УДК 519. 687 (575.2) (04)

РАЗРАБОТКА ИНСТРУМЕНТАЛЬНЫХ ПРОГРАММНЫХ СРЕДСТВ СИСТЕМ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ИЗОБРАЖЕНИЙ В КОМПЬЮТЕРНЫХ СЕТЯХ

М.С. Осмонов – канд. техн. наук, доц.
А.Н. Максимов – инженер

Software toolkits for parallel processing of images in computer networks, including libraries of image processing, a plug-in system of external libraries, are described.

В настоящее время цифровая обработка изображений широко используется в различных отраслях науки и техники. Спектр задач, решаемых с использованием методов цифровой обработки изображений, непрерывно растет наряду с увеличением объемов их данных и необходимых вычислений. Применение для их решения дорогостоящих систем обработки изображений ограничивает круг пользователей, вследствие чего все большее применение находят локальные сети компьютеров (ЛВС) в качестве параллельных вычислителей. Конечно, локальные сети не могут обеспечить такой же производительности как суперкомпьютеры, но постоянный рост их производительности делает привлекательным их использование для решения задач, требующих больших объемов вычислений и отличающихся большими объемами данных, к классу которых можно отнести и задачи цифровой обработки изображений.

Организация параллельных вычислений в компьютерных сетях требует эффективных средств параллельного программирования. В настоящее время самой развитой системой параллельного программирования с передачей сообщений является система MPI (Message Passing Interface) [1, 2], представляющая собой стандартизованную библиотеку функций. Использование MPI позволяет обеспечить переносимость разрабатываемых программных продуктов на разные многопроцессорные системы, в том числе на компьютерные сети рабочих станций или ПК. Существует ряд протестированных эффективных реализаций MPI под различные платформы, некоторые из них распространяются для свободного использования (например, MPICH). Но сложность программирования с использованием механизма передачи сообщений является одним из сдерживающих факторов широкого использования параллельной обработки в компьютерных сетях. Одним из возможных путей преодоления этих трудностей является создание инструментальных программных средств, избавляющие программистов от рутинной работы по написанию параллельных программ для решения базовых задач конкретной предметной области с предоставлением средств подключения внешних библиотек.

В данной работе рассматриваются вопросы разработки инструментальных программных средств систем параллельной обработки изображений, позволяющие на их базе прикладным программистам создавать собственные приложения параллельной обработки изображений, а также обеспечивающие возможность подключения внешних библиотек обработки изображений.

1. Инструментальные программные средства

Инструментальные программные средства включают в себя следующие модули (см. рис. 1):

- ядро;
- модуль-обертку библиотеки, реализующей стандарт MPI;
- библиотеки базовых функций обработки изображений;
- систему встраивания внешних библиотек ("plug-in" систему).

1.1. Ядро инструментальной системы

Ядром поддерживаются базовые типы данных, а также структуры и классы, необходимые для реализации алгоритмов обработки изображений. Базовый тип данных, предоставляемых ядром, включает символьные данные, целые числа, числа с плавающей запятой и т.д.

Поскольку алгоритмы обработки изображений в основном оперируют векторами и матрицами, то ядром поддерживаются структуры, описывающие их: *DPIP-CVector* – шаблон для векторных структур различных типов, *DPIP-VectorInfo* – структура, описывающая длину вектора, *DPIP-CMatrix* – шаблон для матричных структур различных типов, *DPIP-MatrixInfo* – структура, описывающая размеры матрицы.

Кроме того, в ядре реализованы типы, необходимые для работы с изображениями: *DPIP-FilterInfo* – структура, описывающая характеристики фильтра, *DPIP-Filter* – структура-обертка фильтра, *DPIP-ImageInfo* – структура, описывающая характеристики изображения или его фрагмента, *DPIP-CImage* – класс-обертка на матрицу изображения или его фрагмента, *DPIP-RGB* – структура, описывающая характеристику точки изображения в RGB формате, *DPIP-RGB-EX* – структура, описывающая характеристику точки изображения в RGB формате, где составляющие цвета представлены числом с плавающей точкой.

Для простоты экспортирования / импортирования функций из внешних библиотек

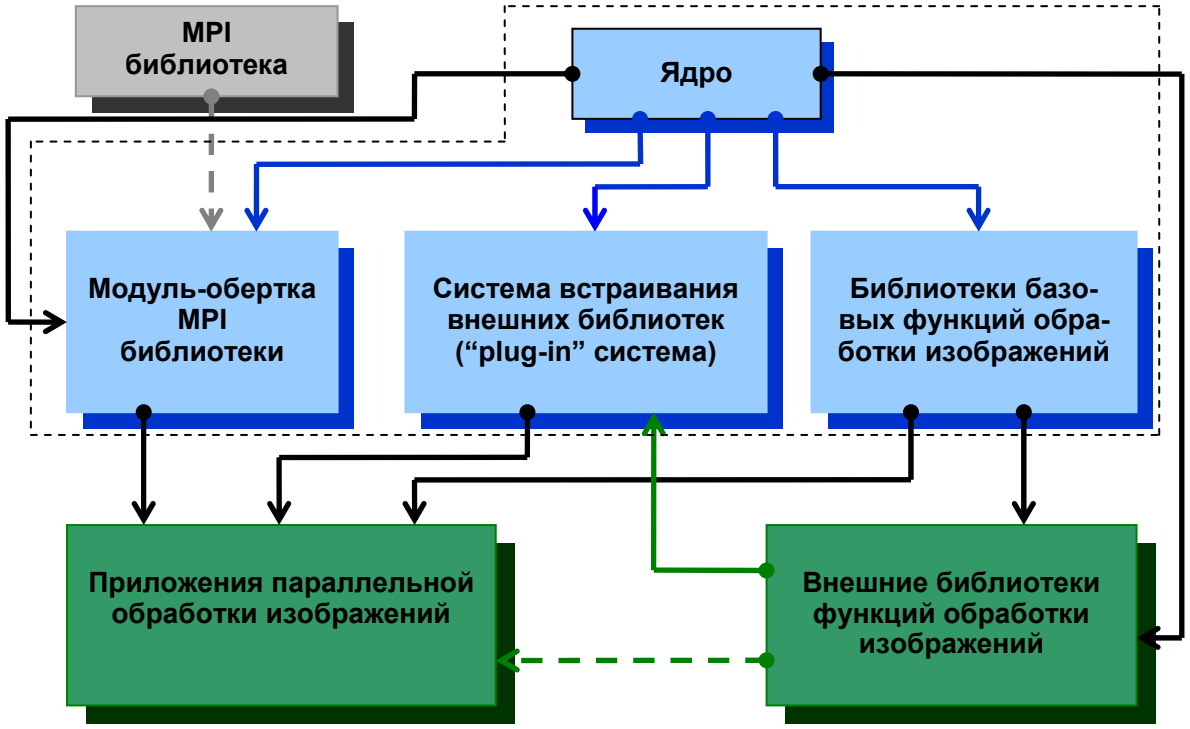

Рис. 1. Структурная схема инструментальных программных средств.

разработан единый интерфейс функций, который определяет правила создания функций прикладными программистами:

DPIP-INT FuncName (DPIP-CFuncArguments pInArgs,*

DPIP-CFuncArguments pOutArgs, DPIP-Error* pError);*

Все функции, экспортируемые из библиотек, должны возвращать код возврата (код удачного завершения, либо код ошибки, в случае ее возникновения во время работы функции). В качестве первого входного аргумента функция должна получать указатель на объект типа *DPIP-CFuncArguments*, который описывает список входных аргументов функции. Вторым входным аргументом должен быть указатель на объект типа *DPIP-CFuncArguments*, который описывает список выходных аргументов функции. Третий входной аргумент функции – это указатель на объект типа *DPIP-Error*, который является списком ошибок, и в случае возникновения ошибки функция передает полное описание возникшей ошибки через данный список.

В силу спецификации общего интерфейса функций ядро реализует три типа: *DPIP-Error* – структура полного описания ошибки, *DPIP-CFuncArguments* – класс-обертка на список аргументов функций, элемент списка имеет тип *DPIP-CFuncArgument*, *DPIP-CFuncArgument* – класс, представляющий "именованное" представление аргумента функции, и описывает работу над парой "логическое имя аргумента" – "значение аргумента".

Таким образом, на вход функции поступают "карты" входных и выходных аргументов, по которым функция определяет, задан ли необходимый ей аргумент или нет. Карты аргументов представляют собой аналогию справочников в базах данных, где логическое имя аргумента может интерпретироваться как его уникальный ключ. Следовательно, функция не может принимать два одноименных входных аргумента или два одноименных выходных аргумента.

1.2. Модуль-обертка MPI библиотеки

Для удобства работы с MPI-библиотекой *модулем-оберткой* предоставляется прикладному программисту набор API-функций. Этот модуль разработан как обертка на библиотеку, реализующую стандарт MPI, и предоставляемые им функции не изменяются от типа используемой библиотеки этого стандарта. Кроме этого, модулем-оберткой реализуются следующие функции:

- инициализация/завершение параллельного приложения;
- предоставление информации о запущенных процессах;
- приведение типов данных ядра инструментальных программных средств к типам данных стандартной библиотеки и, наоборот, приведение типов библиотеки к типам ядра;
- межпроцессорный обмен данных.

1.3. Система встраивания внешних библиотек

Система встраивания внешних библиотек позволяет подключать внешние библиотеки функций (плагины) цифровой обработки изображений, разработанные прикладными программистами на базе инструментальных средств с предоставлением большого набора средств и типов для их создания. При этом интерфейс экспортируемых функций библиотеки должен удовлетворять требованиям единого интерфейса функций, и каждая библиотека функций должна сопровождаться соответствующим конфигурационным файлом.

Конфигурационный файл библиотеки функций содержит три секции:

- Список функций. Начало секции обозначается строкой **[FuncName]**, а конец секции – строкой **[/FuncName]**.
- Список входных аргументов. Начало секции задается строкой **[InArgs],** а конец секции – строкой **[/InArgs].** Секция содержит подсекции, начало и конец которых обрамляются соответственно **[<FuncName>]** и **[/<FuncName>]**, где **<FuncName>** – имя соответствующей функции.
- Список выходных аргументов. Начало секции задается строкой **[OutArgs],** а конец – строкой **[/OutArgs]**. Секция содержит подсекции, начало и конец которых обрамляются соответственно **[<FuncName>]** и **[/<FuncName>]**, где **<FuncName>** – имя соответствующей функции.

В секции "Список функций" перечисляются реальные имена экспортируемых функ-

ций. Они не должны отличаться от имен, используемых в программном коде. В списках входных и выходных аргументов соответствующей функции описываются пары: "логическое имя аргумента" – "тип аргумента".

Для выполнения приложения, реализующего систему встраивания, необходима следующая информация:

- список подключаемых библиотек;
- список путей физического расположения библиотек;
- список конфигурационных файлов, соответствующих каждой подключаемой библиотеке;
- очередь исполнения функций;
- список значений аргументов каждой функции, предписанной к исполнению.

Эта информация хранится в конфигурационном файле, в котором описывается порядок исполнения функций и определяется, какими процессами будут исполняться те или иные функции, задается порядок инициализации аргументов и передачи их значений как между функциями, так и между процессами. Конфигурационный файл позволяет прикладному программисту менять логику приложения, не прибегая к перекомпиляции приложения. При загрузке информации из конфигурационного файла создаются карты:

1) карта уникальных имен функций каждого подключенного плагина;
2) карта уникальных имен аргументов функций;
3) карта значений аргументов каждой функции;
4) карта взаимосвязей функций и их аргументов.

Карта представляет собой список соответствий: "ключ" – "значение", где ключ в списке являлся уникальном идентификатором.

Список функций каждой библиотеки хранится в карте соответствий: логическое имя библиотеки – список имен функций. Информация о логическом имени аргумента и его типе хранится в карте соответствий: *"логическое имя библиотеки": "логическое имя аргумента" – тип аргумента.* В карте соответствий типов аргументов логическое имя библиотеки представляет собой дополнение к имени аргумента для составления уникального ключа карты, поскольку одного логического имени аргумента не достаточно для уникальности из-за возможного совпадения логических имен аргументов разных библиотек.

Для хранения информации об очереди исполнения функций используется список функций, предписанных к исполнению. В списке используется значение: *"логическое имя библиотеки, из которой импортируется функция" + "имя функции" + "номер в очереди*". Схематически записывается так: *"DLLName: FuncName: Order"*. Хотя пара *"логическое имя библиотеки + имя функции"* определяет уникальность имени функции среди всех остальных, требуется прибавления постфиксного номера в очереди, поскольку одна и та же функция может использоваться несколько раз.

Для хранения значений аргументов используется карта соответствий: *"DLLName: FuncName: Order"* – "карта значений аргументов, заданных в конфигурационном файле", где "карта значений аргументов, заданных в конфигурационном файле" является картой соответствий: "логическое имя аргумента" – "значение, указанное в конфигураторе".

Конфигурационный файл состоит из следующих секций:

- *[DLL]...[/DLL]* – список логических имен подключаемых библиотек;
- *[Paths]...[/ Paths]* – список путей физического расположения подключаемых библиотек;
- *[INI]...[INI]* – список путей физического расположения конфигурационных файлов подключаемых библиотек;
- *[FuncQueu]...[/FuncQueu]* – список функций, предписанных на исполнение;
- *[<FuncName>][InArgs]...[/InArgs][OutArgs] ...[/OutArgs][/<FuncName>]* – список значений соответственно входных и выходных аргументов. Здесь <FuncName> – элемент секции [FuncQueu]. Причем число таких секций должно быть равно количеству элементов секции [FuncQueu].

1.4. Библиотеки базовых функций обработки изображений

В состав инструментальных средств входят следующие *библиотеки базовых функций обработки изображений,* реализующие основные операции над изображениями:

- библиотека функций файлового ввода / вывода;

- библиотека функций фрагментации изображения;
- библиотека функций фильтрации изображения;
- библиотека функций поэлементной обработки изображений.

Библиотека функций файлового ввода/вывода обеспечивает ввод/вывод графических файлов формата BMP, а также файлов матриц-сверток, используемых в библиотеке функций фильтрации изображения. Реализованы следующие функции:

- *DPIP-GetFileInfo* – возвращает информацию о файле заданного формата;
- *DPIP-LoadFile* – загружает данные из заданного файла;
- *DPIP-SaveFile* – сохраняет данные в файл.

Библиотека функций фильтрации изображения включает следующие фильтры:

- подавление шумов, сглаживание;
- подчеркивание, выделение контуров;
- "рельефную" фильтрацию (тиснение) и др.

Библиотека функций фильтрации состоит из одной функции: *DPIP-ApplyFurl* (DPIP-CFuncArguments* pInArgs, DPIP-CFuncArguments* pOutArgs, DPIP-Error* pError). Эта функция на вход получает маску (матрицу свертки) и исходное изображение, а на выход выдает результат фильтрации исходного изображения. Варьируя видом матрицы свертки, можно реализовать различные алгоритмы фильтрации исходного изображения.

Библиотека функций фрагментации (разбиения на части) изображений. При распараллеливании по данным исходное изображение разбивается на фрагменты, каждый из которых затем обрабатывается отдельным процессом. Причем при реализации масочных операций необходимо обеспечить перекрытие фрагментов на границах для их корректной обработки. Ширина перекрытия определяется размером матрицы свертки. Библиотекой реализуются следующие функции:
DPIP-GetImageSegmentInfo – выдает информацию о запрашиваемом фрагменте исходного изображения;
DPIP-GetImageSegment – возвращает запрашиваемый фрагмент исходного изображения;
DPIP-SetImageSegment – вставляет заданный фрагмент в исходное изображения и возвращает новое изображение со вставленным фрагментом.

Библиотека функций поэлементной обработки изображений. Имеется группа процедур обработки изображений, где осуществляется так называемая поэлементная обработка. Результат обработки в любой точке изображения зависит только от значения входного изображения в этой же точке. Достоинством таких процедур является их предельная простота. Вместе с тем, многие из них приводят к очевидному субъективному улучшению визуального качества. Данная библиотека реализует следующие функции:

- *DPIP-LinearContrasting* – линейное контрастирование исходного изображения;
- *DPIP-Preparation* – препарирование исходного изображения;
- *DPIP-Solarization* – соляризация исходного изображения.

2. Результаты тестирования

Тестирование проводилось на локальной вычислительной сети, состоящей из шести компьютеров следующей конфигурации: 1) процессоры: Celeron 1.7Hz (4 компьютера), процессор AMD Athlon XP 1.7Hz (1 компьютер), Intel® Pentium® IV 1.7Hz (1 компьютер); 2) чипсет: Intel (4 компьютера), VIA (2 компьютера); 3) оперативная память: 256 Mb (4 компьютера), 512 Mb (1 компьютер), 1Гб (1 компьютер); 4) жесткие диски: частота оборота 7200 оборотов в минуту, шина ATA 133; 5) локальная сеть работает на частоте 100 MHz; все компьютеры являются частью одного домена под управлением Windows 2000 Server, сами компьютеры управляются операционными системами Windows XP Professional.

2.1. Результаты тестирования функционального распараллеливания

В качестве метода тестирования использовался конфигурационный файл, определяющий простейшее функциональное распараллеливание: основной процесс (сервер) выполняет операции загрузки данных и сохранения результатов; остальные процессы используют данные, получаемые от основного процесса, и выполняют операцию свертки над своим фрагментом изображения. Фрагменты изображения передаются основным процессом ос-

тальным непосредственно сразу после инициализации (здесь имеет место распараллеливание по данным), а матрица свертки передается исполняющим процессам только при запросе от них (здесь имеет место функциональное распараллеливание). В тесте использовалось изображение размером 1000x1100 точек (3.1 Мб). Тесты проводились для матриц сверток размеров 3x3, 5x5, 7x7, 9x9, которые представляют собой фильтр выделения контуров (табл. 1). Проводилось пять контрольных запусков и в качестве результата бралось среднее значение.

На рис.2 представлены зависимости изменения времени исполнения приложения от количества используемых компьютеров.

2.2. Результаты тестирования распараллеливания по данным

Этот тест практически ничем не отличается от предыдущего, за исключением того, что матрицы свертки передавались сразу после их инициализации. В качестве тестового фильтра использовался фильтр тиснения (табл. 2).

На рис. 3 представлены зависимости изменения времени исполнения приложения от количества используемых компьютеров.

Таблица 1

Результаты тестирования функционального распараллеливания

Матрица свертки	Количество компьютеров					
	1	2	3	4	5	6
3x3	19.246	68.768	67.538	66.14	66.614	66.136
5x5	38.152	81.97	73.934	70.084	68.924	67.81
7x7	66.574	102.138	85.17	77.102	74.672	72.844
9x9	106.18	121.11	91.486	83.172	80.752	79.018

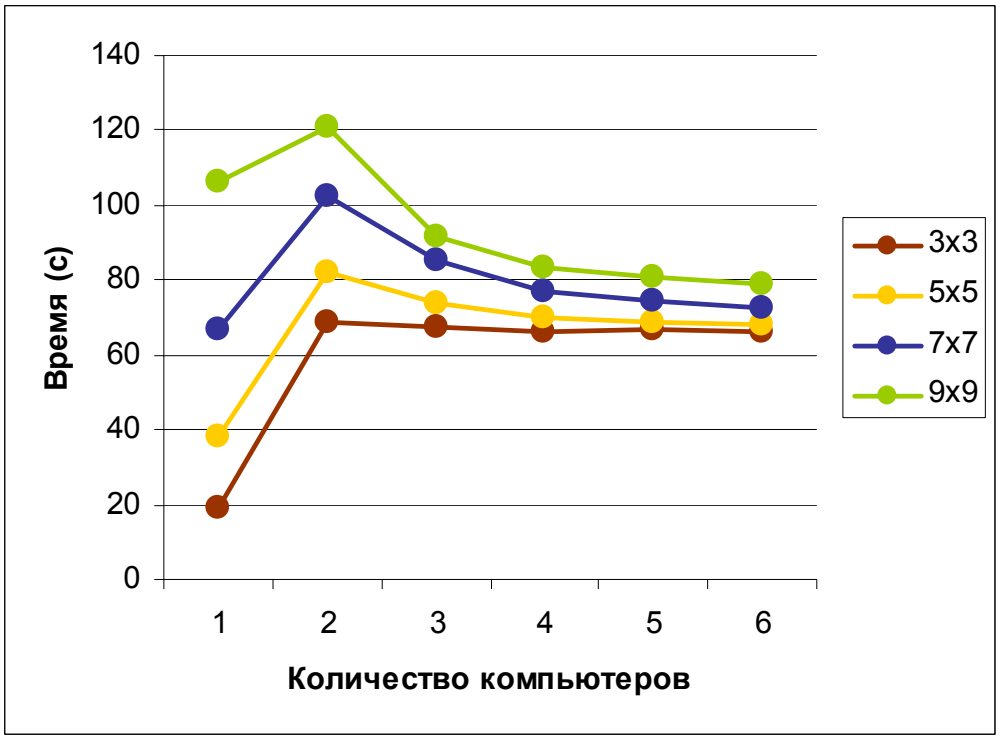

Рис. 2. Результаты тестирования функционального распараллеливания.

Таблица 2

Результаты тестирования распараллеливания по данным

Матрица свертки	Количество компьютеров					
	1	2	3	4	5	6
3x3	19.246	68.768	59.248	61.356	63.862	63.816
5x5	38.152	81.97	59.386	61.234	63.864	64.582
7x7	66.574	102.138	65.582	61.642	64.666	65.73
9x9	106.18	121.11	79.138	66.414	65.262	66.008

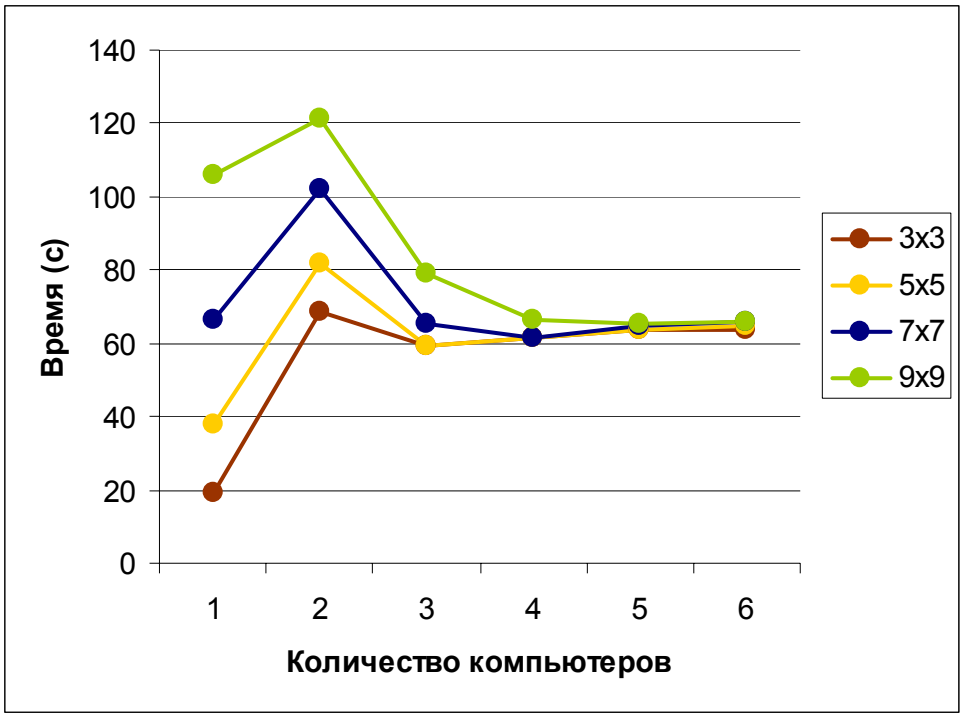

Рис. 3. Результаты тестирования распараллеливания по данным.

Полученные результаты позволяют сделать вывод, что эффективность распараллеливания алгоритмов возрастает с увеличением времени вычислений в узлах сети по отношению к объему передаваемых данных. Чем больше эта разница во времени, тем эффективнее результат распараллеливания.

Литература

1. MPI: The Complete Reference. http://rsusu1.rnd.runnet.ru/ncube/mpi/mpibook/mpi-book.html.
2. *Корнеев В.Д.* Параллельное программирование в MPI. – Новосибирск: Изд-во СО РАН, 2000. – 213 с.