2. **Бессонов А. Ф., Дерюгин Л. Н., Комоцкий В. А.** Оптическое зондирование поверх-
1983.
5. **Бессонов А. Ф., Дерюгин Л. Н., Комоцкий В. А.** Измерение фазовых распределений поверхностных акустических волн методом оптического зондирования с опорной решеткой // Автометрия.— 1982.— № 5.
6. **Бессонов А. Ф., Дерюгин Л. Н., Комоцкий В. А., Котюков М. В.** Анализ взаимодействия световой волны с системой пространственно разнесенных периодических структур при оптическом зондировании ПАВ // Оптика и спектроскопия.— 1984.— Т. 56, вып. 6.
7. **Stegeman G. I.** Optical probing of surface waves and surface wave devices // IEEE Trans. on Sonics and Ultrasonics.— 1976.— V. SU-23, N 1.
8. **Акустические** кристаллы: Справочник/Под ред. М. П. Шаскольской.— М.: Наука, 1982.
9. **Bessonov A. F., Black T. D., Deryugin L. N. e. a.** Theory, experimental realization and applications of SAW optical probing with difractional reference gratings // Proc. of the Internat. Symposium Surface Waves in Solids and Layered Structures.— Novosibirsk, USSR.— 1986.— V. II.— P. 202.

Поступила в редакцию 7 апреля 1986 г.

УДК 681.3 : 519.6

М. А. БЕРЕЗОВСКИЙ, А. Л. МИНКИН
(Москва)

ОПТИМИЗИРУЮЩИЙ ПРЕПРОЦЕССОР ПРОГРАММ НА ФОРТРАНЕ ДЛЯ МАТРИЧНОГО ПРОЦЕССОРА А-12

Опыт численного решения больших задач на ЭВМ со специализированными процессорами параллельной архитектуры [1—3] показывает существенное увеличение трудозатрат на программирование в качестве платы за повышение производительности. По-видимому, дальнейшее расширение использования таких вычислительных комплексов будет в значительной степени определяться наличием эффективных методов и средств автоматизации программирования, в первую очередь компиляторов для языков высокого уровня, способных распараллеливать (или векторизовать) пользовательские программы.

При программировании параллельных ЭВМ различают проблемно ориентированный и машинно-ориентированный подходы к распараллеливанию. При первом подходе выявление присущего задаче параллелизма и учет его в программе возлагаются на пользователя. Во втором подходе распараллеливание производится ЭВМ на стадии компиляции.

Данная работа посвящена созданию средств автоматического распараллеливания в фортрановских программах применительно к матричному процессору А-12 (МП) [4]*.

МП подключается к основной ЭВМ в качестве дополнительного вычислительного средства. Он содержит в себе два арифметических устройства с плавающей запятой: умножитель и сумматор. В силу конвейерности обоих процессирующих устройств наибольшая эффективность работы МП достигается при условии непрерывности потоков данных,

---

* Процессор А-12 архитектурно близок и программно совместим с процессорами АР-120В [3], ЕС 2706, ИЗОТ 2001С [5], «Электроника МТ-70М» [6].

поступающих на вход умножителя и сумматора. При одновременной полной загрузке обоих конвейеров достигается максимальное быстродействие в 12 Мфлоп/с. Наличие нескольких конвейерных процессирующих устройств свидетельствует о присущем МП как пространственном, так и временном параллелизме.

Запоминающие устройства МП включают в себя основную память данных, табличную память, память программ и сверхоперативную память. При поэлементной обработке векторных операндов основная память данных является фактически единственным, достаточно емким хранилищем операндов, промежуточных и окончательного результатов вычислений арифметического оператора. Поэтому для выполнения бинарной операции необходимо извлечь из памяти данных два операнда и записать туда результат, т. е. для вычисления одной компоненты вектора результата требуется три обращения к памяти, на что тратится как минимум три такта работы МП. Таким образом, память данных в МП является наиболее «узким» местом при арифметических операциях над векторами.

Математическое обеспечение МП состоит из компилятора с языка Фортран, математической библиотеки подпрограмм, написанных на Ассемблере МП, Ассемблера, загрузчика и других необходимых для работы с процессором программных средств. Математическая библиотека содержит, в частности, большой набор программ, эффективно реализующих векторные арифметические действия, т. е. поэлементное выполнение одних и тех же вычислений над длинными последовательностями данных.

Основное время работы программы, как правило, занимает вычисление циклов. Поэтому естественно потребовать от компилятора с языка высокого уровня проводить векторизацию циклов в программах пользователей. Однако анализ кода, генерируемого базовым компилятором Фортрана, показал, что он не производит такого рода оптимизации. В результате этого программа, написанная на языке Фортран, работает в 2—10 раз медленнее, чем аналогичная программа с векторизованным вручную циклом, т. е. фактически не используются конвейерные свойства МП. Поэтому для повышения производительности комплекса необходимо нарастить имеющийся компилятор с Фортрана оптимизатором, выполняющим необходимые действия по автоматической векторизации циклов. Обычно такого рода оптимизаторы непосредственно встраиваются в компилятор, однако наличие компилятора с Фортрана и математической библиотеки подпрограмм векторной арифметики позволяет упростить задачу. Процесс векторизации предлагается проводить не на стадии компиляции программы, а в рамках препроцессора к компилятору. Результатом оптимизации является замена в теле цикла обычных фортрановских арифметических операторов операторами обращения к векторным подпрограммам математической библиотеки (см. рис. 2, *в*).

Выбор методов распараллеливания циклов. Как уже отмечалось, МП обладает как пространственным, так и временным параллелизмом. Поэтому естественно попытаться при оптимизации использовать обе эти возможности. Кроме того, при выборе методов учитывались те «узкие» места процессора, которые понижают эффективность его использования.

Вообще говоря, векторизация состоит в переходе от исходной повторяющейся последовательности арифметических действий над скалярами к такой последовательности, при которой каждый оператор тела цикла выполняется сразу для всего диапазона значений индекса векторизованного цикла.

В качестве основы для векторизации был взят «координатный» метод распараллеливания циклов [7], созданный применительно к ЭВМ типа ОКМД (одна команда — множество данных, классификацию см. в [3]). Метод требует выполнения ряда условий внутри тела цикла. Если они выполнены, то из множества вложенных циклов (гнезда) выделяется подмножество, для которого вычисления можно вести в параллельном режиме, т. е. фактически осуществляется переход к векторным опе-

рандам. При выборе этого метода было проведено формальное обоснование его применимости не только для ЭВМ типа ОКМД, но и для конвейерной ЭВМ [8].

Были сделаны следующие модификации алгоритма с учетом специфики МП. В ходе векторизации может потребоваться перестройка тела цикла и, в частности, введение дополнительных арифметических операторов присваивания, осуществляющих, по сути, пересылки данных из одной области памяти в другую [7]. Естественно, что количество таких операторов влияет на общее время выполнения векторизованного цикла. Возникает проблема построения такой последовательности операторов в новом теле цикла, при которой количество дополнительных операторов было бы минимально. Для решения этой проблемы был разработан специальный алгоритм, подробно описанный в [8].

Другая модификация метода касается выбора из гнезда циклов подмножества циклов для векторизации. В исходном алгоритме это подмножество может состоять более чем из одного цикла. Однако в случае МП в качестве векторизуемого следует всякий раз рассматривать лишь один цикл из гнезда. Это обусловлено тем, что МП эффективно работает с векторами, компоненты которых регулярно (с постоянным шагом) расположены в памяти процессора. С учетом этой особенности написаны подпрограммы математической библиотеки. При векторизации более одного цикла параллельные вычисления затрагивают более одного измерения в массивах, что требует переменного шага при размещении элементов в памяти. Это всякий раз приводит к нарушению непрерывности вычислений (остановке конвейеров), и реальный выигрыш от распараллеливания будет фактически сводиться к векторизации самого внутреннего из выбранных циклов.

Помимо перечисленных, был сделан ряд дополнений к исходному методу, не связанных непосредственно со спецификой МП и ускоряющих работу самого метода. Суть этих предложений представлена в [8].

После векторизации цикла производится оптимизация каждого арифметического оператора в нем. На этом этапе делается попытка использовать пространственный параллелизм процессора и смягчить влияние недостаточной скорости выборки из памяти на процесс вычислений. Это достигается путем выделения из арифметического выражения так называемых триадных операций, т. е. таких, в которых производятся две операции с тремя операндами. Пусть требуется вычислить выражение $A+B*C$, где все три операнда — векторы. Вычисление такого выражения эффективно, если промежуточный результат от умножения не отсылается в память, а сразу загружается на сумматор и производится сложение. При этом оба процессирующих устройства работают одновременно. Это, кроме того, дает в ходе вычислений экономию двух обращений к памяти для каждой компоненты вектора результата.

Уменьшения количества обращений к памяти можно достичь выделением триад, содержащих не только разные, но и однотипные операции, например, при вычислении выражения типа $A+B+C$, что также повышает эффективность.

Еще одним достоинством выделения триадных операций является экономия памяти при хранении промежуточных результатов вычислений. Например, если оператор $K=A*B+C*M$ выполнять как последовательность трех бинарных операций, то требуется две дополнительные области памяти для хранения результатов двух умножений. Если же это выражение вычислить как последовательность одной бинарной и одной триадной операций, то потребуется лишь одна дополнительная область памяти.

«Узость» памяти данных МП связана не только с недостаточной скоростью ее работы, но зачастую и с малостью ее размеров, что требует дополнительных пересылок между основной ЭВМ и процессором и увеличивает общее время работы программы. Поэтому следует выбирать такую последовательность выполнения арифметических операций, кото-

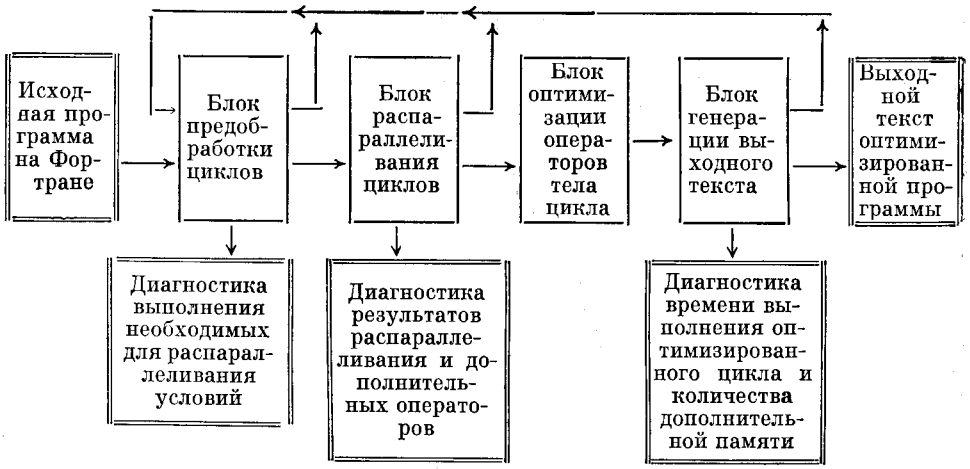

Рис. 1. Общая структура препроцессора

рая требует минимального количества памяти для хранения промежуточных результатов.

Хотя после векторизации речь фактически идет об оптимизации векторных арифметических операторов, тем не менее в них могут содержаться подвыражения, вычисляемые над скалярными операндами. При этом важной оптимизационной задачей является выделение как можно более длинных скалярных подвыражений. Поясним это на примере. Пусть есть арифметический оператор $C = A + B + K$, где C и B — векторы; A и K — скаляры. Если выполнять эти операции в обычном порядке, то потребуются две векторные операции сложения. Однако если вначале выполнить сложение A и K и затем результат прибавить к вектору B, то потребуется одно векторное сложение и одно скалярное, **т. е.** фактически оператор будет выполняться в 2 раза быстрее. При этом скалярные вычисления осуществляются обычными арифметическими операторами, а не путем обращения к подпрограммам.

Для всех перечисленных задач второго этапа оптимизации был разработан специальный алгоритм *. Алгоритм допускает перенастройку на различные допустимые операции, составляющие триады. В данной конкретной реализации с учетом специфики МП допустимыми являются следующие триады: +*, *+, −*, *−, ++, **, +−, −+, −−.

В заключение отметим, что в ходе своей работы указанный алгоритм использует свойства ассоциативности и коммутативности, справедливые для определенных арифметических операций. Но при этом выполняется условие неприкосновенности скобок, расставленных программистом. Это позволяет при необходимости жестко определить порядок вычислений.

Структура препроцессора. Можно выделить четыре логических части, составляющие препроцессор: блоки предобработки циклов, векторизации циклов, оптимизации операторов тела цикла, генерации выходного текста программы (рис. 1). Рассмотрим механизм функционирования каждого из блоков. На вход первого блока подается синтаксически корректная программа на языке Фортран. Затем последовательно просматриваются операторы исходной программы и в каждом операторе выделяются лексемы. Если распознаваемый оператор оказался описателем переменных, то запоминается соответствующая информация. При просмотре линейного участка программы операторы без изменения переносятся в выходной файл. Если обнаружен оператор заголовка цикла, то запоминается информация о количестве итераций и шаге цикла, а для всех последующих операторов ведется проверка выполнения необходи-

* Публикация статьи планируется в 1988 г.

мых условий и перекодировка их во внутреннее представление в виде последовательности лексем. Лексемы определяются парой параметров и запоминаются в специальной таблице, с которой работают все последующие блоки препроцессора.

Для тела цикла проверяются следующие условия: 1) отсутствие диагностика. Если по завершении просмотра тела заданные условия выполнены, то управление передается блоку векторизации, в противном случае возобновляется поиск следующего цикла вплоть до конца программы.

Второй блок препроцессора собственно предназначен для векторизации обрабатываемого гнезда циклов (или одиночного цикла). Вначале на основе информации об индексных выражениях у элементов массивов строятся так называемые векторы направлений, определяющие информационные зависимости между элементами массивов [7]. При их вычислении также выделяются те циклы из гнезда, которые заведомо нельзя векторизовать на основе утверждений из [8]. Далее делается попытка произвести распараллеливание по допустимым циклам. Если выбранный цикл удалось векторизовать, то осуществляется построение оптимального тела цикла на основе алгоритма, предложенного в [8]. В противном случае выбирается следующий кандидат из оставшихся циклов. Если векторизация прошла успешно, то для пользователя выдается информация о распараллеленном цикле и количестве дополнительных операторов, после чего управление передается следующему блоку оптимизации. В случае неудачного распараллеливания печатается соответствующее сообщение и происходит возврат в блок предобработки.

Третий блок препроцессора предназначен для второго этапа оптимизации. Блок последовательно обрабатывает арифметические операторы нового тела цикла. Для каждого оператора строится ассоциативное дерево синтаксического разбора [9]. Алгоритмы этого этапа осуществляют два «обхода дерева»: первый раз снизу вверх (в обратном порядке). а второй раз сверху вниз (в прямом порядке). Поэтому целесообразно иметь два описания дерева — последовательное представление в обратном порядке и представление в прямом порядке [10]. Вначале строится первое указанное представление дерева и определяется тип каждой вершины дерева. Если вершина соответствует операции в исходном выражении, то она может быть одной из трех типов — унарная, бинарная или триадообразующая. Если же вершина соответствует операнду, то она может быть либо скаляром, либо вектором. После построения дерева идет обход его в обратном порядке, начиная с листьев. При этом производятся следующие действия. Если обнаруживается поддерево, соответствующее скалярному подвыражению, то оно удаляется из дерева, а вместо него остается вершина, соответствующая скалярному операнду. Каждой вершине дерева присваивается метка (вес) на основе определенных правил. Достраивается представление дерева в прямом порядке, которое используется при втором обходе дерева.

Во время прямого обхода на основе проведенной разметки ведется соответственно разбиение арифметического выражения на подвыражения (поддеревья), которые уже можно реализовать как обращения к подпрограммам.

В вершинах дерева по возможности выделяется триадная операция, причем так, чтобы минимизировать количество дополнительной памяти для хранения промежуточных результатов.

Результатом работы блока является последовательность унарных, бинарных или триадных операций с идентификацией используемых опе-

а

```
SUBROUTINE E(A, B, C, D, T, F)
REAL A(100, 100), B(100, 100). C(100), D(100), T(100), F(100, 100)
N1 = 3
N2 = 98
DO 1 I = N1, N2
DO 1 J = 1, 100
    A(I, J) = B(I, J) + C(I)*A(I, J) + SIN(D(I))
    C(I) = B(I — 1, J) + R/ALOG10(V)*SL
    B(I, J) = A(I + 1, J) + C(I + 2)/B(I, J)
    T(I — 1) = ABS(T(I + 1))*C(I)*F(I — 2, J)
1 CONTINUE
```

ВВЕДЕН ОДИН ДОПОЛНИТЕЛЬНЫЙ ОПЕРАТОР
ИСПОЛЬЗУЕТСЯ 2 ДОПОЛНИТЕЛЬНЫХ МАССИВА ДЛИНОЙ В (N2 — N 1+1) ЭЛЕМЕНТА
ВРЕМЯ ВЫПОЛНЕНИЯ ОДНОЙ ИТЕРАЦИИ ВЕКТОРИЗОВАННОГО ЦИКЛА (В ТАКТАХ):
МИНИМАЛЬНОЕ — 43, СРЕДНЕЕ — 51, МАКСИМАЛЬНОЕ — 58
ВРЕМЯ ВЫПОЛНЕНИЯ ОДНОЙ ИТЕРАЦИИ ИСХОДНОГО ЦИКЛА (В ТАКТАХ): 259

в

```
SUBROUTINE E(A, B, C, D, T, F)
REAL A(100, 100), B(100, 100), C(100), D(100), T(100), F(100, 100)
REAL TEMP (200)
N1 = 3
N2 = 98
* DO 1 I = N1, N2
  NUM = (N2 — N1 + 1)
  DO 1 J = 1, 100
C ADDITIONAL OPERATOR
C
CALL FVMOV(B(N1, J), 1, TEMP(1), 1, NUM)
C B(I, J) = A(I + 1, J) + C(I + 2)/B(I, J)
C
CALL FVDIV(C(N1+2), 1, B(N1, J), 1, TEMP(NUM + 1), 1, NUM)
CALL FVADD(A(N1+1, J), 1, TEMP(NUM + 1), 1, B(N1, J), 1, NUM)
C       A(I, J) = B(I, J) + C(I)*A(I, J) + SIN(D(I))
C
  CALL FVSIN(D(N1), 1, TEMP(NUM + 1), 1, NUM)
  CALL FVMVA(C(N1), 1, A(N1, J), 1, TEMP(NUM + 1), 1, TEMP(NUM + 1), 1, NUM)
  CALL FVADD(TEMP(NUM + 1), 1, TEMP(1), 1, A(N1, J), 1, NUM)
C       C(I) = B(I — 1, J) + R/ALOG10(V)*SL
C
  S1 = ALOG10(V)
  S2 = R/S1
  S3 = S2*SL
  CALL FVSADD(B(N1—1, J), 1, S1, C(N1), 1, NUM)
C       T(I — 1) = ABS(T(I + 1))*C(I)*F(I — 2, J)
C
  CALL FVABS(T(N1+1), 1, TEMP(1), 1, NUM)
  CALL FVMM(TEMP(1), 1, C(N1), 1, F(N1—2, J), 1, T(N1—1), 1, NUM)
1 CONTINUE
  RETURN
  END
```

Рис. 2. Исходный цикл (*а*), диагностика для пользователя результатов оптимизации (*б*), цикл после оптимизации (*в*)

рандов и результата, а также последовательность скалярных подвыражений, если они обнаружены.

Задача последнего блока — генерация выходного текста программы, содержащего результат проделанной оптимизации. На основе информации, сформированной предыдущими блоками, для каждого исходного арифметического оператора образуется последовательность обращений к подпрограммам математической библиотеки. Кроме того, если были выделены скалярные подвыражения, то так же формируется соответствующая последовательность обычных арифметических операторов.

Для генерации каждого обращения на основе кода операции и типа операндов (скаляр или вектор) подбирается имя соответствующей подпрограммы. Передаваемые подпрограммам аргументы задаются адресом первой компоненты и шагом размещения в памяти. Длина всех векторных операндов одинакова и определяется количеством итераций векторизованного цикла. В зависимости от кода операции выбирается подпрограмма, либо реализующая элементарные арифметические действия, либо соответствующая стандартным математическим функциям, допускаемым Фортраном.

Помимо генерации выходного текста, данный блок выдает сообщения для пользователя о продолжительности обработки одной итерации оптимизированного цикла в тактах процессора. Это позволяет сделать вывод об эффективности проделанной работы. По завершении своей работы последний блок передает управление блоку предобработки для анализа последующих циклов.

Описанная структура препроцессора, состоящего из четырех автономных блоков, имеет ряд преимуществ. В случае применения вместо Фортрана другого языка достаточно заменить лишь первый и последний блоки. Есть возможность расширять алгоритм векторизации, это затронет только второй блок. Третий блок препроцессора может быть применен для оптимизации векторных операторов в параллельных языках типа EVAL [11].

Заключение. На рис. 2 приведены пример тестовой программы, результаты ее обработки препроцессором и выдаваемая при этом пользователю диагностика.

В данном цикле (отмечен звездочкой) в результате оптимизации введены дополнительные оператор (FVMOV) и массив TEMP. В тексте видно использование триадных операций (FVMVA, FVMM), встроенных функций (FVSIN, FVABS), выделение скалярных подвыражений. Скорость работы программы после оптимизации возросла в 5 раз.

В настоящее время над препроцессором ведется работа в двух направлениях: анализ причин, мешающих проведению оптимизации в определенных циклах, и оценка эффективности оптимизации на представительном наборе тестовых и реальных программ.

Выявление причин, мешающих проведению распараллеливания, позволяет определить те ограничения на тело цикла, которые следует ослабить в первую очередь. В частности, предполагается снять ограничения на отсутствие логических операторов, скалярных переменных в левой части арифметического оператора, предоставить более гибкие возможности по работе с индексными выражениями, позволить использовать не только стандартные процедуры-функции.

Для оценки эффективности оптимизации использовался целый ряд пользовательских программ и специально написанных тестов. Результаты показывают, что для разных циклов скорость выполнения их увеличивается в 2—10 раз по сравнению с исходным вариантом компиляции.

ЛИТЕРАТУРА

1. **Энслоу Ф. Г.** Мультипроцессорные системы и параллельные вычисления.— М.: Мир, 1976.
2. **Sumner F.** Supercomputer systems technology // Pergamon Infotech. Limited. Ser. 10.— 1982.— N 6.

3. **Хокни Р., Джессхоуп К.** Параллельные ЭВМ.— М.: Радио и связь, 1986.
4. **Бродский И. И. и др.** Высокопроизводительный периферийный векторный процессор А-12 // Вопросы кибернетики.— М., 1985.— № 104.
5. **Марков С., Лазаров С.** Специализированная высокопроизводительная вычислительная система // Вычислительная техника социалистических стран.— М., 1986.— № 19.
6. **Дыбой В. А., Косицын В. Г., Лазарев В. О.** Быстродействующий периферийный процессор «Электроника-МТ70М» // Автометрия.— 1985.— № 3.
7. **Lamport L.** The parallel execution of DO loops // Communication of ACM.— 1974.— V. 27, N 2.
8. **Березовский М. А., Минкин А. Л.** Развитие координатного метода распараллеливания DO-циклов // Автометрия.— 1988.— № 1.
9. **Ахо А., Ульман Д.** Теория синтаксического анализа, перевода и компиляции.— М.: Мир, 1978.— Т. 2.
10. **Кнут Д.** Искусство программирования для ЭВМ.— М.: Мир, 1976.— Т. 1.
11. **Мучник В., Шафаренко А.** EVAL — язык для программирования параллельных компьютеров.— Новосибирск, 1985.— (Препринт/АН СССР, Сиб. отд-ние, ИАиЭ; 281).

Поступила в редакцию 22 января 1987 г.

УДК 519.524

Ю. Е. ВОСКОБОЙНИКОВ
(Новосибирск)

РЕШЕНИЕ ОБРАТНЫХ ИЗМЕРИТЕЛЬНЫХ ЗАДАЧ С ЗАДАННЫМИ ТОЧНОСТНЫМИ ХАРАКТЕРИСТИКАМИ

Для многих линейных измерительных преобразователей математической моделью, связывающей входное воздействие $\varphi(y)$ и выходной сигнал $f(x)$, является интегральное соотношение вида

$$\int_{a_\varphi}^{b_\varphi} K(x-y)\,\varphi(y)\,dy = f(x), \qquad (1)$$

где $K(x, y)$ — аппаратная функция, искажающая входное воздействие. Задача восстановления $\varphi(y)$ по зарегистрированным значениям $f(x)$ (т. е. устранение аппаратной функции) называется обратной измерительной задачей и относится к классу некорректно поставленных [1]. Из-за неустойчивости задачи малые погрешности измерения $f(x)$ вызывают большие ошибки восстановления. Для построения устойчивого решения уравнения (1) разработаны эффективные регуляризующие методы и алгоритмы [2, 3], использующие дискретное преобразование Фурье. Однако ряд вопросов, возникающих при практическом использовании этих методов, в настоящее время остается без ответа. Это, прежде всего, относится к характеристикам разрешающей способности регуляризующего алгоритма (РА) и к построению РА с заданными точностными характеристиками. Действительно, экспериментатор может рассматривать процесс измерения и последующую обработку данных как работу комплекса «прибор + алгоритм + ЭВМ», и в этом случае важно знать динамические характеристики всего этого комплекса. Тогда, задаваясь требуемыми характеристиками комплекса «прибор + алгоритм + ЭВМ», экспериментатор выбирает «управляющие» параметры как прибора, так и алгоритма обработки. В общей постановке такой подход рассматривался в работах [4, 5], а также в докладе автора [6]. В данной работе для уравнения (1) вводятся точностные характеристики его решения, позволяющие провести содержательный анализ комплекса «прибор + алгоритм + ЭВМ» и выбрать параметр регуляризации, исходя из требуемых точностных характеристик.