управляющих воздействий на объект управления на основе текущих значений технологических параметров, отображение технологических данных на мнемосхемах, получение и представление на экранах графиков развития (тенденций) ключевых параметров. Основной особенностью перечисленных выше и других алгоритмов обработки массивов данных в системах управления является наличие в них условных операторов и циклов, контролируемых результатами выполнения операций над отдельными элементами этих массивов или же таблицами решений, задаваемых на этапе

проектирования программного обеспечения системы управления. В качестве демонстрации рассмотрим функционирование технологической базы данных пакета КВАРЦ [5].

В технологическую базу данных, хранящуюся в оперативной памяти, в реальном масштабе времени поступает информация, снимаемая с УСО контроллеров, а также значения технологических параметров, вычисляемых в контроллерах. Данные из массивов этой базы данных обрабатываются набором алгоритмов с заданным периодом повторения или при изменении входных значений. На рис. 1 приведено окно мастера со списком технологических блоков (стандартных алгоритмов) базы данных пакета КВАРЦ.

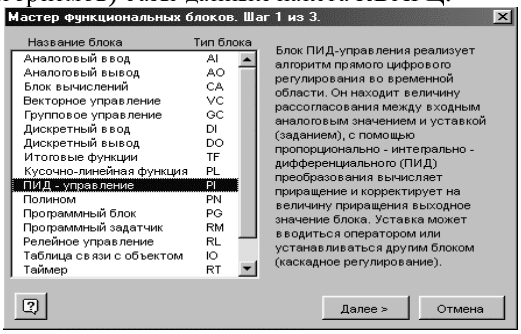
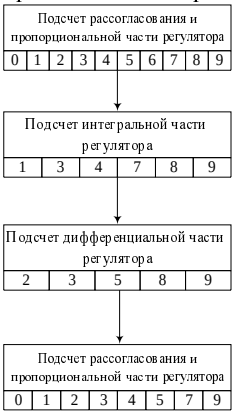


Рисунок 1 - Список алгоритмов технологической базы данных пакета КВАРЦ

Монитор реального масштаба времени, реализованный как COM – надстройка к MS Excel XP, поочерёдно вызывает приведенные в списке блоки на выполнение для обработки соответствующих массивов технологической информации. В частности, выделенный на рисунке блок ПИД – управления формирует сигнал рассогласования между входным сигналом и заданным значением (уставкой) и выполняет пропорционально - интегрально – дифференциальное преобразование этого сигнала. Упрощённая схема работы этого блока представлена на рис. 2.



x0	x1	x2	x3	x4	x5	x6	x7	x8	x9
xh0	xh1	xh2	xh3	xh4	xh5	xh6	xh7	xh8	xh9
xl0	xl1	xl2	xl3	xl4	xl5	xl6	xl7	xl8	xl9
sp0	sp1	sp2	sp3	sp4	sp5	sp6	sp7	sp8	sp9
sh0	sh1	sh2	sh3	sh4	sh5	sh6	sh7	sh8	sh9
sl0	sl1	sl2	sl3	sl4	sl5	sl6	sl7	sl8	sl9
kp0	kp1	kp2	kp3	kp4	kp5	kp6	kp7	kp8	kp9
0.0	ti1	**0.0**	ti3	ti4	**0.0**	**0.0**	ti7	ti8	ti9
0.0	**0.0**	td2	td3	**0.0**	td5	**0.0**	**0.0**	td8	td9
dh0	dh1	dh2	dh3	dh4	dh5	dh6	dh7	dh8	dh9
dl0	dl1	dl2	dl3	dl4	dl5	dl6	dl7	dl8	dl9
ma0	ma1	ma2	ma3	ma4	ma5	ma6	ma7	ma8	ma9
mx0	mx1	mx2	mx3	mx4	mx5	mx6	mx7	mx8	mx9
yh0	yh1	yh2	yh3	yh4	yh5	yh6	yh7	yh8	yh9
yl0	yl1	yl2	yl3	yl4	yl5	yl6	yl7	yl8	yl9
yc0	yc1	yc2	yc3	yc4	yc5	**1**	yc7	**1**	yc9
sy0	sy1	sy2	sy3	sy4	sy5	sy6	sy7	sy8	sy9
y0	y1	y2	y3	y4	y5	y6	y7	y8	y9

Рисунок 2 - Схема вычисления массива управляющих воздействий ПИД – регулятором

В правой части рисунка представлен массив, содержащий данные для работы 10 - ти регуляторов в том виде, в каком они представляются на рабочем листе MS Excel. Элементы массива представлены их символьными обозначениями (sp – уставка регулятора, kp – коэффициент пропорциональности, ti – постоянная времени интегрирования, td – постоянная времени дифференцирования, ус – запрет изменения выхода регулятора) с индексами регуляторов. Числовые значения даны лишь для опорных элементов, по значениям которых осуществляется ветвление обработки. Эти значения выделены жирным шрифтом.

В левой части рисунка представлена укрупнённая схема векторного алгоритма работы ПИД – регулятора над массивом данных. Алгоритм включает четыре последовательно выполняющихся блока. Под прямоугольником с названием блока перечислены индексы столбцов, над элементами которых, в зависимости от значений опорных элементов массива, должна выполняться программа соответствующего блока. Например, интегральная часть регулятора для данных, приведенных на рисунке, не должна вычисляться для столбцов 0, 2, 5 и 6, так как постоянная времени интегрирования ti для этих регуляторов равна нулю. Дифференциальная часть вычисляется только для половины регуляторов, так как для другой половины элементов (с индексами 0, 1, 4, 6 и 7) постоянная времени дифференцирования td в момент вычисления также установлена в нуль. Наконец, итоговое вычисление управляющего воздействия запрещено для регуляторов 6 и 8, находящихся в ручном режиме управления, установкой в 1 настроечной переменной ус.

Как видно из рисунка, при наличии в системе программирования эффективных средств векторной обработки с возможностью её ветвления в зависимости от динамически возникающих условий, вычисление массива регуляторов может быть произведено только за один проход программы регулятора. Точно так же может строиться обработка и для других технологических блоков системы управления. Такая организация вычислений может дать существенную экономию времени обработки за счёт того, что в алгоритмах все вспомогательные инструкции будут выполняться только один раз, а не повторяться для каждого столбца в отдельности. Кроме того, использование шаблона valarray для векторных вычислений в системах управления даст экономию времени за счёт агрессивной оптимизации вычислений компилятором, особенно при использовании для оптимизации вычислений одного векторного расширителя процессора с архитектурой MMX, SSE или SSE2 [7].

Задача оптимальной загрузки конвейера при обработке больших массивов данных в

системах управления технологическими процессами может быть сформулирована следующим образом.

Требуется разработать программные средства, обеспечивающие сбалансированную работу конвейера центрального процессора за счет векторизации, ветвящейся по условиям, и циклическую обработку элементов больших массивов данных систем управления технологическими процессами. Разработанные средства векторного программирования должны встраиваться в современные системы подготовки программ, подобные Microsoft Visual Studio, и должны быть ориентированы на встроенные в компиляторы этих систем средства векторного распараллеливания.

Шаблон valarray с использованием вспомогательных механизмов, таких как срезы, обобщённые срезы, маски или косвенные массивы, позволяет выполнять операции над подмножествами элементов векторов. Но использование этих механизмов затруднено из-за их нетрадиционного синтаксиса и приводит к головоломным алгоритмическим и программным конструкциям. Их практически нельзя использовать для регулярной векторной обработки по ветвящимся алгоритмам.

Важным условием нормальной работы конвейера процессора является отсутствие конфликтов, то есть данные, подаваемые в конвейер, должны быть независимыми [8]. В том случае, когда очередной операнд зависит от результата предыдущей операции, возникают такие периоды работы конвейера, когда он пуст. При скалярной обработке это существенно снижает производительность конвейерных систем.

В конвейерах команд также могут возникать простои, источником которых является зависимость между командами. Такие ситуации возникают при наличии в программах не распараллеливаемых циклов и условных ветвлений.

Одной из причин нарушения ритмичности работы конвейера может служить команда условного перехода в тексте программы [9].

Конвейеризация эффективна только тогда, когда загрузка конвейера близка к полной, а скорость подачи новых операндов соответствует максимальной производительности конвейера. Если происходит задержка, параллельно будет выполняться меньше операций и суммарная производительность снизится. Векторные операции обеспечивают идеальную возможность полной загрузки вычислительного конвейера.

При выполнении векторной команды одна и та же операция применяется ко всем элементам вектора (или чаще всего к соответствующим элементам пары векторов). Для настройки конвейера на выполнение конкретной операции может потребоваться некоторое установочное время, однако затем операнды могут поступать в конвейер с максимальной скоростью, допускаемой возможностями памяти. При этом не возникает пауз ни в связи с выборкой новой команды, ни в связи с определением ветви вычислений при условном переходе.

Программные средства условных векторных вычислений

Способ обработки данных в шаблоне valarray не позволяет ему эффективно обрабатывать массивы, потому что в любой момент обработка данных может быть прервана на несколько тактов командой условного перехода. Чтобы усовершенствовать используемый в нем подход, следовало бы избавиться от этого недостатка – минимизировать число условных переходов при обработке массивов. Для преодоления перечисленных недостатков шаблона valarray был разработан другой шаблон классов – vcarray, включающий в себя практически все возможности шаблона valarray, но обеспечивающий совместно с другим управляющим классом возможности ветвления векторной обработки.

Кроме того, реализация нового шаблона vcarray обеспечивает более быстрое выполнение векторных операций по сравнению с выполнением этих же операций стандартным шаблоном valarray в его реализации фирмой Microsoft в рамках Visual Studio .NET.

Внешне объявление шаблона vcarray выглядит практически так же, как и объявление стандартного библиотечного шаблона valarray:

```
template <class T > class vcarray
{
public:
// конструкторы класса
//vcarray с размером size()==0
vcarray();
//vcarray с n элементами
explicit vcarray(size_t n);
//vcarray с n элементами со значением val
vcarray(const T& val, size_t n);
//vcarray с n элементами со значениями
//pv[0], pv[1],…,pv[n-1]
vcarray(const T* pv, size_t n);
//копия vc
vcarray(const vcarray& vc);
//преобразование valarray в vcarray
vcarray(const valarray& va);
…
// деструктор класса
~vcarray();
```

```
// операторы – члены класса:
// оператор "="
vcarray<T>& operator=(const
vcarray<T>& vc); //копирование vc
valarray<T>& operator=(const T& val);
//присваивание val каждому элементу
// оператор "[]"
T operator[](size_t) const;
T& operator[](size_t);
// унарные операторы
vcarray operator- () const;
// -vc[i] для каждого элемента
vcarray operator+ () const;
// +vc[i] для каждого элемента
vcarray operator~ () const;
// ~vc[i] для каждого элемента
vcarray operator! () const;
// !vc[i] для каждого элемента
// вычисляемые присваивания
vcarray<T>& operator*=(const
vcarray<T>& vc);
vcarray<T>& operator*=(const T& val);
// аналогично для: /=, %=, +=, -=,
// ^=,&=, |=,<<= и >>=
// f(vc[i]) для каждого элемента
vcarray<T> apply (T f(T)) const;
vcarray<T> apply (T f(const T&)) const;
…
private:
T* ptr;
size_t lenghth;
};
```

Помимо одноместных и двуместных операторов, являющихся членами шаблона классов vcarray, имеется также полный набор трехместных операторов, которые не являются членами шаблона классов, и определены в ранге функций – помощников шаблона vcarray. Это все логические, арифметические, тригонометрические операции, а также несколько функций, обеспечивающих удобное для пользователя создание массива данных.

Как видно из текста объявления шаблона и связанных с ним операторов, интерфейс шаблона vcarray практически идентичен интерфейсу стандартного шаблона valarray. Реализация конструкторов и деструктора шаблона vcarray во многом подобна их реализации для шаблона valarray.

Хотя синтаксис одноместных, двуместных и трёхместных операторов шаблона vcarray практически совпадает с синтаксисом этих же операторов для valarray, их семантика и реализация существенно отличаются. Чтобы обеспечить выборочную обработку элементов векторов в зависимости от динамически вырабатываемых векторных условий, все арифметические, логические операции и операции сравнения выполняются только для тех элементов векторов, индексы которых перечислены в текущем векторе ветвления. Текущий вектор ветвления всегда размещается на вершине специального управляющего массива, организованного по стековому принципу и предназначенного для хранения векторов ветвления, в общем случае, разной размерности (рис. 3).

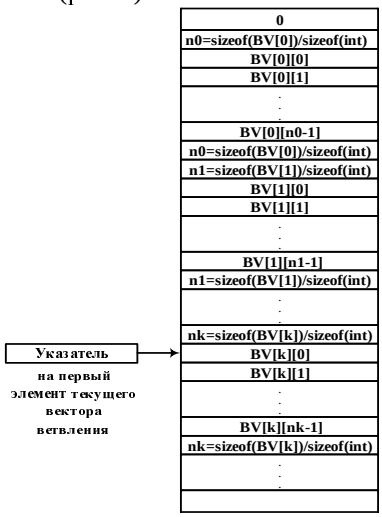


Рисунок 3 - Структура для управления ветвлениями при обработке векторов

В области видимости, в которой находится шаблон vcarray, для управления ходом векторной обработки определятся ряд управляющих функций. Первая из этих функций устанавливает текущую размерность векторной обработки, т. е. размер всех векторов vcarray, участвующих в обработке независимо от типа их элементов. Вторая функция добавляет в управляющий массив вектор ветвления в соответствии с результатами анализа булевских результатов вычисления векторного выражения. Третья функция управления формирует на месте текущего вектора ветвления другой, содержащий все индексы, которых не было в текущем векторе ветвления, но были в предшествующем по уровню. Последняя, четвёртая, функция управления из этой группы удаляет из массива текущий вектор ветвления и делает текущим предшествующий в стеке вектор ветвления.

Шаблон vcarray построен так, что все арифметические операции проводятся с элементами, входящими в текущий вектор ветвления. Изначально в него входят все элементы последовательности. Условные операторы - VcarrayIf(vcarray<bool>), VcarrayElse, операторы повторения - VcarrayWhile(vcarray<bool>) – EndVcarrayWhile, VcarrayDo - VcarrayDoWhile(vcarray<bool>) формируют новый вектор ветвления, и внутри них обрабатываются лишь элементы последовательности, удовлетворяющие условию (входящие в vcarray<bool>). Команда EndVcarrayIf, а также команды завершения циклов удаляют последний вектор ветвления, и происходит возвращение к предыдущему

вектору ветвления, с которым работали до цикла (условного оператора). Таким образом можно строить обработку любой степени вложенности – каждый цикл и условный оператор будет добавлять в специальный стек новый вектор ветвления, а по его завершении этот вектор ветвления будет удаляться. Valarray не позволяет делать вложенную выборочную обработку.

Доказательство универсальности и достаточности для программирования [10] предложенного программного механизма управления векторной обработкой изложено ниже.

Лемма 1. Для шаблона vcarray возможности векторных операторов присваивания с арифметическими и логическими векторными выражениями, отнесенные к одному индексу элементов обрабатываемых векторов, полностью соответствуют их скалярным аналогам – оператору присваивания и выражениям языка C++.

Лемма 2. Для шаблона vcarray ветвление векторной обработки по индивидуальным условиям для каждого из индексов элементов обрабатываемых векторов выполняется точно так же, как если бы элементы векторов обрабатывались скалярными алгоритмами с использованием оператора “if () {} else {}” (рис. 4).

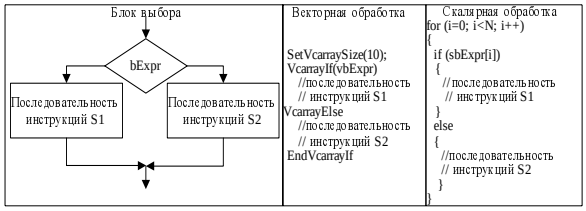


Рисунок 4 - Векторное программирование условия if () {}

Лемма 3. Для шаблона vcarray циклы VcarrayWhile(vcarray<bool>) - EndVcarrayWhile; VcarrayDo-VcarrayDoWhile(vcarray<bool>) выполняются так же, как если бы каждый элемент вектора обрабатывался в отдельном цикле while() или do – while(), где условием выполнения цикла была бы проверка, удовлетворяет ли элемент вектора какому-то логическому условию (рис. 5).

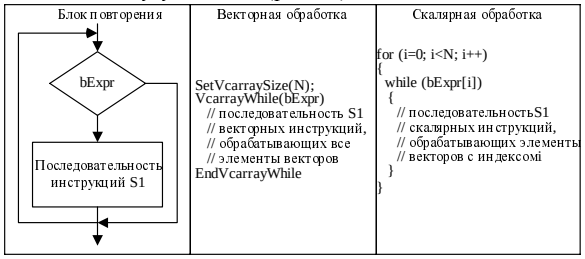


Рисунок 5 - Векторное программирование цикла while() {}

Лемма 4. Приведенных инструкций следования, выбора и повторения достаточно, чтобы выполнить выборочную векторную обработку любой степени вложенности.

Основное утверждение. Шаблона vcarray, структуры для организации ветвлений и набора макросов для ветвления обработки VcarrayIf(vcarray<bool>), VcarrayElse, EndVcarrayIf, макросов повторения VcarrayWhile(vcarray<bool>) -EndVcarrayWhile; VcarrayDo - VcarrayDoWhile(vcarray<bool>) и инструкций следования достаточно для построения ветвящихся программ векторной обработки любой степени сложности так же, как и при скалярной обработке данных.

Так как стандартный шаблон valarray библиотеки STL языка программирования C++ в первую очередь был разработан с целью эффективного использования конвейеров и других средств ускорения вычислений, было проведено сравнительное тестирование разработанного шаблона vcarray с этим стандартным шаблоном на примерах, которые допускают программирование с помощью того и другого шаблона.

Времена выполнения семи тестовых программ для шаблонов классов vcarray и valarray представлены на диаграмме (рис. 6).

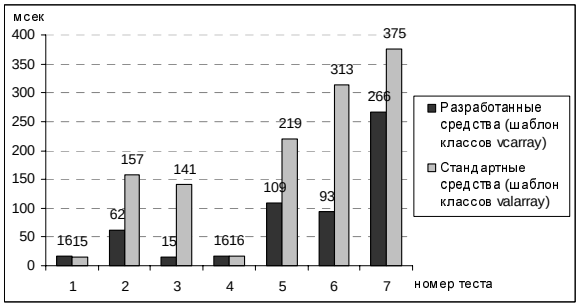


Рисунок 6 - Соотношения времен выполнения различных тестов разработанными средствами и стандартными средствами

Из приведенной диаграммы видно, что шаблон классов vcarray показывает меньшее время выполнения во вложенных условных операторах или циклах, когда число элементов вектора ветвления существенно уменьшается от итерации к итерации, либо когда производится множество операций с элементами после вычисления очередного вектора ветвления.

В описанных семи тестах эффективность использования шаблона vcarray сравнивалась со стандартным шаблоном valarray на фрагментах программ, на которых максимальная загрузка конвейера при скалярной обработке обычно не обеспечивается из-за условий, не позволяющих обрабатывать одинаковым образом все элементы векторов.

На обычных же векторных операциях разработанный шаблон vcarray превосходит по скорости стандартный шаблон valarray в поставке Microsoft Visual Studio.NET версии 7.1 в среднем в 2.3 раза. На рис. 7 представлены

времена выполнения основных векторных операций средствами шаблонов классов valarray и vcarray.

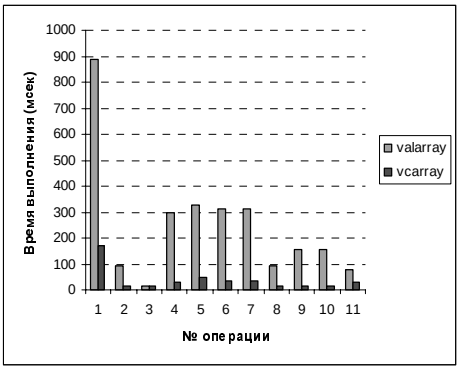


Рисунок 7 - Время выполнения операций различными средствами обработки векторов

Тестирование проводилось на процессоре Intel Pentium 4 с тактовой частотой 2.4 ГГц. Во всех тестах для измерения временных интервалов использовалась API-функция GetTickCount(), измеряющая времена выполнения тестов с точностью до миллисекунды.

Выводы

1. Разработаны метод и программные средства управления условной векторной обработкой массивов данных, ориентированные на более полную загрузку конвейера центрального процессора на ветвящихся участках произвольной сложности и участках с последовательностями зависимых по операндам векторных операций.

2. Доказано, что разработанные средства условных векторных вычислений дают возможность обрабатывать массивы данных по алгоритмам любой степени сложности, в том числе включающие структурные операторы ветвления и циклов, вложенные друг в друга на любую глубину.

3. Разработанные средства сводят условную векторную обработку данных к последовательностям простых циклов, которые распараллеливаются компиляторами Intel 8.0, MS VC++ 7.1, GCC, Borland C++ Builder v.2.3 в последовательности векторных операций над независимыми элементами данных и таким образом практически полностью загружающими конвейер микропроцессора во время выполнения.

4. За счет более полной загрузки конвейера центрального процессора скорость выполнения математических операций при использовании разработанного шаблона классов vcarray превосходит более чем в 2 раза скорость выполнения тех же операций, выполняемых с помощью стандартного шаблона классов valarray из библиотеки STL языка программирования C++, обладающего меньшими возможностями.

5. Новизна разработанных средств заключается в том, что впервые разработаны средства векторной обработки, при использовании которых каждый элемент векторов данных может обрабатываться по своему собственному алгоритму. Операции над отдельными элементами векторов могут пропускаться, ветвиться или повторяться в зависимости от результатов предшествующих операций над элементами данных с этим же индексом.

Литература

1. Черносвитов А.Г. Visual C++ 7: Учебный курс – СПб.; Питер, 2001 – 528 с.

2. Леен Аммерааль STL для программистов на C++ - М.: ДМК, 1999. – 240с.

3.Плаугер П., Степанов А., Ли М., Массер Д. STL – стандартная библиотека шаблонов C++: пер. с англ. – СПб.: БХВ-Петербург, 2004. – 656 с.

4. Елисеев В.В., Ларгин В.А., Пивоваров Г.Ю. Программно – технические комплексы АСУ ТП: Учебн. пособие – К.: Издательско – полиграфический центр „Київський університет”, 2003. – 429 с.

5.Щербаков Е.В., Щербакова М.Е., Охрамович В.К. Автоматизированное проектирование ППО КСУ на базе пакета программ "КВАРЦ". Монография /под ред. д.т.н., проф. А.Г. Руденко – Луганск: Издательство Восточноукраинского национального университета имени В.Даля, 2003.-200 с.

6. Стенцель Й.І. Автоматизація технологічних процесів хімічних виробництв : навч. посібник – К.:ІСДО, 1996. – 260 с.

7. Грегори Эндрюс Р. Основы многопоточного, параллельного и распределённого программирования: пер. с англ. - К.: Диалектика, 2002. – 512 с.

8.Воеводин В.В., Воеводин Вл.В. Параллельные вычисления – СПб.: БХВ-Петербург, 2002. – 608 с.

9. Цилькер Б.Я., Орлов С. А. Организация ЭВМ и систем: Учебник для вузов – СПб.: Питер, 2004. – 668 с.

10. Гради Буч Объектно-ориентированный анализ и проектирование с примерами приложений на C++ - М.: Изд-во Бином, СПб.: Невский диалект, 1999. – 560с.

Поступила в редколлегию 10.03.2009