

## Параллельные алгоритмы математических моделей: исследование локальности и применение языка Ada

Л. С. Киргорова, С. И. Киркоров

*Белорусский государственный университет, Беларусь  
Научно-производственное предприятие МедиаСкан, Беларусь*

Article is devoted to working out to modelling of parallel algorithms and checks of their conformity to mathematical model. On an example of coding of a video information, necessity of creation of the stand for testing and application of Ada language is shown. This approach includes research of locality of data in algorithms and aspects of designing of system Xilinx, report CCSDS 133.0-B-1. Advantages of Ada language in realisation of hardware-software systems are proved.

### 1. Общая проблема

Актуальность моделирования алгоритмов, реализующих параллельные вычисления в области сжатия данных, объясняется необходимостью их дальнейшего применения в аппаратно-программных или аппаратных системах. Известно много схем аппаратной интерпретации выражений, на которых основаны эти алгоритмы. Решение проблемы осложняется с одной стороны определенными технологическими трудностями в реализации при построении заказной вычислительной системы, ориентированной на эффективное решение класса задач из конкретной прикладной области. С другой стороны - языковыми проблемами и, наконец, сложной логикой самих решений и сложностью эффективного применения параллельных вычислений [1,2]. Особенность аппаратных средств – это возможность параллельного функционирования двух и более модулей системы, причем любой модуль может сам состоять из параллельно работающих частей. Для всех конкретных вычислительных систем именно степень согласованности структуры алгоритмов с архитектурой систем играет самую важную роль в достижении наивысших скоростей вычислений. Также необходимо отметить, что существуют прикладные задачи, например, в области технической защиты информации, где требуется как формальное доказательство правильности алгоритма, так и его фактическая реализация. В таких приложениях, как дистанционное зондирование Земли (ДЗЗ), где параллельно с алгоритмами шифрования и кодирования данных применяются средства сжатия информации, моделирование алгоритмов, реализующих параллельные вычисления возможно с применением пакетов типа MatLab. Одной из причин ограничений на использование такого инструментария является сложность и высокая стоимость обеспечения соответствующего уровня гарантий конечного продукта в области технической защиты информации по всем трем составляющим – конфиденциальность, целостность и доступность. Это следует из существенной разницы между оптимизированным под

прикладную область алгоритмом и его первоначальной математической моделью, реализуемой подобными пакетами. В [3] утверждается, что «каждый численный метод порождает бесконечно большое множество математически эквивалентных алгоритмов и программ за счет математически эквивалентных преобразований. Все они при одних и тех же входных данных дают одни и те же результаты только в тех случаях, когда все операции выполняются точно. Однако на данном множестве имеется огромный разброс вычислительных свойств, таких как общее число выполняемых операций, влияние ошибок округления, число параллельных ветвей вычислений, размер требуемой памяти, сложность коммуникационных связей и многих других. Важно подчеркнуть, что наличие одних хороших свойств у алгоритма или программы не гарантирует, что хорошими также будут и какие-то другие свойства.»

Еще большей степени сложность и стоимость возрастают, когда стоит задача переноса модели алгоритма и ее реализации на другую доверенную платформу. Под доверенной платформой подразумевается собственный вычислитель, например специализированный процессор с установленной на нем другой операционной системой (или без ОС) или программируемые пользователями вентильные матрицы (Field-Programmable Gate Array – FPGA). Для тестирования доверенной платформы и реализуемых на ней параллельных вычислений, как правило, создается стендовое оборудование. Это стендовое оборудование, рассматриваемое как средство измерения параметров осуществлённой реализации, также требует соответствующего уровня гарантий правильности своего функционирования.

Правильным выбором в этом случае было бы использовать инструментальное средство, которое:

1. Само могло бы пройти сертификационные испытания в области технической защиты информации, то есть, как минимум имело открытые для испытателей спецификации, коды программ и так далее.
2. Соответствовало стандарту, строго его выполняло, и существовал специальный стандарт, регламентирующий эти проверки.
3. Было реализовано (или могло быть адаптировано) для целевых платформ.
4. Обеспечивало поддержку многозадачности для моделирования алгоритмов, реализующих параллельные вычисления на уровне языка высокого уровня (этим достигается стабильность семантики, разработчик избавляется от необходимости использования разнородных внешних библиотек или собственных решений для обеспечения многозадачности).
5. Удовлетворяло бы требованиям экономической эффективности. Например, можно положить, чтобы стоимость приобретения и совокупного владения такого инструментария не превышала 1/3 от всей стоимости разработки.

В связи с этим отметим, что встроенная поддержка многозадачности является уникальной и широко известной особенностью языка программирования Ada, которая выгодно отличает его от большинства современных языков программирования. Следует особо подчеркнуть, что поддержка многозадачности обеспечивается не с помощью каких-либо расширений или внешних библиотек, а с помощью строго стандартизированных средств, которые встроены непосредственно в язык программирования [4-5].

## 2. Постановка задачи и используемые результаты прежних исследований

Рассмотрим некоторую обобщенную постановку [2]. Пусть имеется вычислительный фрагмент (например, выражение), выполняющий содержательную функцию в программе. Реализация этого фрагмента требует выполнения набора каких-то элементарных базовых операций и передач данных от одних операций к другим. Выполнение операции при этом зависит только от готовности данных для этой операции. Для анализа рассмотрим пример вычисления произведения нескольких матриц

$$Y = \prod_{i=1}^n X_i \quad (1)$$

Реализация такого преобразования обычно приводит к последовательной программе с циклами и, возможно, условными операторами. Другой путь – использовать параллельные вычисления (реализованные, по возможности, аппаратно). Тогда для осуществления преобразования данные «прогоняются» через доступные в данной реализации ресурсы – сумматоры, умножители, специализированные подсистемы, элементы локальной памяти.

В русле этой известной общей идеи авторы работы провели исследования по улучшению локальности многомерных алгоритмов с помощью так называемых тайлов (см. [6-18]) при новом ключевом условии использования инструментального средства, удовлетворяющего сформулированным выше требованиям 1-5.

Использование тайлов предполагает уменьшение накладных расходов на обмен данными между процессорами, адаптацию алгоритма к использованию иерархической памяти одного процессора. Тайл используется в алгоритмах обработки растровых изображений, которые предполагают зависимость яркости текущей точки изображения от яркостей точек окружения как по оси X, так и по оси Y. Умножение квадратных матриц тайла изображения размером 8 точек в строке и 8 точек в столбце на матрицу коэффициентов такой же размерности, это один из способов эффективной реализации алгоритма сжатия JPEG (см. [19], в данном случае размер и конфигурация тайла заданы стандартом). Блочные алгоритмы шифрации и кодирования данных используют тайл с целью распараллеливания вычислений и при аппаратной реализации этих алгоритмов. Понятие “TILE” и сопутствующих ему структур определяется и в других графических стандартах, например в JPEG 2000 [20, page 6]. Рекомендуемый размер тайла для большинства применений JPEG 2000 это 1024 по горизонтали на 1024 по вертикали (в данном случае размер тайла может выбираться в зависимости от прикладной области).

Под тайлом (зерном вычислений [8]) понимается множество операций алгоритма, выполняемых как одна единица вычислений, атомарно. При задании зерна вычислений, как правило, решаются следующие задачи [9]:

1. Выбор формы тайла.
2. Выбор размера, числа операций тайла.
3. Выбор последовательности выполнения операций тайла.

(Заметим, порядок выполнения операций влияет на локальность данных, а, следовательно, и на время вычислений. Один путь — повторное разбиение тайлов на тайлы второго уровня. Другой путь — установление приоритетов

выполнения циклов с точки зрения локальности данных и перестановка их в соответствии с приоритетами. Определить оптимальный порядок вычисления циклов, иначе говоря, сравнивать локальность гнезд циклов, можно с помощью так называемого вектора локальности.)

4. Выбор последовательности выполнения тайлов, реализуемых одним процессором, что может влиять на загрузенность процессоров.

(Естественно стремиться в первую очередь выполнять тайлы, от результатов операций которых зависят вычисления других процессоров.)

Для наглядности тайлу сопоставляют множество точек многомерной целочисленной решетки, которое чаще всего является прямоугольным параллелепипедом. Например, если алгоритм может атомарно работать с клетками, на которые разбита прямоугольная матрица, то умножение на такую клетку – тайл, и ему сопоставляются все точки плоскости, координаты которых равны индексам элементов, принадлежащих клетке.

В вычислительных экспериментах настоящей работы в качестве типичной модели алгоритма, подлежащего оптимизации, выбрано умножение квадратных матриц. Рассмотрены различные модификации алгоритма, полученные с помощью перестановки циклов и с помощью зерен вычислений на основе способов улучшения локальности многомерных циклов (перестановка циклов, зерен вычислений). Проведен вычислительный эксперимент по выбору размера (числа операций) тайла, учитывающий параметры: производительность процессора и их количество, пропускную способность каналов доступа к общей памяти процессоров, время инициализации подзадач выполняемых параллельно или псевдопараллельно.

Вне каких-либо технологических ограничений этот подход апробировался в работе [10] (численные эксперименты проводились на кафедре математического моделирования и управления факультета прикладной математики и информатики Белорусского государственного университета).

### **3. Материалы и результаты исследования**

Технические и финансовые условия, в которых находятся авторы, однозначно привели их к инструментальному средству, которое включает: а) – компилятор языка программирования Ada для разработки различных моделей параллельных вычислений и б) – стендовое оборудование собственной сборки.

Критерии выбора инструментария согласно пунктам 1-5 раздела 1 зависят от региона и экономических условий, в которых находится разработчик моделей, от схемы распараллеливания вычислений. Авторы работы использовали инструментарий от фирмы AdaCore (<http://libre.adacore.com>) GNAT GPL 2009 (20090519) для операционной системы Windows XP фирмы Microsoft (доступы также выпуски под другие популярные платформы).

В <http://www.mediascan.by/index.files/Tile9.zip> (файл Tile9.zip) приведен упрощенный исходный текст разработанного программного средства Tile9 (сокращения не влияют на понимание сути) для оценки производительности версий алгоритма на выбираемой аппаратно-программной платформе.

В результате многих серий вычислительных экспериментов на разных компьютерах подтвердилось влияние перестановки циклов и применения тайлинга на скорость работы модельного алгоритма. Оптимальные решения

были разные в зависимости от аппаратной платформы (сравнение проводилось на одноядерных и двухядерных процессорах фирмы Intel). Эксперименты с помощью Tile9 позволили определить максимальные значения количества тайлов, которые можно реализовать в виде задач языка Ada-95 и их оптимальное для быстродействия количество для конкретных сочетаний программных и аппаратных средств.

Использование тайлов значительно повышает скорость расчетов, а в некоторых алгоритмах сжатия изображения является основой для его реализации [11-12]. При разной последовательности циклов в алгоритме оптимальный размер тайла не одинаков для всех случаев. Если рассматривать не модифицированный алгоритм перемножения квадратных матриц, то основная часть алгоритма представляет собой тройной цикл по  $i$ ,  $j$ ,  $k$  с одним оператором S1:

```

declare
  subtype Count is Integer range 1 .. 1000;
begin
  for i in Count loop
    for j in Count loop
      for k in Count loop
        <<S1>>      c(i)(j) := c(i)(j) + a(i)(k) * b(k)(j);
      end loop;
    end loop;
  end loop;
end;
```

Перестановкой циклов можно получать различные модификации данного алгоритма. Всего здесь три цикла, а, значит, существует  $3!$ , то есть шесть, модификаций алгоритма перемножения квадратных матриц.

Для того чтобы характеризовать локальность гнезд циклов и выбрать наилучшую модификацию алгоритма перемножения квадратных матриц из шести возможных, вводится, следуя [6], вектор локальности. Чем выше локальность, тем больший объем вычислений может выполняться отдельно, а, следовательно, может быть сильнее эффект от применения параллельной архитектуры реализующих программ.

Пусть  $a$  – массив размерности большей или равной двум, хранение элементов массива в памяти компьютера осуществляется по строкам. Рассмотрим данные  $a(F(J)), J \in V$ , используемые на  $q$ -ом вхождении массива  $a$  в оператор  $S$ , где  $a$  – некоторый массив с данными, встречающийся в операторе  $S$ ;  $F$  – некоторая функция, которая описывает индексы элементов массива данных  $a$ , относящегося к  $q$ -ому вхождению элементов этого массива в оператор  $S$ ;  $J$  – итерация цикла;  $V$  – область итераций, то есть область изменения параметров гнезда циклов для оператора  $S$ .

Положим  $\lambda = (\lambda_0, \lambda_1)$ , где  $\lambda_0$  – число внутренних циклов, на итерациях которых используется элемент массива  $a$  со всеми фиксированными индексами

(то есть, используется только один элемент массива);  $\lambda_1$  – число внутренних циклов, на итерациях которых используются элементы массива  $a$  с одним последним переменным индексом (если  $a$  – двумерный массив, то используется одна строка). Вектор  $\lambda$  называется *вектором локальности данных*  $a(F(J)), J \in V$ . Первая компонента характеризует временную локальность, вторая – пространственную.

Для оптимизации памяти необходимо, чтобы как можно больше векторов локальности  $\lambda$  были ненулевыми и значения координат были как можно больше.

Рассмотрим на примере алгоритма перемножения квадратных матриц вектор локальности и исследуем, какая из шести модификаций данного алгоритма является наилучшей.

По постановке задачи и алгоритму имеются три массива  $a$ ,  $b$  и  $c$  размерности два. Хранение элементов массива в памяти компьютера осуществляется по строкам, так как при реализации алгоритма на компьютере используется именно такое представление данных в памяти. В данном случае итерация  $J$  представляет собой  $J = (i, j, k)$ . Область итераций  $V = \{(i, j, k) \in Z \mid 0 \leq i, j, k < 1000\}$ .  $J$  и  $V$  определяются для единственного оператора  $S1: c[i][j] = c[i][j] + a[i][k] * [k][j]$ .

Теперь определим вектор локальности  $\lambda = (\lambda_0, \lambda_1)$  для каждого из массивов  $a$ ,  $b$  и  $c$ , то есть определим векторы  $\lambda^a = (\lambda_0^a, \lambda_1^a)$ ,  $\lambda^b = (\lambda_0^b, \lambda_1^b)$  и  $\lambda^c = (\lambda_0^c, \lambda_1^c)$ , где, как уже говорилось,  $\lambda_0^i, i = \{a, b, c\}$  – число внутренних циклов, на итерациях которых используется элемент  $i$ -го массива со всеми фиксированными индексами;  $\lambda_1^i, i = \{a, b, c\}$  – число внутренних циклов, на итерациях которых используются элементы  $i$ -го массива с одной последней переменной строкой, так как все массивы в данной задаче – двумерные. Векторы локальности представлены в таблице 1:

Таб. 1. Векторы локальности для различных модификаций алгоритма перемножения матриц

	ijk	ikj	jki	jik	kji	kij
$\lambda^c$	(1,2)	(0,2)	(0,0)	(1,1)	(0,0)	(0,1)
$\lambda^a$	(0,2)	(1,2)	(0,0)	(0,1)	(0,0)	(1,1)
$\lambda^b$	(0,0)	(0,1)	(1,1)	(0,0)	(1,2)	(0,2)

Как уже говорилось выше, для оптимизации памяти необходимо, чтобы как можно больше векторов локальности  $\lambda$  были ненулевыми и значения координат были бы как можно большими.

Анализируя таблицу 1, несложно увидеть, что наилучшей модификацией (с точки зрения времени реализации) алгоритма является алгоритм с последовательностью циклов  $ikj$ , а наихудшими –  $jki$  и  $kji$ .

Экспериментальные данные были получены на двух компьютерах: одноядерном Intel Pentium 4531, 3000MHz, 1 GB RAM и двухъядерном Intel Core2Duo 6420, 2,13 GHz, 2 GB RAM. В данном случае использовалась матрица размера тысяча на тысячу. В табл. 2 демонстрируются полученные данные:

Таб. 2 – Векторы локальности и экспериментальные данные (время вычислений в сек) для различных модификаций алгоритма перемножения матриц

	ijk	ikj	jki	jik	kji	kij
$\lambda^c$	(1,2)	(0,2)	(0,0)	(1,1)	(0,0)	(0,1)
$\lambda^a$	(0,2)	(1,2)	(0,0)	(0,1)	(0,0)	(1,1)
$\lambda^b$	(0,0)	(0,1)	(1,1)	(0,0)	(1,2)	(0,2)
Pentium 4	25,484	12,063	32,265	21,031	44,547	12,813
Core2Duo	20,641	11,578	23,953	15,797	32,063	12,360

Из табл. 2 видно, что векторы локальности в основном верно предсказывают то, насколько хорошо происходит оптимизация памяти. Исключение: по векторам локальности алгоритм с последовательностью циклов  $jki$  должен работать хуже всего, а на практике получилось, что это цикл  $kji$ . Для объяснения этого факта требуются более точные исследования эффективности использования иерархической памяти.

Можно заключить, что локализация данных сильно влияет на время работы программы, причём самая быстрая и самая медленная модификации алгоритма могут отличаться раза в два-три и больше для задач большего размера, а это довольно существенная разница, когда важна скорость работы.

Для каждой перестановки циклов оптимальный размер тайла может быть свой. Рассмотрим численные результаты эксперимента по оптимизации размера (см. табл. 3) для наиболее распространенной модификации алгоритма – ИК.

В данной таблице для реализаций, трактующих тайлы, как задачи Ады (логические процессоры), для разных размеров тайла представлено абсолютное и относительное время (точнее, «ускорение») перемножения квадратных матриц 1000x1000. «Ускорение» рассматривается относительно реализации тайлинга без использования задач как величина, обратная относительно времени:

$$\text{Ускорение} = \frac{\text{время без подзадач}}{\text{время с подзадачами}} \quad (2)$$

Данные представлены для различных компьютеров и с различными операционными системами.

Таб. 3. Время перемножения квадратных матриц в сек

Размер тайла	20	25	40	100	125	200	250	500
<i>Intel Pentium 4531 3GHz 3.5GB RAM OSWin32XP</i>								
Без подзадач	7,210	7,535	8,196	11,247	11,065	10,644	10,892	11,913
С подзадачами	25,210	18,504	11,583	11,125	11,410	11,636	12,585	12,442
Ускорение	0,29	0,41	0,71	1,01	0,97	0,91	0,87	0,96
<i>Intel Core2CPU 6400 2.13GHz 3.50GB OSWin32XP</i>								
Без подзадач	3,820	3,897	4,059	4,494	4,462	2,429	4,645	7,237
С подзадачами	11,079	6,589	3,644	2,872	2,843	2,847	3,031	5,731
Ускорение	0,34	0,59	1,11	1,56	1,57	0,85	1,53	1,26
<i>Intel Core2CPU 6400 2.13GHz 4.00GB OSWin64Server</i>								
Без подзадач	3,788	3,899	4,053	4,495	4,464	4,468	2,503	7,365
С подзадачами	10,979	9,046	3,581	2,871	2,834	2,858	3,050	5,991
Ускорение	0,35	0,43	1,13	1,57	1,58	1,56	0,82	1,23
<i>Intel Core2CPU 6400 2.13GHz 4.00GB OSWin64XP</i>								
Без подзадач	3,773	3,888	4,043	4,487	4,457	2,372	4,637	7,200
С подзадачами	11,294	8,257	3,865	2,885	2,841	2,850	1,035	5,756
Ускорение	0,33	0,47	1,05	1,56	1,57	0,83	4,48	1,25
<i>Intel Core2CPU 6400 2.13GHz 8.00GB OSWin64XP</i>								
Без подзадач	3,820	6,331	4,069	4,516	4,483	6,710	4,667	7,340
С подзадачами	17,506	8,590	3,923	2,905	5,169	2,864	3,060	5,678
Ускорение	0,22	0,74	1,04	1,55	0,87	2,34	1,52	1,29
<i>Intel Core2Duo E675 2.66GHz 2.00GB OSWin32XP</i>								
Без подзадач	3,031	3,126	3,245	3,599	3,574	3,580	3,716	5,455
С подзадачами	7,909	6,764	2,843	2,295	2,267	2,284	2,416	4,059
Ускорение	0,38	0,46	1,14	1,57	1,58	1,57	1,54	1,34
<i>Intel Pentium 4 CPU 2.60GHz 2.00GB OSWin32XP</i>								
Без подзадач	8,275	9,131	9,607	11,662	9,520	11,510	12,246	13,256
С подзадачами	24,042	22,875	15,984	18,954	18,924	20,153	20,160	20,085
Ускорение	0,34	0,40	0,60	0,61	0,50	0,57	0,61	0,66
<i>Intel Xeon CPU 5130 2.00GHz 4.00GB OSWin64Server</i>								
Без подзадач	4,037	4,160	4,330	4,802	4,767	4,778	4,961	7,284
С подзадачами	14,497	10,014	4,191	3,094	3,045	3,064	3,232	5,356
Ускорение	0,28	0,42	1,03	1,55	1,57	1,56	1,53	1,36
<i>Intel(R) Pentium(R) D CPU 2.80GHz OSLinux</i>								
Без подзадач	7.894	8.136	8.556	10.338	10.283	10.807	11.420	12.343
С подзадачами	11.953	8.763	7.009	7.426	6.773	5.578	5.771	6.424
Ускорение	0,660	0,928	1,221	1,392	1,518	1,938	1,979	1,921

Результаты из табл. 3 подтверждают естественное ожидание, что тайлинг «с задачами» целесообразно проводить только в случае многоядерных процессоров (для мультипроцессорных у авторов данных нет) или процессоров ориентированных на сервера (*Intel Xeon CPU 5130 2.00GHz*). Тогда в большинстве случаев (где размер тайлов не самый маленький) имеем ускорение работы, по крайней мере, в полтора раза. Для одноядерных компьютеров мы видим в одном случае (при размере тайла 100) ускорение 1.01, а в остальных – значительное замедление работы. Неожиданный эффект провала наблюдается при работе с 4-8GB оперативной памяти с процессором *Intel Core2CPU 6400 2.13GHz*.

Для наглядного представления этого эффекта сравним графики ускорения для двух случаев: для одноядерного компьютера Intel Pentium 4531 3GHz CPU, 3.5Gb RAM, OS MS Windows XP (32 bit) и многоядерного IntelCore2 CPU 6400 2.13GHz CPU, 3.50GB RAM, OS MS Windows XP (32 bit), рис 1:

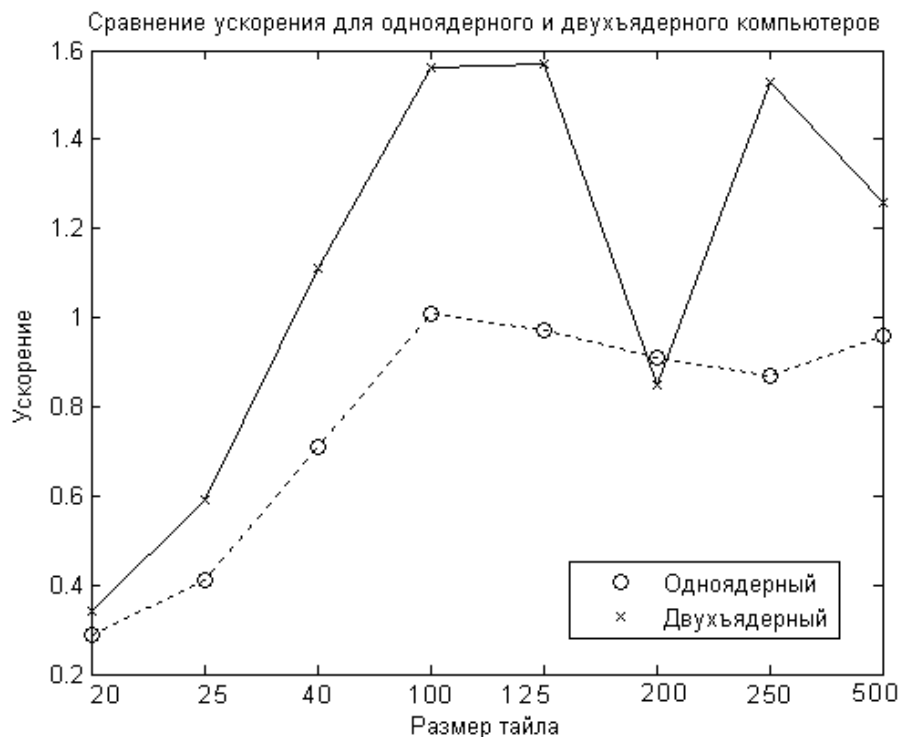


Рис. 1. Сравнение ускорения.

Для других алгоритмов (не ИЖ, а для другой последовательности циклов) программа Tile9 должна быть очевидным образом изменена. Тогда эксперимент с исследованием модификаций этого алгоритма позволит подобрать оптимальный размер тайла и улучшить время вычислений.

На основе описанного исследования данный подход к моделированию алгоритмов был дополнен созданием стендового оборудования на одном из предприятий при разработке программных средств сжатия информации, ориентированных на системы дистанционного зондирования Земли. Конечным результатом этой работы были готовые к внедрению средства для исследования на стендовом оборудовании аппаратных и программных реализаций алгоритмов сжатия и восстановления изображений. Программная часть (исключая, конечно, такие покупные средства как операционные системы) полностью реализована на языке Ada-95. Это программные модели алгоритмов сжатия данных, полностью соответствующие аппаратной реализации на FPGA фирмы Xilinx и имеющие формат данных, аналогичный протоколу CCSDS-133.0-B-1. Программные модели алгоритмов восстановления сжатых данных реализованы также на языке Ada-95. Программные реализации моделей алгоритмов могут использоваться вне стендового оборудования в прикладных системах, реализованных на языке

Ada-95, так как являются платформо-независимыми и имеют высокие показатели производительности и использования вычислительных ресурсов целевой аппаратуры.

Стендовое оборудование реализует «мета-компьютинг» в первоначальном его смысле. Основная цель построения мета-компьютера заключается в оптимальном распределении частей работы по вычислительным системам различной архитектуры и различной мощности на базе высокоскоростной сетевой инфраструктуры [13].

На рис. 2 приведен процесс разработки алгоритмов сжатия данных для целевой аппаратуры дистанционного зондирования Земли.

Программно-аппаратный комплекс, реализующий схему рис. 2, включает:

1. программные средства сжатия данных (ПС СВ);
2. программные средства восстановления данных (ПС ВВ);
3. программные средства стендового оборудования (ПС СО);
4. библиотека наборов данных видеоинформации (БНДВ).

### Процесс разработки алгоритмов

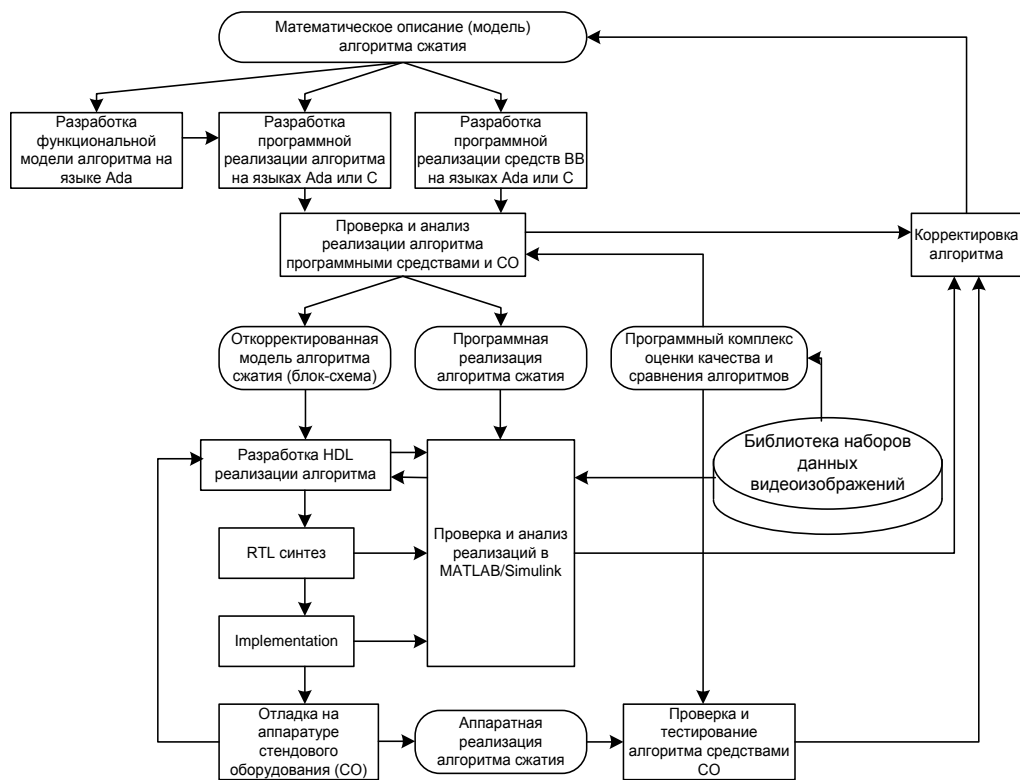


Рис. 2. Процесс разработки алгоритмов сжатия данных для целевой аппаратуры.

Структура программных средств представлена на рис. 3:

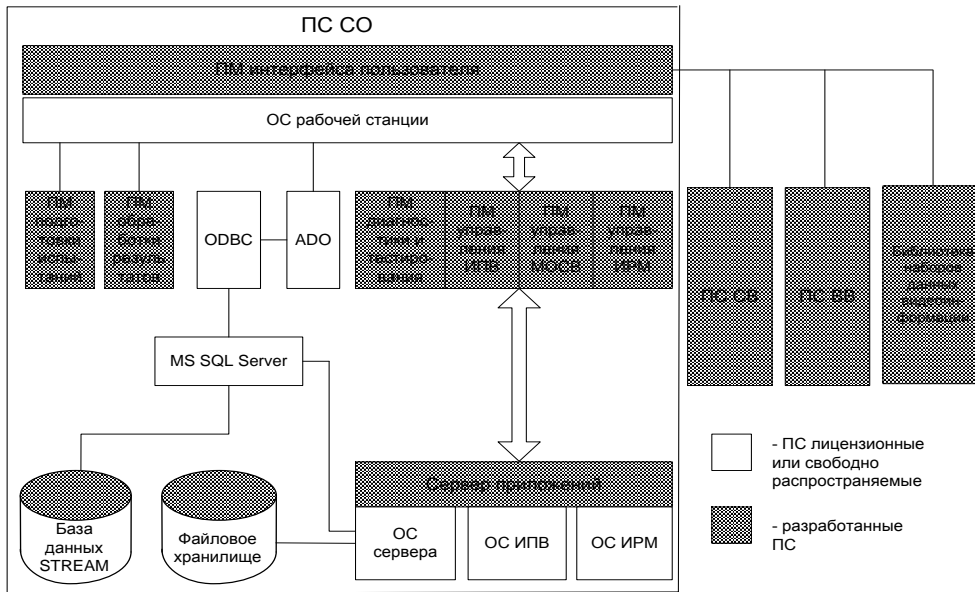


Рис. 3 – Структура программных средств

Аппаратные средства комплекса (стендовое оборудование) предназначены для моделирования, отладки и испытания программных средств сжатия и восстановления данных и подтверждения возможности реализации алгоритмов PC CB в бортовой аппаратуре сжатия на МОСВ.

Структура аппаратных средств стендового оборудования (АС СО) представлена на рис.4.

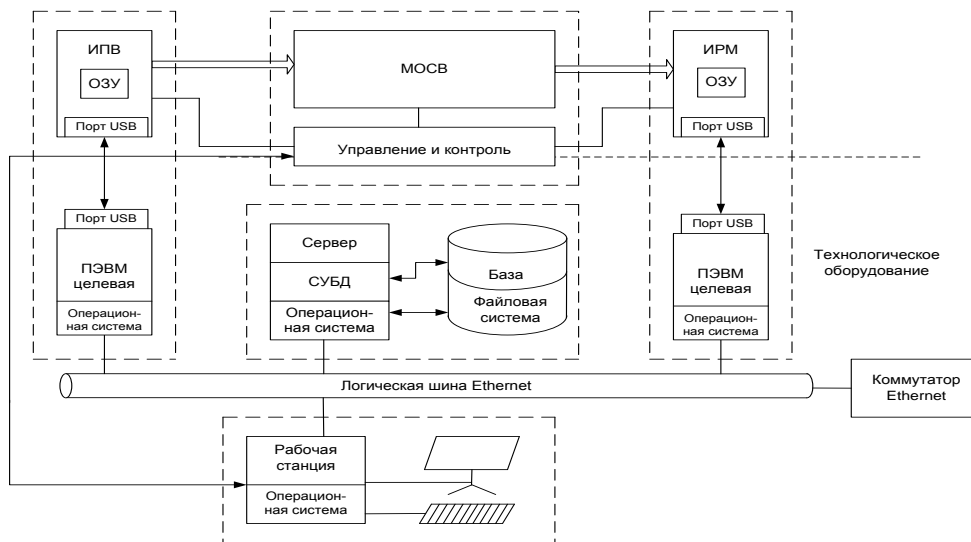


Рис. 4 – Структура АС СО

#### 4. Заключение

Компьютерное моделирование параллельных вычислений, организованных на основе тайлинга и других методов, впервые реализовано так, что эксперименты по исследованию локализации алгоритмов и оптимизации параметров (например, числа тайлов) могут выполняться на любых платформах. В частности, они внедрены в готовом к использованию аппаратно-программном комплексе.

Ключом к решению проблемы полиплатформенности в данной задаче послужило использование языка Ada и свободно распространяемых компиляторов GNAT. Использована программная модель, реализующая основные, причём параллельные, вычисления на языке Ada. Отработанная программная модель была адаптирована под конкретный кристалл FPGA на языке VHDL. Возможна автоматизация этого процесса. Например, используя JRS Research Labs [14] создала Ada-to-VHDL Translator.

Дальнейшие исследования целесообразно вести в направлении объединения моделей, разработанных на языке Ada и на языке VHDL. Язык VHDL используется для моделирования аппаратуры и создания конфигурации для загрузки в FPGA. Такой инструментарий поставляется несколькими фирмами, в том числе и бесплатно, например WEB PACK ICE фирмы Xilinx. Отметим близость синтаксиса описания типов, функций, процедур и управляющих структур на языке Ada и VHDL [15]. Сравнение языков Ada и VHDL можно найти в работе [16].

Эффективная реализация параллельных вычислений в технических аппаратно-программных комплексах – очень сложный по своей природе процесс. С этой точки зрения моделирование и реализация параллельных вычислений на языке Ada позволила в рассматриваемом круге задач достичь идеала – получить программный проект для различных платформ и проект микросхемы, который автоматически трансформируется при помощи кремневого компилятора в конфигурацию FPGA (например, WEB PACK ICE фирмы Xilinx) или в набор фотошаблонов для изготовления кристаллов.

#### ЛИТЕРАТУРА

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002. – 608 с.
2. Элементы параллельного программирования / В.А. Вальковский, В.Е. Котов, А.Г. Марчук, Н.Н. Миренков; Под ред. В.Е. Котова. – М.: Радио и связь, 1983. – 240 с.
3. Воеводин В.В. Вычислительная математика и структура алгоритмов. – М.: Изд-во МГУ, 2006.-112 с.
4. Ada 95 Language Reference Manual ANSI/ISO 8652.1995-std: [Electronic resource]. – <http://www.adapower.com/rm95/index.html>.
5. Гавва А. Е. “Адское” программирование. Ada-95. Компилятор GNAT: [Электрон. ресурс]. – <http://www.ada-ru.org>.
6. Лиходед Н. А. Методы распараллеливания гнезд циклов : курс лекций – Минск : БГУ, 2008. -100 с.

7. Баханович С. В., Соболевский П. И. Отображение алгоритмов на вычислительные системы с распределенной памятью: оптимизация тайлинга для одно- и двумерных топологий // Весці НАН Беларусі. Сер. фіз.-мат. навук. 2006. № 2. С. 106-112.
8. Лиходед Н. А., Пашкович А. К. О выборе зерна вычислений при реализации алгоритмов на параллельных компьютерах с распределенной памятью. // Весці НАН Беларусі. Сер. фіз.-мат. навук. 2008. № 2. С. 121—123
9. Ahmed N., Mateev N., Pingali K. Synthesizing transformations for locality enhancement of imperfectly-nested loop nests. // Proceedings of the International Conference on Supercomputing. – 2000. P. 141–152.
10. Киркорова Л. С. Экспериментальные исследования локальности алгоритмов перемножения матриц, реализованных на двухъядерных компьютерах. – Мн.: БГУ. 2008: [Электрон. ресурс]. – <http://www.mediascan.by/index.files/lkir-doc.pdf>.
11. Алгоритм JPEG // Рабочие материалы. Мн.: НПП МедиаСкан. 2006: [Электрон. ресурс]. – <http://www.mediascan.by/index.files/jpeg-ref.pdf>
12. Information Technology – JPEG 2000 Image Coding System: Core Coding System. International Standard, ISO/IEC 15444-1:2004. 2nd ed. Geneva: ISO, 2004.
13. Что такое Мета-компьютинг? Мета-компьютинг: Распределенные вычисления в Интернет: [Электрон. ресурс]. – <http://parallel.ru/computers/reviews/meta-computing.html>.
14. Robert J. Sheraga ANSI C to Behavioral VHDL Translator, Ada to Behavioral VHDL Translator. The RASSP Digest – Vol. 3. September 1996.
15. Язык описания аппаратуры цифровых систем – VHDL. Описание языка (ГОСТ Р 50754-95), стандарт – М., 1995 г.
16. Gregory P. Chapelle, Michael L. Lewis. Software Metrics Lead the Way to Better HDL Coding Practices – Integrated System Design Magazine, 2000.