КИЕВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. Т. Г. ШЕВЧЕНКО

На правах рукописи

БОЯН
Елена Трифиливна

ИССЛЕДОВАНИЕ И РАЗРАБОТКА МЕТОДОВ РАСПАРАЛЛЕЛИВАНИЯ ПРОГРАММ

05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов систем и сетей

Автореферат
диссертации на соискание ученой степени
кандидата физико-математических наук

Киев — 1992

884.03

Работа выполнена в Институте математики АН Молдовы.

Научный руководитель:

кандидат физико-математических наук, старший научный сотрудник *Маричук М. Н.*

Официальные оппоненты:

доктор физико-математических наук *Анисимов А. В.*
кандидат физико-математических наук *Касаткина И. В.*

Ведущая организация:

Институт кибернетики им. В. М. Глушкова АН Украины.

Защита состоится « » октября 1992 г. в час. на заседании Специализированного совета К.068.18.10 при Киевском государственном университете по адресу: 252207, г. Киев, пр-т Академика Глушкова, 6, факультет кибернетики КГУ, ауд. 40.

С диссертацией можно ознакомиться в библиотеке КГУ.

Автореферат разослан « » июля 1992 г.

Ученый секретарь Специализированного совета
кандидат физико-математических наук *Бейко И. В.*

ЛННБ України ім.В.Стефаника
00816387 (X)

ЛНБ им. В. Стефаника
АН УРСР

Ав-25.742

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность темы.

Появление многопроцессорных вычислительных систем (МВС) потребовало развития и расширения программного обеспечения и его адаптации к требованиям новой аппаратуры, к существующим и новым языкам программирования, к методам построения программных продуктов. С одной стороны, естественный параллелизм реальных процессов является предпосылкой развития и разработки параллельных алгоритмов, языков их описания. С другой стороны, последовательный характер мышления человека, многолетний опыт использования и разработок последовательных алгоритмов делают предпочтительным автоматическое распараллеливание. Последнее поставило вопрос о необходимости эффективного распараллеливания вычислительного процесса. Методы распараллеливания представляют собой современный подход к решению проблемы использования существующего наследия последовательных программ на МВС.

Суть задачи распараллеливания заключается в выделении в последовательных программах тех компонент, которые могут выполняться параллельно. Большинство существующих задач поддается рапараллеливанию. Одни программы распараллеливаются статически, другие требуют тщательного анализа и отслеживания хода их выпонения в динамике.

Цель работы.

Целью настоящей работы является исследование и разработка эффективных методов распараллеливания последовательных программ. Предлагается весьма простой и приемлемый механизм распараллеливания последовательных програм. Он применяется только на операционном уровне распараллеливания програм на функциональной основе. Метод функционального статического распараллеливания состоит из двух частей :

- преобразование последовательной программы в функциональную, эквивалентную исходной;
- распараллеливание полученной функциональной программы.

На первом шаге последовательная программа преобразуется известными методами в структурированную, затем последняя пере-

водится в функциональную.

На втором шаге используется процедура распознавания и "расщепления" компонент функциональной программы. Эта процедура применяется вначале с учетом заданного отношения порядка к компонентам самого верхнего уровня, затем к составляющим компонентам следующего уровня и так далее, пока не доходим до самого низкого уровня. Для распараллеливания составляющих используются специальные процедуры распараллеливания, определяемые типами компонент функциональной программы.

Методы исследования.

В работе используется теория рекурсивных функций для описания метода функционального распараллеливания последовательных программ. Элементы теории множеств и математической логики применены в доказательстве лемм, теорем, утверждений.

Научная новизна.

Механизм вышеизложенного способа распараллеливания позволяет выявить естественным образом достаточно большое количество параллельных компонент функционального выражения. Выполнение полученной программы не требует средств синхронизации, кроме как для чтения исходных данных из общей памяти, так как в функциональной программе присутствуют повторяющиеся функциональные выражения, что позволяет исключить конкуренцию по памяти в процессе вычисления функциональной программы. Каждая компонента функциональной программы, таким образом, может образовать автономные ветви.

Практическая ценность работы.

Метод функционального статического распараллеливания может быть положен в основу автоматического распараллеливания программ в системах трансляции.

Публикация и апробация.

По теме диссертации опубликованы II печатных работ и сданы в ГосФАП 2 работы.

Структура работы.

Настоящая работа состоит из введения, четырех глав и заключения.

Диссертационная работа содержит 122 страниц машинописного текста, в том числе библиографии - 52 наименования.

СОДЕРЖАНИЕ РАБОТЫ

Во введении обосновывается актуальность темы диссертации, указаны цель и методика исследований, подчеркнута научная новизна и практическое значение полученных результатов. Кратко изложено содержание диссертационной работы.

Первая глава содержит описание и классификацию методов распараллеливания последовательных программ. В ней приводится краткая характеристика существующих методов статического распараллеливания программ.

Во *второй главе* предлагаются алгоритмы преобразования последовательной программы P в функциональную $\mathcal{F}$, распараллеливания $\mathcal{F}$ и доказательство их корректности. Распараллеливание осуществляется функциональным методом распараллеливания. Сформулированы критерии целесообразности создания ветвей для их параллельного выполнения на МВС.

Пусть последовательная программа представлена схемой P, элементы которой построены множеством L. Каждому элементу a из L, названному правилом, однозначно ставится в соответствие функция f_a из множества F. Содержимое F определено множеством L.

Функциональная программа $\mathcal{F}$ есть функция, аргументы которой принадлежат F.

Для правил из L и функции из F определяются множества входных (входов) $In(a) \in M$ и $In(f) \in M$, соответственно, а также множества выходных (выходов) $Out(a) \in M$ и $Out(f) \in M$, где M множество всех ячеек памяти.

Пара $< M, L >$ определяет информационный базис последовательной программы P, а пара $< M, F >$ - функциональной программы $\mathcal{F}$.

Схемой P над информационным базисом $< M, L >$ называется последовательность (конечная или бесконечная): $P = p_0 p_1 \ldots p_n$.

Начальный отрезок $p_0\, p_1 \ldots p_k$ обозначим ${}_kp$ и назовем префиксом схемы P.

ОПРЕДЕЛЕНИЕ I. Функцией $\mathcal{F}$ над информационным базисом $\langle M, F \rangle$ называется последовательность (конечная или бесконечная) :

$$\mathcal{F} = f_k\, (f_{l_{k-1}}\, (\ldots f_0\, (x_1, x_2, \ldots, x_n) \ldots) \; ; \; f \in F.$$

Самый внутренний отрезок уровня k в $\mathcal{F}$ есть f_k (....). Обозначим его ${}_kf$ и назовем отрезком уровня k функции $\mathcal{F}$.

Интерпретацией I_L^P информационного базиса последовательной программы называется тройка : $I_L^P = \{ \Delta, \delta_0, \{ L_a : a \in L \}\}$,

где Δ - множество значений, $\delta_0 : M \to \Delta$ - некоторое начальное состояние памяти; $L_a : \Delta^m \to \Delta^n$, m, n - соответственно, количество входов и выходов для правила $a \in L$.

ОПРЕДЕЛЕНИЕ 2. Интерпретацией $I_F^{\mathcal{F}}$ информационного базиса функциональной программы называется тройка : $I_F^{\mathcal{F}} = \{ \Delta, \delta_0, \{ F_f : f \in F \}\}$,

где Δ и δ_0 - имеют тот же смысл что и для I_L^P,

$F_f : f \in F$ и $F_f : \Delta^{m'} \to \Delta^{n'}$, $\langle f \rangle \neq \perp$;

m' и n' - соответственно, количество входов и выходов для функции $f \in F$, $m' \leq m$, $n' \leq n$;

$\Delta^{m'}$ и $\Delta^{n'}$ - множества значений для m'и n' входов и выходов для функции $f \in F$;

$\perp$ - знак неопределенности.

Множества $\{ \delta_i\, (x) \}_P$ и $\{ \delta_i\, (x) \}_{\mathcal{F}}$ определяют совокупность всех промежуточных состояний ячеек x в процессе выполнения P и $\mathcal{F}$, соответственно.

Определим понятие преобразователя σ над множеством ($M \cup L$) $\sigma : (M \cup L) \to F$.

I. Если $x \in M$, то $\sigma(x) = Ca\,(x) \vee Cv\,(x)$,

где Ca и Cv - это, соответственно, функции взять "адрес" и "содержимое" ячейки $x \in M$.

2. Пусть $a \in L$, $In(a) = (x_1, \ldots, x_n)$, $Out(a) = (y_1, \ldots, y_n)$.

Тогда $\sigma(a) = f_a(x_1,\ldots,x_n)$ для каждого $a \in L$ и $f \in F$. Для $y_l \in Out(a)$ $\sigma(y_l) = \sigma(a) = f_a(x_1,\ldots,x_n)$. Множество всех преобразователей $\sigma : M \cup L \rightarrow F$ обозначим $\sigma(M, L)$.

Преобразователь - историей или σ - историей вхождения p_l в P назовем преобразователь $\sigma(p_l) = \sigma(p(x_1, \ldots, x_n))$,

если $l = I$, $p_l = a$, $(x_1,\ldots,x_n) \in In(a)$.

$$\sigma(p_{m+1}) = \sigma(a(\sigma(p_{l_1}),\ldots,\sigma(p_{l_n}))),$$

если $\sigma(p_1),\ldots, \sigma(p_m)$ определены и $In(a) = \{x_1,\ldots,x_n\}$, где p_{l_j} - последнее вхождение p_l перед p_m, вырабатывающее x_j.

σ- историей всех вхождений p_l в P обозначим $\sigma(p_l)$. Введем понятие σ - истории ячейки $x \in M$ после выполнения конечной последовательности актов $_k p$ схемы P:

$\sigma_k p(x) = x$, если в p_l нет ни одного вхождения, вырабатывающего переменную x, в противном случае

$\sigma_k p(x) = \sigma(p_l)$, где p_l - последнее вхождение в P, вырабатывающее x.

ТЕОРЕМА I. P -последовательная и $\mathcal{F}$- функциональная программы функционально эквивалентны, если : I. $In(P) = In(\mathcal{F})$,

2. $Out(P) = Out(\mathcal{F})$, 3. $I_L^P = I_F^{\mathcal{F}}$,

4. $\{\delta_l(x)\}_{\sigma p(x)} = \{\delta_j(x)\}_{\mathcal{F}}, \forall x \in M, l,j \geq 0$.

где $\sigma(p_l(x))$ есть множество промежуточных состояний ячейки x в процессе преобразования P в $\mathcal{F}$.

Опишем процесс преобразования P в $\mathcal{F}$.

Пусть $M_{\mathcal{F}}$ - множество ячеек памяти, необходимые для $In(FSP) \cup Out(\mathcal{F})$, где $In(\mathcal{F})$ и $Out(\mathcal{F})$ являются соответственно множество входных и выходных данных $\mathcal{F}$. Область всевозможных значений обозначим V_x для $x \in M$ есть V_M.

Разобьем $\mathcal{F}$ на последовательные отрезки таким образом, чтобы они совпали с аргументами $\mathcal{F} = \{f_l\}$, $l = I,\ldots,L$, L - количество аргументов $\mathcal{F}$. f_l может быть либо композицией примитивных функций, либо if-функций, либо функций повторения, либо def-функций, определяемых пользователем, либо композицией вышеуказанных функций.

С каждым f_l связаны множества $In(f_l)$- входных и $Out(f_l)$ - выходных данных ; $f_l: V_{In(f_l)} \rightarrow V_{Out(f_l)}$. Вычисление, произ-

водимое f_l, содержит также функцию извлечения содержимого $Cv(x)$ или адреса $Ca(x)$ ячейки $x \in M$. Обозначим эту функцию $Comp_{In\ (f_l)}$. Чтение входных данных и запись выходных значений в ячейке x определяет отображение типа f_l : $V_x \rightarrow V_x$, при условии, что остальные значения ячеек $M_{\mathcal{F}} \setminus Out(f_l)$, в основном, не меняют свои значения.

$$\hat{f}_l = f_l \circ Comp_{Out\ (f_l)} \star Comp_{M_{\mathcal{F}} \setminus Out\ (f_l)}.$$

$\hat{f}_l$ является расширением f_l в том смысле, что

$$\left.\begin{aligned} & Comp_{Out\ (f_l)} \circ f_l = f_l \circ Comp_{In\ (f_l)} \\ & Comp_{M_{\mathcal{F}} \setminus Out\ (f_l)} \circ f_l = f_l \circ Comp_{M_{\mathcal{F}} \setminus Out\ (f_l)} \end{aligned}\right\} \text{(I)},$$

где $\circ$ и $\star$ являются операциями суперпозиции и композиции функции. Соотношение (I) определяет f_l однозначно. Функция $\mathcal{F}$ есть : $\mathcal{F} = f\ (f_1, \ldots, f_L)$, где f_i (i = I,...,L) имеет вид :

$f_i = f\ (f_{i_1}, f_{i_2}, \ldots, f_{i_n})$ (2) или

$f_i = f\ (f_{i_k}(f_{i_{k-1}}(\ldots f_{i_1}(\ldots) \ldots)))$ (3) .

ОПРЕДЕЛЕНИЕ 3. f_i и f_j информационно независимы, если для $i \neq j$ (i,j = I,...,L) $In\ (f_i) \cap In\ (f_j) = \emptyset$.

Так как f_i и f_j ($i \neq j$) содержат повторяющиеся выражения, то, независимо от входных данных, f_i и f_j информационно независимы . Условие параллельного выполнения функций f_i и f_j ($i \neq j, i, j$ = I,...,L) следующее :

$In(f_i) \cap Out(f_j) \cup In(f_j) \cap Out(f_i) \cup \ldots \cup Out(f_i) \cap Out(f_j) = \emptyset$ (4)

Очевидно, что данное условие выполняется. $Out(f_i) \cap Out(f_j) = \emptyset$ получается из процесса преобразования P в $\mathcal{F}$. Условие :

$$\begin{cases} In\ (f_i) \cap Out\ (f_j) = \emptyset \\ In\ (f_j) \cap Out\ (f_i) = \emptyset \end{cases}$$

вытекает из определения информационной независимости f_i и f_j ($i \neq j$, i, j = I,...,L), где f_i и f_j имеет вид (2).

УТВЕРЖДЕНИЕ I. { f_l } (l = I,...,L) могут выполняться параллельно.

f_l в параллельно выполняемом виде имеют вид:

$f_l = f'_{i_1} \star \ldots f'_{i_i} \star \ldots \star f'_{i_k}$, $l \in L$, (i = I,..., k, $k > 0$),

где $f'_{l_1},\dots, f'_{l_k}$ - это преобразованные к параллельному виду функции $f_{l_1},\dots,f_{l_k}$.

Распараллеливание $\mathcal{F}$ имеет смысл, если

1. $\mathcal{F}_c: V_{In(\mathcal{F})} \to V_{Out(\mathcal{F})}$, 2. $\mathcal{F} \approx \mathcal{F}_c$,
3. $\sum T_{f_l} \geq T_{f_{cl}}$, где $\mathcal{F}_c$ распараллеленная $\mathcal{F}$, $T_{f_l}, T_{f_{cl}}$ - время, необходимое, соответственно, для выполнения f_l и f_{cl}.

УТВЕРЖДЕНИЕ 2. Пусть $Comp_{Out(\mathcal{F})} \circ \mathcal{F}$ вычисляет значения выходных данных. Существует единственное расширение $\mathcal{F}$, удовлетворяющее следующим условиям (A) :

$$\begin{cases} Comp_{Out(\mathcal{F})} \circ \mathcal{F} = Comp_{Out(\mathcal{F})} \\ (f_1 * f_2 * \dots * f_L) = (f_1 * f_2 * \dots * f_L) \circ Comp_{In(\mathcal{F})} \\ Comp_{Out(f_1)} \circ (f_1 * f_2 * \dots * f_L) = f_1 \circ Comp_{In(f_1)} \\ \dots\dots\dots\dots \\ Comp_{Out(f_L)} \circ (f_1 * f_2 * \dots * f_L) = f_1 \circ Comp_{In(f_L)} \\ Comp_{M_{\mathcal{F}} \setminus Out(\mathcal{F})} \circ (f1 * f2 * \dots * fL) = Comp_{M_{\mathcal{F}} \setminus Out(\mathcal{F})} \end{cases}$$

где $In(\mathcal{F}) = In(f_1) \cup In(f_2) \cup \dots \cup In(f_L)$,

$Out(\mathcal{F}) = Out(f_1) \cup Out(f_2) \cup \dots \cup Out(f_L)$.

$M_{\mathcal{F}} = In(\mathcal{F}) \cup Out(\mathcal{F}) \cup \dots \cup InOutdef(\mathcal{F})$,

$InOutdef(\mathcal{F})$ - это множества входных и выходных данных def-функций, определяемые пользователем.

УТВЕРЖДЕНИЕ 3. Параллельное выполнение $f_1, f_2, \dots, f_L$ дает тот-же результат, что и их последовательное выполнение в произвольном порядке.

Утверждения 2 и 3 определяют условия корректного распараллеливания $\mathcal{F}$.

Для более эффективного изложения сути вышерассматриваемого преобразования выделим для наших целей подкласс структурированных последовательных программ, так как любую неструктурированную программу P можно преобразовать в структурированную. В дальнейшем все рассуждения будут относиться к структурированным программам SP.

В третьей главе излагаются проблемы, связанные санализом и разработкой метода функционального распараллеливания и реализацией блока функционального распараллеливания над структурирован-

ными последовательными программами SP. [illegible] в п. 3.2. является доказательство теоремы о функционально эквивалентном преобразовании SP в функционально структурированные программы FSP. Доказательство правильности такого преобразования сводится к доказательству правильности преобразования составляющих SP. Проверка правильности преобразования составляющих SP осуществляется посредством анализа элементов SP и их преобразования в FSP. На входе системы трансляции подается программа P, написанная на языке высокого уровня (например, алголоподобного типа), которая, при необходимости, преобразуется из неструктурированного в структурированный вид SP. SP переводится на функциональный промежуточный язык FSP, затем FSP распараллеливается и генерируется объектный код. Выбор промежуточного языка на функциональной основе обусловлен возможностью "расщепления" функциональной программы на наибольшее количество асинхронных параллельных ветвей, выполнение которых проводится без существенных издержек, связанных со средствами синхронизации. Система трансляции, построенная на функциональной основе, состоит из двух этапов :

- анализ и преобразование SP в FSP, описанные в п. 3.I. и 3.2;
- распараллеливание FSP и генерирование объектного кода.

Определим систему функционального программирования, состоящую из следующих элементов : A - множество атомов, O - множество объектов, F - множество примитивных функций, $\mathcal{F}$ - множество функциональных форм.

Обозначим через A множество вида $a \in \{ B, nil, S, C \}$,
где $B = \{ true, false \}$ - булевские значения,

nil - специальный элемент, используемый в рекурсивных структурах данных,

S - множество символьных и строковых констант,

$C = I \cup R$ - множества целых I и вещественных констант R.

Обозначим через O множество, элементы которого обладают одним из свойств :

а) $\perp$, где $\perp$ - признак " неопределенности ",

в) $A \subseteq O$, то есть все атомы есть объекты,

с) если $x_1,\dots,x_n \in O$ и $\forall i$ $(i := 1,\dots,n)$, $x_i \neq \perp$, то последовательность $x = \{ x_1,\dots,x_n \} \in O$,

в) $x = (x_1,\dots,x_n) = \perp$, если x_i $(i = 1,\dots,n)$, что $x_i = \perp$.

Введем предикатную функцию $p(x)$, вырабатывающует булевские значения *true* и *false*.

ОПРЕДЕЛЕНИЕ 4. $f(x)$ неопределена, если $x = \perp$, или $f(x) = \perp$, т.е., если $x = (x_1,\dots,x_n) = \perp$, или результат $f(x)$ неопределен.

В п.3.1.2. определено множество примитивных функций F.

ОПРЕДЕЛЕНИЕ 5. $f(x) \in \mathcal{F}$, если $x \in O$ или $x \in \mathcal{F}$.

Функция условия : *if* - функция :

$$if\ p(x)\ then\ f_1\ else\ f_2 = \begin{cases} f_1, \text{ если } p(x) = true, \\ f_2, \text{ если } p(x) = false, \\ \perp, \text{ если } p(x) = \perp. \end{cases}$$

ОПРЕДЕЛЕНИЕ 6. Если g и f - функции, то $g \circ f$ является их композицией и определяется следующим образом :

$(g \circ f) : x == g:(f:x)$, или $(g \circ f) : x == g(f(x))$,

где $\circ$ обозначает композицию функций.

Для определения функций повторения используется понятие накапливающих параметров функций. Основная идея их использования состоит в том, чтобы определить вспомогательную функцию с лишним параметром, использующуюся для накопления требуемого результата. Опишем эту функцию для каждой из функций повторения.

I. *for* - функция. *SP* : *for i:=E1 to E2 do S.*

FSP : *for* $f_1(i,E1)$ *to* $f_2(E2)$ *do* $f(x)$, то есть

```
      if LT( E1, E2)
      then if LE(f1(i,E1), f2(E2))
           then while f(LE(i, E2),f2(i,I)) do f (f(x))
           else f*(x) ,
```

где i - параметр *for*-функции.

2. *while* - функция. *SP : while bool do S.*

FSP : while $f(p(x),v(x))$ *do* $f(x)$,

где $f(p(x),v(x))$ - содержит накапливающие параметры *while*-функции. Количество $v(x)$ и его содержимое определяются множеством $In\ (p(x))$ и анализом тела цикла в *SP*. В процессе преобразования выявляются функциональные выражения накапливающих параметров.

3. *repeat* - функция аналогична *while*-функции.

repeat $f(x)$ *until* $p(x)$ == *while* $f(p(x),v(x))$ *do* $f(f(x))$.

Функции повторения имеют вид :

- *while* - функция :

$$while\ f(p(x),v(x))\ do\ f(x) == if\ f(p(x),v(x))\ then\ while\ f(p(x),v(x))\ do\ f(f(x));$$

- *repeat* - функция :

$$repeat\ f(x)\ until\ p(x) == while\ f(p(x),v(x))\ do\ f(f(x));$$

- *for* - функция :

$$for\ f1(i,E1)\ to\ f2(E2)\ do\ f(x)$$

FSP : *if p* (*E1*,*E2*)
then while f(*LE*(*i*,*E2*),*AD*(*i*,I)) *do f* (*f*(*x*))
*else f**(*x*) ,

где *i* - параметр *for*-функции.

ОПРЕДЕЛЕНИЕ 7. Программа, состоящая из композиций вышеопределенных примитивных и функциональных форм, называется функционально структурированной *FSP* .

ОПРЕДЕЛЕНИЕ 8. Интерпретацией *FSP* назовем последовательное сопоставление некоторым элементам памяти значения функциональных выражений, являющихся в обоих случаях отображениями вида :

$$f : X^m \longrightarrow X^n,\ n,m \geq 0.$$

Реализация *FSP* при заданной интерпретации заключается в вычислении функциональных выражениях, определяющих *FSP*. Реализация каждой компоненты определяет шаг выполнения *FSP* .

Функциональными компонентами являются примитивные функции и функциональные формы.

Две программы функционально эквивалентны, если описания действий на данные тождественны - от исходного состояния до их конечного состояния .

ТЕОРЕМА 2. Любую *SP* можно преобразовать в функционально эквивалентную *FSP*.

УТВЕРЖДЕНИЕ 4. При преобразовании инструкции повторения из *SP* в функцию повторения в *FSP* цикл "расщепляется" на $r \leqslant q$ функций повторения, где q - это количество генерируемых выходных переменных операторов, составляющих действие-часть тела цикла в *SP* и зависящих от исходных параметров условия инструкции повторения.

Истинность данного утверждения вытекает из процесса построения и преобразования *SP* в *FSP*.

Теорема функционального структурирования описывает по сути

процедуру преобразования SP в FSP.

ТЕОРЕМА 3. Если компоненты FSP являются композицией примитивных функций и функциональных форм, то правильность такой программы доказывается анализом последовательности функциональных выражений и каждый элемент FSP является композицией следующих форм:

1. f - примитивные функции;
2. *if* $p(x)$ *then* f_1 *else* f_2 ,
3. *while* f $(p(x), v(x))$ *do* $f(x)$,
4. *repeat* $f(x)$ *until* $p(x)$,
5. *for* $f1(i, E1)$ *to* $f2(E2)$ *do* $f(x)$.

Теорема 3 является теоремой о корректности покомпонентного разложения FSP, которая по своей сути представляет собой соответствующую процедуру.

Вышеизложенные теоремы обосновывают алгоритм преобразования SP в FSP.

Задача проверки правильности преобразования SP в FSP состоит в следующем. Необходимо преобразовать SP в FSP таким образом, чтобы в результате преобразования была получена FSP, эквивалентная исходной SP.

ОПРЕДЕЛЕНИЕ 9. Преобразование σ правильно, если $\sigma(SP) = FSP$ и

$$\begin{cases} In(SP) = In(FSP), \\ Out(SP) = Out(FSP). \end{cases}$$

Элементарная SP - это SP, состоящая только из одной из следующих конструкций : последовательности, инструкции выбора или инструкции повторения .

ОПРЕДЕЛЕНИЕ 10. Элементарная FSP - это FSP, состоящая только из одной из следующих функций : примитивная функция, функция условия или функция повторения .

ОПРЕДЕЛЕНИЕ 11. Элементарное преобразование SP в FSP - это преобразование типа $\sigma(p_i) = f_j$, $p_i \in SP$, $f_j \in FSP$, где SP принадлежит множеству структурированных программ,

FSP - множеству функционально структурированных программ;

p_i - это элементарная SP;

f_j - элементарная FSP ($i = 1, \dots, n$, $j = 1, \dots, k$, $k, n \geq 0$).

Преобразование SP в FSP осуществляется на каждом шаге процедуры анализа посредством элементарного преобразования SP в FSP. При этом преобразование составляющих

элементарной программы *SP* выполняется таким образом, чтобы полученная элементарная *FSP* была эквивалентна исходной.

Если *SP* является более сложной программой, то она разбивается на элементарные составные.

Если *SP* есть ациклическая программа, то она представляется конечным объединением конструкций последовательности или инструкции выбора. Правильность преобразования *SP* в *FSP* определяется путем анализа составляющих *SP* и их преобразование в соответствующие *FSP* . Для доказательства правильности преобразования циклических *SP* в *FSP* используется лемма о преобразовании циклических *SP* в рекурсивные *FSP*. Определим предикат : *term* (*FSP*, *SP*) = "*SP* преобразуется в *FSP* для каждого $x \in In(SP)$ и $y \in Out(SP)$", вычисление которого предполагается осуществимым (истинным) для любой *SP*.

ОПРЕДЕЛЕНИЕ 12. *FSP* рекурсивна, если $FSP = f(x) = f(f(x))$.

Задача правильности преобразования циклических *SP* в *FSP* сводится к задаче проверки правильности преобразования ациклических *SP*.

ЛЕММА. Случай 1. (*while* - цикл).

$$(FSP = \sigma\ (\textit{while bool do } S)) \longleftrightarrow$$
$$(\textit{term}\ (FSP, \textit{while bool do } S)\ \&\ FSP =$$
$$\textit{if } f(p(x), v(x))$$
$$\textit{then while } f(p(x), v(x)) \textit{ do } f\ (f(x))$$
$$\textit{else } f^{*}(x))$$

где $\longleftrightarrow$ знак отношения эквивалентности.

Случай 2. (*repeat*- цикл).($FSP = \sigma(\textit{repeat } S \textit{ until bool})$) $\longleftrightarrow$ ($\textit{term}\ (FSP, \textit{repeat } S \textit{ until bool})\ \&\ FSP = \textit{while } f(p(x), v(x)) \textit{ do } f\ (f(x))$)

Случай 3. (*for* - цикл).($FSP = \sigma\ (\textit{for } i := E1 \textit{ to } E2 \textit{ do } S)$) $\longleftrightarrow$ ($\textit{term}\ (FSP, \textit{for } i := E1 \textit{ to } E2 \textit{ do } S)\ \&\ FSP$) =

$$\textit{if } LE\ (E1, E2) \textit{ then while } f(LE(i, E1), \text{ав}(i, I)) \textit{ do } f\ (f(x))$$
$$\textit{else } f^{*}(x).$$

Теорема правильности строит процедуру корректного преобразования *SP* в *FSP* на основе понятий интерпретации и реализации, с учетом также леммы о сведении преобразования итерационных *SP* в рекурсивные *FSP*. Лемма показывает, что все *SP* программы, выраженные через элементарные *SP*, могут быть правильно преобразованы в

элементарные *FSP* посредством методов преобразования *SP*, состоящей из последовательности или инструкции выбора.

ТЕОРЕМА 4. Правильность преобразования любой *SP* в *FSP* определяется следующим условием :

$\sigma(SP) = FSP \longleftrightarrow term\ (\sigma(SP), SP)\ \&\ \{ (x,y)\ |\ C\ (X,Y) \}$,

где *SP*, *FSP* и *C(X,Y)* определяются по следующей таблице :

n	вид конструк	SP	FSP	C(X,Y)
I.	последовательность	$S1 \ldots Sn$	$f(f \ldots f(x) ; \ldots)$	$Y = f(f \ldots f(x) \ldots)$
2.	инструкция условия	*if bool then S*	*if* $p(x)$ *then* $f(x)$ *else* $f^{\star}(x)$	$p(x) \rightarrow Y = f(x)$ $NOT\ p(x) \rightarrow Y = f^{\star}(x)$
		if bool then S1 else S2	*if* $p(x)$ *then* $f_1(x)$ *else* $f_2(x)$	$p(x) \rightarrow Y = f1(x)$ $NOT\ p(x) \rightarrow Y = f_2(x)$, если $Out\ (S1) = Out\ (S2)$
			if $p(x)$ *then* $f_1(x)$ *else* $f_1^{\star}(x)$	$p(x) \rightarrow Y = f_1(x)$ $NOT\ p(x) \rightarrow Y = f_1^{\star}(x)$, если $Out\ (S1) \neq Out\ (S2)$
			if $p(x)$ *then* $f_2^{\star}(x)$ *else* $f_2(x)$	$p(x) \rightarrow Y = f_2^{\star}(x)$ $NOT\ p(x) \rightarrow Y = f_2(x)$, если $Out(S1) \neq Out(S2)$
3.	инструкция выбора	*case i of* *l1: S1;* -.-.-.- *ln: Sn* *end*	*if* $EQ(i,l1)$ *then* $f_1(x)$ *elSe* $f_1^{\star}(x)$ -.-.-.-.-.- *if* $EQ(i,ln)$ *then* $f_n(x)$ *else* $f_n^{\star}(x)$	$EQ(i,l1) \rightarrow Y = f_1(x)$ $NOT\ EQ(i,l1) \rightarrow Y = f_1^{\star}(x)$ -.-.-.-.-.-.-.-.- $EQ(i,ln) \rightarrow Y = f_n(x)$, $NOT\ EQ(i,ln) \rightarrow Y = f_n^{\star}(x)$,

4. инструкции повторения	*while bool* *do S*	*while* $f(p(x),$ $v(x))$ *do* $f(f(x))$	$f(p(x),v(x)) \rightarrow$ $Y = f(f(x))$ *NOT* $f(p(x),v(x)) \rightarrow$ $Y = f\star(x)$
	repeat S *until bool*	*repeat* $f(x)$ *until* $f(p(x)$ $,v(x))$	$Y = f(p(x),v(x) \rightarrow$ $f(f(x))$ *NOT* $f(p(x),v(x))$ $\rightarrow f\star(x)$
	for i:= E1 *to E2* *do S*	*if* $LE(E1,E2)$ *then while* $(LE(i,E2),$ $AD(i,I))$ *do* $f(f(x))$ *else* $f\star(x)$	$LE(E1,E2) \rightarrow$ $Y = (LE(i,E1),$ $AD(i,I)) \rightarrow$ $f(f(x))$ *NOT* $LE(i,E1) \rightarrow f\star(x)$ *NOT* $LE(E1,E2) \rightarrow Y=f\star(x))$

$x \in In(SP)$, $Y \in Out(\sigma(SP))$.

$term(\sigma(SP),SP) = true$ для ациклических *SP*.

В четвертой главе содержится описание реализации метода функционального распараллеливания *SP*.

Метод функционального статического распараллеливания применяется к крупноблочному, операторному и операционному уровням распараллеливания последовательных программ. Он состоит из двух шагов. На первом шаге из элементов последовательности правил схемы *P*, построенной по правилам множества *L*, по системе функций *F* и алгоритму *G* строится новая функция. Процесс формирования продолжается до тех пор, пока не будут исследованы все элементы схемы *P*. Каждому правилу из *L* однозначно ставится в соответствие функция f из *F*. Алгоритм *G* указывает способ построения $\mathcal{F}$, функционально эквивалентной схеме *P*. Преобразование *P* в $\mathcal{F}$ осуществляется системой эквивалентных преобразований в функциональное выражение. На втором шаге используется процедура *Q* распознавания и "расщепления" компонент полученной $\mathcal{F}$. Эта процедура применяется вначале к компонентам самого верхнего уровня, затем к составляющим компонентам следующего уровня и так далее, пока не доходим до самого низкого уровня с учетом заданного от-

ношения порядка. Механизм вышеизложенного метода предполагает наличие P, L, Q, G, F, множества отношений Σ, а также процедур B_j (j=I,...,k: $k \geqslant$I) распараллеливания выделенных компонент .

Определим все указанные компоненты.

P это последовательная программа, написанная на некотором языке программирования высокого уровня L. Она преобразуется в структурированную программу SP.

F это система функций, определенная в главе 3, п. 3.I.2.

Множество отношений Σ определено в главе 2, п. 2.2.2.

В процессе преобразования SP в FSP автоматически выполняются некоторые машинонезависимые оптимизационные преобразования , такие как "распространение" констант, исключение "мертвых" переменных, инвариантных вычислений из тела цикла, чистка фрагментов, *case*- и *if*-операторов. В результате анализа получаем FSP, написанную на промежуточном функциональном языке .

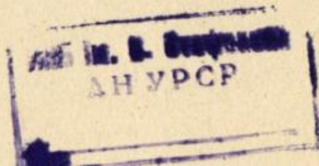
Условно алгоритм G преобразования SP в FSP делится на три части

I. Выделение всех описанных переменных и имен констант. На этом этапе каждому имени ставится в соответствие два указателя *p1* и *p2*, первоначальные значения которых равны нулю. Они указывают на специально отведенные поля: предыдущее и текущее функциональные выражения генерируемой переменной. Переменной присваивается 0, если она используема, и I, если она генерируема. Как только переменная становится текущей использованной, признак генерируемости меняется с I на 0.

2. Определение типа структурированной компоненты SP и ее преобразование в функционально структурированную. Процесс преобразования структурированных компонент SP в функциональные элементы FSP осуществляется процедурами "инструкция присваивания","составная инструкция", "*if*-инструкция", "*repeat*-инструкция", "*while*-инструкция", "*case*-инструкция", "*for*-инструкция", "выражение".

3. Формирование выходного функционального выражения FSP. На этом этапе выделяется список генерируемых переменных. Затем по указателям *p2* выбираем соответствующие функциональные выражения. Последовательность выбираемых выражений и образует искомую FSP.

Для распараллеливания FSP разработаны процедуры B_j (j=I,...,k: $k \geqslant$I), определяемые элементами F. Они используют



процедуру Q для распознавания и "расщепления" компонент полученной FSP. Q применяется сперва к компонентам самого высокого уровня, затем к составляющим компонентам следующего порядка и так далее, пока не доходим до самого низкого уровня с учетом заданного отношения порядка. Процедура Q состоит из двух этапов.

На первом этапе распараллеливания производится выявление и распараллеливание примитивных функций, вложенных if-, def-функций, определяемых пользователем, функций повторения. Распараллеливание вышеуказанных функций проводится соответствующими процедурами.

На втором этапе производится образование ветвей и формирование объектного кода.

В процедурах распараллеливания примитивных функций используется следующее определение.

Одной или группе примитивных функций ставится в соответствие приоритет - целое значение - $pr\ (f(x)) \geqslant 0$ или $pr\ (f_1,\dots,f_n) > 0$.

ОПРЕДЕЛЕНИЕ 13. Примитивные функции f_i и f_j попарно сравнимы, если $pr(f_i)\ \$\ pr(f_j)$, где $\$$ - один их знаков операции отношения ($=$, $\neq$, $<$, $\leqslant$, $>$, $\geqslant$), и упорядочены следующим образом : $pred(f_i) \leqslant pred(f_j)$ или $pred(f_i) > pred(f_j)$,
где $pred\ (f)$ - функция, указывающая на порядковый номер в упорядоченной последовательности примитивных функций одного и того же приоритета. Например, пусть примитивные функции сложения (AD), вычитания (SB), умножения (MP) и деления (DV) соответственно попарно сравнимы. Для сравнимых функций :

$$pr\ (SB) = pr\ (AD) \text{ и } pred\ (SB) < pred\ (AD);$$

$$pr\ (MP) = pr\ (DV) \text{ и } pred\ (DV) < pred\ (MP).$$

В целях достижения наибольшего параллелизма функциональное выражение приводится к виду не изменяющему результат вычислений, но приводящему к увеличению числа одновременно (на одном уровне иерархии) выполняемых примитивных функций. Преобразования выполняются только над простыми функциональными элементами, составляющие функционального выражения. Все простые функциональные выражения можно разделить на три группы :

1. $n = 2$, где n - количество аргументов примитивной функции;
2. $n > 2$;

3. n = I, то есть аргумент есть простой аргумент примитивной функции. Исследуем каждый из вышеуказанных случаев .

Случай $n = 2$. Пусть простое функциональное выражение будет иметь вид : $f\ (f_k\ (\ f_{k-1}\ (\ldots(f_1\ (x)\ldots))))$.
Для любого $i = k$-I, k-2,...2 судем рассматривать пары функций f_i и f_{i-1}. В случае, если эти функции сравнимы и имеет место отношение $pred(f_1) \geq pred(f_{i-1})$, то производим следующее преобразование: ставим функцию f_i вместе с ее первым аргументом перед аргументом функции f_{i-1}, все остальное остается без изменений. Затем переходим к функции f_{i-2}. Если же пара функций несравнима, то переходим к функции f_{i-1}. Получаем подвыражение следующего вида : $f\ (f_k\ (\ f_{i+2}\ (f_i(x),(f_{i-1}\ (x),\ldots,f_1)$.
Например, $SP = f + b - c * b\ /\ (e * f\ )$.

$FSP = AD\ (f,\ SB(\ b,\ MP\ (c,\ DV\ (b,MP(e,f)))))$.

После первого шага: $FSP_1 = AD(f,\ SB(\ b,\ DV(MP(c,b),MP(e,f))))$.
После второго шага: $FSP_2 = SB(AD(f,b),DV(MP\ (c,b),MP(e,f)))$.
FSP выполняется последовательно за 5 тактов, после преобразования FSP в FSP_2 - за 3 такта.
I такт $t_{11} = MP(c,b)$ и $t_{12} = MP(e,f)$;
2 такт $t_{21} = AD(f,b)$ и $t_{22} = DV(MP(c,b),MP(e,f))$;
3 такт $t_{31} = SB(MP(c,b),DV(MP(c,b),MP(e,f)))$.
Такты выполняются параллельно .

Случай $n > 2$. Для $n>2$ все преобразования аналогичны случаю $n = 2$.

Случай n = I. Пусть n = I. Тогда функциональное выражение имеет вид : $f(x)$, где f - унарная примитивная функция. Преобразование выполняется только тогда, когда имеется по крайней мере пара унарных функций, следующих друг за другом. Процесс преобразования FSP продолжается до тех пор, пока не обработано все выражение.

Описанная процедура распараллеливания функционального выражения, состоящего из композиций примитивных функций, приводит к параллельному виду вышеописанными способами, способствующими увеличению степени статического параллелизма.

Для определения алгоритма распараллеливания вложенных if-функций введем ряд обозначений. Пусть дана f_{n+1} вложенная if- функция вида : $f_{n+1} = if\ p_1(x)\ then\ f_1(x)$

$else\ if\ p_2(x)\ then\ f_2(x)$

$\dots$

$else\ if\ p_n\ (x)\ then\ f_n(x)$

$else\ f_{n+1}(x)$.

Для каждой ветви ξ_i (i = I,...,$n+1$) вычисления функции f_i определим параллельное вычисление f_i тогда и только тогда, когда

I.$\forall f_i$: если $p_i(x) = true$, то для выполнения f_i необходимо, чтобы $\rho_{i-1} = \bigcap_{j=1}^{i-1} (p_j(x) = false)$.

2. f_j (j = I,...,i-I) по ветве ξ_i вычисляется f_i тогда и только тогда, когда $\rho_{i-1} = \bigcup_{i=1} (p_i(x) = true)$.

Алгоритм.

I. j = I. Вычисляется $p_1(x)$. Положим $J = J0$ (J0 - это множество ветвей ξ_i вычисления $\{ p_j(x) \}$).

$f_j \in \{ \xi_j : \rho_{j-1} \vee \exists i : f_i \in \{f_k\} \ \&\ p_i(x) = true \}$. $(k \leq i)$

2. Пусть сформировалось некоторое множество ветвей вычисления f_j (j = I,...,k) $f_j \in \{f_k\}$. Исследуем условия включения f_{j+1} в указанное множество :

а) f_{j+1} включается в ξ_{j+1} тогда и только тогда, когда $f_{j+1} \in \{ \xi_{j+1} : \exists (\rho_{j-1} = false) \vee \exists j+1 : f_{j+1} \in \{f_k\}, f_j \in \xi_j$

b) f_{j+1} вычисляется тогда и только тогда , когда $f_{j+1} \in f_j$ $f_{j+1} \in \{ f_j \ \&\ f_{j+1} \in \{ \exists \rho_{j-1} = true \vee \exists j+1 : f_{j+1} \in \{f_k\} \ \&$ $(p_{j+1} = true) \ \&\ f_j \in \xi_j \}\}$, $j = 1,\dots,k-1$.

3. Для $j = k$: f_k вычисляется, если $\rho_{k-1} = false$ и $p_k = true$, в противном случае выполняется f_{k+1}.

Вначале образуется вектор предикатных функций $\{\rho_i\ (x)\}_{i=1}^{k}$

$\rho_i = \bigcap_{j=1}^{k} (p_j(x) = false)$ (j = I,...,i) $i < k$.

$J = \{p_i(x) = false \}$(i = I,..,j) $0 \leq j \leq k$.

Для выполнения f_i из $\{f_k\}$ с другими функциями,если таковые присутствуют в функциональном выражении, необходимо выбрать первую предикатную функцию $p_i(x)$ из $\{ p_j(x) \}$ (j = I,...,k), имеющую значение $true$. Если $\rho_k = \bigcap_{i=1}^{k} (p_i(x) = false)$, то для выполнения выбирается f_{k+1}.

Основные трудности организации вычисления f_k "скрыты" в предварительном анализе и параллельном выполнении $p_i(x)$, (i=I,...,k), формировании ρ_k и в выборе $f_i \in \{f_j\}$(j=I,...,k),

для которого $p_l(x) = true$, в противном случае ($\rho_k = false$) выбирается f_{k+1}.

Def-функция - это функция, определяемая пользователем в своих программах. Пусть f_k - *def*-функция типа (2), то есть $f_k = f_k(f_{k_1}, f_{k_2}, \ldots, f_{k_m})$.
Тогда аргументы функции $f_{k_1}, f_{k_2}, \ldots, f_{k_m}$ будут выполняться параллельно, так как условия

$$\begin{cases} In(f_{k_l}) \cap In(f_{k_j}) = \varnothing \\ Out(f_{k_l}) \cap In(f_{k_j}) = \varnothing \\ In(f_{k_l}) \cap Out(f_{k_j}) = \varnothing \end{cases}$$ выполняются.

Пусть f_k - *def*-функция типа (3), то есть $f_k = (f_{k_m}(\ldots f_{k_1}(\ldots)\ldots)$. Тогда f_k распараллеливается процедурами, рассмотренными выше.

В состав функций повторения входят *for* -, *while* - и *repeat*-функции .

Все, что будет изложено для *while* - функции , в равной степени применимо как к *for*-, так и к *repeat* - функции.

While -функция имеет вид : *while* $p(x)$ *do* $f(x)$.
В цикле, как было уже отмечено, можно выделить основные три части : I.инициализация входных переменных цикла (*init* - часть); 2.условие цикла (*bool* - часть); 3.тело цикла (*body* - часть), в которой выделены 3.I. действие цикла и 3.2. приращение переменных цикла, принадлежащих множеству входных переменных *bool* - части.

При преобразовании *while* цикла из *SP* в *while*- функцию в *FSP*
init - часть переходит в примитивные функции;
bool - часть - в логическое выражение $p(x)$;
body - часть в функциональное выражение.

Рассмотрим более детально преобразование *body* - части в $f(x)$. Все переменные цикла, принадлежащие входному множеству *bool* - части цикла *while*, становятся накапливающими параметрами. При преобразовании они становятся примитивными функциями. Действие тела цикла "расщепляется" на составные части согласно ниже изложенному утверждению.

УТВЕРЖДЕНИЕ 5. При преобразовании *while* цикла из *SP* в *while* -

функцию в *FSP* цикл "расщепляется" на $r \leqslant q$ *while*-функций, где q - это количество генерируемых выходных переменных операторов, составляющие действие-часть тела цикла в *SP* и зависящие от входных параметров условия *while* цикла.

Истинность данного утверждения вытекает из процесса построения и преобразования *SP* в *FSP* и оно отражает сущность процедуры распараллеливания *while* -функции.

В результате распараллеливания из полученных компонент функционального выражения образуются ветви и формируется объектный код для параллельного выполнения на МВС.

Простота и наглядность изложенного метода, его естественное соответствие логической и информационной структуре исходных программ делают этот подход приемлемым для автоматического распараллеливания последовательных программ.

Заключение перечисляет основные результаты работы и перспективы дальнейших исследований.

Список литературы содержит основные источники изучения.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ

В работе предлагается метод функционального статического распараллеливания последовательных программ. Он применяется только к операционному уровню и состоит из двух частей:

- на первом этапе последовательная программа преобразуется в функциональную, эквивалентную исходной;
- на втором производится распараллеливание полученной функциональной программы.

В отличие от других методов в данной работе проблема распараллеливания решается на основе принципов функционального программирования. Процедуры распараллеливания естественным образом, сохраняя как информационную, так и логическую структуру исходной программы, выявляют в ней асинхронные ветви. Полученные ветви более эффективно реализуются на МВС с гибкой динамически настраиваемой структурой, в которой встроены примитивные функции.

В работе получены следующие основные результаты.

Приведен алгоритм преобразования последовательной программы в функциональную, доказана корректность этого преобразования.

Построена процедура коректного преобразования как ацикли-

ческих, так и циклических *SP* в *FSP*.

Определены достаточные условия корректного распараллеливания *FSP*.

Определены элементы промежуточного функционального языка и структура блока распараллеливания.

Разработаны процедуры распараллеливания примитивных, *def*- и вложенных *if*-функций, функций повторения.

Исследовано взаимодействие алгоритма преобразования *SP* в *FSP* и машинонезависимой оптимизации программ.

Приведены сравнительные характеристики метода последовательного углубления и метода функционального распараллеливания программ.

Публикации.

Основные положения диссертации изложены в следующих работах:

I. Боян Е.Т..Функциональный метод статического распараллеливания последовательных программ. - в сб. Теория и практика программирования.- Кишинев, Штиинца, 1989, с.28-36.
2. Боян Е.Т. Анализ сложности статического распараллеливания структурированных программ. - в сб. тезисов 7-ая Всесоюзная школа - семинар "Параллельное программирование и высокопроизводительные системы", Киев, Институт кибернетики им. В. М. Глушкова АН УССР, 1988, с. 99-100.
3. Боян Е.Т. Промежуточный функциональный язык в системе трансляции АСВТ ПС. - в сб. Докладов школы - семинара "Прикладное программное обеспечение ЭВМ архитектурной линии СМ/I/ СМ-2, АСВТ ПС", Калинин, с.6-8.
4. Боян Е.Т. О корректности преобразования структурированных в функционально структурированные программы. - в сб. Программное и математическое обеспечение ЭВМ. - Кишинев, Штиинца, 1990, с.36-47.
5. Боян Е.Т. Теоретические основы преобразования структурированных программ в функционально структурированные. - в сб. Системное и теоретическое программирование. Прикладные аспекты. - Кишинев, Штиинца, 1988, с. 22-44.
6. Кожокарь С.К., Топал Л.Л., Демидова В.В., Боян Е.Т.

Программный комплекс "Транслятор с языка Паскаль на язык виртуальной машины".- Инв. номер ГОСФАП - 50880000222 от 24.02.88.

7. Кожокарь С.К., Чеботарь К.С., Демидова В.В., Боян Е.Т. Швец Н.М., Салинская Т.И. и др. Интегрированная система трансляции на основе языка ПАСКАЛЬ для УВК "САМСОН". - Инв. номер ГОСФАП - 50910000249 от 02.06.91.

8. Боян Е.Т.,Демидова В.В., Кожокарь С.К., Терехов А.Н., Топал Л.Л. Организация информационных связей этапа анализа в компиляторе с языка Паскаль. В сб. Системное и теоретическое программирование. - Кишинев, Штиинца,1987, с.45-49.

9. Боян Е.Т. Алгоритм преобразования структурированных программ в функционально структурированные и машинонезависимая оптимизация.- в сб. Программное обеспечение вычислительных комплексов. - Кишинев, Штиинца, 1988, с.27-39.

10. Боян Е.Т. Распараллеливание циклов. - в сб. Теория и практика разработки программных комплексов. - Кишинев, Штиинца, 1986, с. 17 - 28.

11. Боян Е.Т. Сравнение методов распараллеливания последовательных программ. - в сб.Адаптируемые средства программирования. Методы оценки трансляторов. Материалы школы-семинара. 25 мая - 10 июня 1989 г. г. Кишинев, с. 72-74

БОЯН ЕЛЕНА ТРИФИЛИВНА

ИССЛЕДОВАНИЕ И РАЗРАБОТКА МЕТОДОВ РАСПАРАЛЛЕЛИВАНИЯ ПРОГРАММ

05.13.11 - Математическое и програмное обеспечение вычислительных машин, комплексов систем и сетей

(а в т о р е ф е р а т)

Подписано в печать 15.07.92. Формат 60x84/16. Ротапринт. Объем 1,5 п.л. Заказ 247. Тираж 100.

ИПП "Штиинца". 277028. Кишинев, ул. Академией,3

467846

Ab 25.742