

Вадим Исаев



ФОРТРАН

великий, ужасный
и непобедимый

СОДЕРЖАНИЕ

Предисловие автора

1. Фортран и его история

1.1 Краткая история Фортрана до своей стандартизации,
а заодно и пара слов о вычислительной технике

1.2 Фортран стандартизованный – вперёд к звёздам

1.3 «Параллельные» стандарты – OpenMP и MPI

1.4 Фортран в СССР

2. Фортран и текущая реальность

2.1 Чем можно пользоваться для программирования на
Фортране

2.2 Чем пользуется автор (GNU Fortran)

2.3 Установка GNU Fortran (gfortran) в дистрибутивах
Linux

2.4 Установка GNU Fortran (gfortran) в Windows

2.5 Установка GNU Fortran (gfortran) в BSD-системах

2.6 Как компилировать программы с помощью gfortran

ПРЕДИСЛОВИЕ АВТОРА

Учитывая текущее плачевное состояние наших программ, можно сказать, что программирование – определенно всё ещё чёрная магия и пока мы не можем называть её технической дисциплиной.

Билл Клинтон, бывший президент США

Я всегда мечтал о том, чтобы моим компьютером можно было пользоваться так же легко, как телефоном. Моя мечта сбылась – я уже не могу разобраться, как пользоваться моим телефоном.

Бъёрг Страуструп

Однажды армянское радио спросили:

- Почему люди до сих пор используют Фортран?
- По привычке – ответило армянское радио.

От себя к армянскому радио хочу добавить, что если вы занимаетесь разнообразнейшими вычислениями и вам нет дела до всевозможных графических интерфейсов с ихними рюшечками-котиками-бантиками, то привычка использовать Фортран такая же хорошая, как чистить зубы или мыть руки перед едой. Посудите сами, когда в 50-ых годах прошлого века появился Фортран, у него не было никаких конкурентов (Ассемблер конкурентом считать не будем – совсем другая специфика), однако в 60-ых и 70-ых годах, конкурентов появилось вагон и маленькая тележка, потому что тот Фортран не был идеалом. Вдобавок, в конце прошлого века появилось множество математических программ, в которых есть как специальные функции среднего, так и высокого уровня. Это сильно отодвинуло Фортран от популярных языков. Но до сих пор Фортран, как вычислительный язык, совсем не забыт, в отличие от других языков, например, Алгола. А это уже о чём-то говорит. Надо сказать, что многочисленные математические программы, которые используют численные методы, сделаны как раз по образцу и подобию Фортрана. И это тоже говорит в его пользу...

Так что если вам нужен язык программирования для сложных, или не очень, вычислений с большими объёмами данных или компилятор, которого делает быстрые программы (Си сейчас, при определённых условиях, не менее быстр), и является простым в использовании, без изнуряющей отладки ошибок (которая у языка Си порою занимает на порядок больше времени, чем само программирование), то Фортран – это то, что врач прописал. И если вам нужно по-скорому накалякать какой-нибудь вычислительный пустяк, не сильно задумываясь о типах данных – Фортран тоже поможет.

Книга предназначена для тех, кто когда-то (в школе, колледже или университете) краем уха слышал о программировании и существовании языков программирования, на которых не в меру умные «ботаны» пишут какую-то абракадабру под называнием «программы».



Сразу оговорюсь – это не учебник, а всего лишь небольшое, порою ироничное, знакомство с языком, которое довольно сильно насыщено сравнениями с очень распространённым языком Си. Практическая часть книги начинается, в небольшом количестве, с базовых принципов использования языка. Это поможет немедленно, не теряя ни секунды, приступать к программированию.

Естественно, поскольку книга о Фортране, сравнение с языком Си всегда выходят не в пользу последнего. Такие сравнения отнюдь не говорят о том, что Си плохой язык, просто по жанру так положено и не более того ☺. И если вам, как и мне, Фортран понравится – я буду чрезвычайно рад.

Успехов и удачи вам в любом начинании!

1. ФОРТРАН И ЕГО ИСТОРИЯ

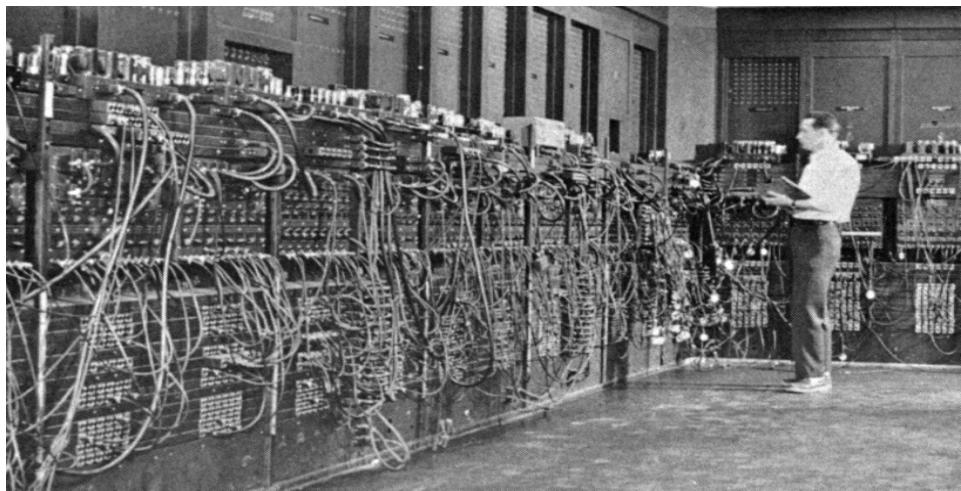
*История – это вымысел, с которым все согласны.
Вольтер*

1.1 Краткая история Фортрана до своей стандартизации, а заодно и пара слов о вычислительной технике

*Без шлифовки и алмаз не блестит.
Японская поговорка*

Эта увлекательнейшая и полная мистических тайн история случилась давным-давно. ЭВМ только-только начали менять свои ламповые потроха на транзисторные. А оперативная память у этих доисторических компьютеров перевалила за свой первый килобайт. Народ, эксплуатирующий подобных ма-стодонтов, начал задумываться, а как бы облегчить свой очень нелёгкий труд. Этих людей сейчас называют «программисты», а тогда они именовались «оператор ЭВМ».

Сначала программы набирались на пульте ЭВМ специаль-



Первый американский компьютер «Эниак»

ными переключателями «0 – 1», а результаты выдавались в виде

горящих или не горящих лампочек. Правда такая система программирования не касается американского монстра «Эниак» [1]. Это несчастное создание программировался обычными проводами. Кабели, которые вы видите на фото, это не электропитание блоков – это программа ☺.

Небольшое лирическое отступление про «Эниак»

Хочу сказать, что у меня в детстве была примерно такая же (не по размерам, конечно, а по системе программирования) ЭВМ. И стоила она в отличие от американского «Эниака» не 6 500 000 долларов, а всего 10 рублей ☺ и называлась ДВМ-1М.

Вообще, от американцев хватовства на три порядка больше, чем реальной пользы. «Эниак» не стал здесь исключением. Для начала надо сказать, что отнюдь не первая реально работающая в мире ЭВМ. На самом деле первую вычислительную машину сделал Конрад Цузе [2] в Германии, в 1936-1941 годах. Хотя, справедливости ради, нужно сказать, что машину Цузе следует называть не электронно-вычислительной, а электрически-вычислительной, т. к. триггеры он делал на основе телефонных реле. Надо отметить, что Цузе вовсе не был принципиальным противником электронных ламп. Дело в том, что при создании своей первой модели машины Z1, он в электронике вообще ничего не понимал. Поэтому его Z1 была механической и работала от электродвигателя, который, согласно составленной программе, соединял и разъединял металлические пластинки. Однако в последующем к разработке подключился его друг, инженер Гельмут Шреер, который как раз и был инженером-электронщиком, и про электронику знал практически всё. Следующая разработка, с его участием, уже велась с использованием электронных ламп, однако практического воплощения не получила. Там предполагалось применить порядка 2 000 ламп, а получить или купить их в предвоенной обстановке было делом совершенно невозможным. Для гражданского населения электронника была дорогой, а расчёт на военных полностью провалился. Когда военные заказали своим учёным провести экспертизу этого проекта, то научное сообщество было настолько ошеломлено количеством электронных ламп, что просто отказалось воспринимать это всерьёз. Ну, сами посудите – в радиоприёмнике использовалось от 3 до 6 ламп, в приёмопередатчике и магнитофоне – 10-12 (это была самая серьёзная электроника в то время). А тут извольте любоваться – целых две тысячи штук... Поэтому от ламп пришлось полностью отказаться и использовать простые реле.

Однако во всём остальном его машина была намного круче американской. Судите сами:



Советский аналог
американского «Эниака» –
Детская Вычислительная
Машина (ДВМ-1)

• Программы автоматически вводились с помощью перфорированной бумажной ленты. Правда сначала предполагалось использовать 35-мм киноплёнку, как более прочную и с которой удобно копировать программы. Но этот вариант оказался дороговатым. У американцев ввод программ – вручную, перетыканием кабелей. И лишь намного позднее стали применять перфокарты, которые тоже сначала скармливались машине вручную, а потом сделали специальное автоматическое устройство.

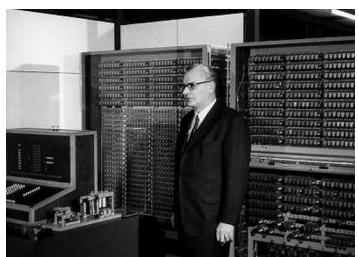
• Потребление электроэнергии намного меньше, т. к. отсутствовал постоянный подогрев анодов ламп. У американцев – 174 кВт, вне зависимости от того, работает машина или стоит как пень. У Цузе машина потребляла 4 кВт только во время вычисления.

• Самое главное – надёжность работы. У американцев в машине использовалось чуть менее 18 000 электронных ламп. Едва ли не каждый день перегорало по одной лампе, на поиск которой уходило порою по несколько часов. Это происходило из-за довольно большой плотности монтажа, а вот система охлаждения была продумана недостаточно хорошо. При включении машины их иногда перегорало до нескольких десятков из-за сильного скачка тока. Потом, правда, американцы придумали одну хитрушку – при выключении машины, отключалось только высокое напряжение, которое работало как сигнальное, а подогрев анодов не отключался совсем, его напряжение просто понижалось до 5 вольт.

• Ну и чтобы совсем уж добить хвастунов-американцев. Вес «Эниака» – 27 тонн, вес Z3 – 1 тонна ☺.

Так что самовосхваление американцев несколько сомнительно из-за особенностей эксплуатации «Эниака». А ещё хочу сказать, что полная стоимость американской разработки составила 6 500 000 долларов, а разработка Цузе стоила всего лишь 50 000 рейхсмарок. Почувствуйте разницу ☺.

До 1944 года машина Цузе Z3 работала на немецкий исследовательский институт аэродинамики по расчётом крыльев самолётов и аэrodинамики баллистических ракет. В 1944 году её разбомбили вместе с институтом. Правда, Цузе успел вывезти свою следующую разработку Z4 в безопасное место.



Конрад Цузе и его вычислительная машина Z3

Компьютеры, которые создавались в более цивилизованных странах, например, в Великобритании, Франции или СССР, имели специальные пульты управления, с помощью которых вводилась расчётная программа.



Оператор за вводом программы с пульта

У такого пульта было только одно преимущество – оператору не надо было бегать, как ошпаренному, по залу управления ЭВМ и перетыкать провода. Однако скорости ввода программ пульт сильно не способствовал.

Однажды какая-то умная голова решила приспособить клавиатуру от печатной машинки к ЭВМ, что реально повысило удобство работы с

машиной. И от этого сразу возникла мысль, а не повысить ли заодно и удобство программирования. Вот только как?

Сначала появился язык Ассемблер, который ноли-единицы команд процессора представлял в читаемом текстовом виде. С высоты сегодняшней нашей избалованности разнообразными языками программирования, Ассемблер покажется не совсем человеческим, но в те легендарные времена этот язык был явным прогрессом.

Когда оперативная память ЭВМ перевалила за второй килобайт, возникла мысль улучшить и ассемблерное программирование, поскольку программы стали большими и набирать по 20-40 страниц кода – запаришься, да и ошибок много с такими маловразумительными словечками.

Как раз в это время, примерно с середины 50-ых годов прошлого столетия, возникли три специализированных языка высокого уровня, т. е. они программировали не с помощью команд процессора, а с помощью команд, которые математики и прочий учёный люд используют при ручных расчётах. Это были языки Лисп, Кобол и Фортран.

Лисп предназначался для создания систем искусственного интеллекта, Кобол – для обслуживания разнообразнейших бизнес-задач, а Фортран – для научных вычислений. Особо подчеркну, что языки были именно специализированные, универсализм тогда никто не планировал в силу ограниченности компьютерных ресурсов. Именно поэтому, взяв, к примеру, язык Кобол, решить на нём какую-нибудь систему линейных уравнений будет, мягко скажем, затруднительно. У Лиспа с математикой ситуация намного лучше, т. к. искусственный интеллект и математика ходят где-то рядом друг с другом, но син-

таксис у языка его скорому изучению никак не способствовал. В те времена Лисп мало походил на классическую математическую формулу. Следовательно, нужен был отдельный язык для решения всяких разных физмат задач.

Зачинатель Фортрана – известнейший в мире программирования человек Джон Бэкус, практически всю свою жизнь проработал в фирме IBM. После службы в армии, Бэкус изучал электротехнику и сильно интересовался звуковоспроизводящими устройствами (как и я в своё время ☺). Во время учёбы, его научный руководитель посоветовал Бэкусу получить ещё и математическое образование, поскольку юноша был очень умным и очень настырным. Получив второе образование математика, он устроился в IBM, где его сразу же определили в группу разработки компьютеров. Первым его самостоятельным делом было создание интерпретатора для IBM 704. Работая над ним, изрядно вспотев и изматерившись от пользования Ассемблером, он пришёл к мысли, что надо бы создать какой-нибудь нормальный человеческий язык программирования для простых человеческих задач, типа задачи Коши и прочих там Рунге-Кутты.



Джон Бэкус – отец-основатель Фортрана

Небольшое лирическое отступление о Джоне Бэкусе

Вообще-то за идею создать высокоуровневый язык программирования для научных вычислений Джону Бэкусу надо памятник поставить. Дело в том, что IBM 704 вообще не имела человеческих, в нашем сегодняшнем понимании, средств общения с собой. Как выглядели клавиатура и дисплей которые, подключались к машине, можно увидеть на рисунке.

Честное слово, глядя на этот агрегат очень трудно прийти к мысли, что с ЭВМ можно общаться на человеческом языке. Вводить туда с помощью этого пульта управления можно



Клавиатура и дисплей у ЭВМ IBM 704

было только двоичные данные. Получить информацию возможно также только двоичную – в виде свечения неоновых лампочек в верхней части пульта.

На самом деле программа для вычисления составлялась на бумажке, а потом компилировалась на совершенно отдельном устройстве – перфораторе, который с машиной вообще никак не был связан. И уже готовые, пробитые в нужных местах перфокарты, скармливались машине.

Так что, как ни крути, а Джон Бэкус самый настоящий гений. И был бы гением только за одно изобретение Фортрана, даже без учёта всех его остальных великих дел на ниве программирования.

Как математик он мыслил, что язык, конечно же, будет похож на математические формулы. Задача, прямо скажем, достойная гения. Ну, сами посудите, с одной стороны язык должен быть достаточно простым, чтобы учёный человек, который сроду ни разу не сталкивался с программированием, не пялился на код программы, как баран на новые ворота. А с другой стороны, в те времена не существовало никаких прототипов, кроме Ассемблера, с которых можно было бы брать пример. В общем, в таких вот противоречиях и родилась идея создать язык Фортран (сокр. ФОРмул ТРАНслятор – переводчик формул), который собственно и является помощником воплощения всяких разных математических формул в компьютере. Поскольку никакой другой ЭВМ у Бэкуса под рукой не было, кроме уже упомянутой IBM 704, программным обеспечением которой он должен был заниматься не покладая рук, первая реализация Фортрана была разработана именно для этой машины [3].

Сначала, как Джон ни старался, сделать Фортран сильно уж не похожим на Ассемблер не удалось. В Ассемблере, к примеру, используются неявные циклы, которые базируются на условных переходах к какой-то метке. В первоначальном Фортране, хоть и появился оператор начала цикла DO, но вот оператора окончания цикла не было. Понятно, было необходимо, чтобы новый язык не был совсем уж незнакомым и переход на него не был бы таким уж болезненным. Поэтому начало цикла задавалось с помощью ключевого слова DO, а его последняя строчка определялась какой-нибудь меткой. Вот так, например, выглядело вычисление среднего арифметического значения вектора:

```
SUMM = 0
DO 45 I = 1, 10
45  SUMM = SUMM + VECTOR(I)
     PRINT *, SUMM/10
```

Здесь, после оператора начала цикла DO указывается номер метки 45, которая определяет последний оператор цикла.

Так же не было синтаксических единиц, которые бы обозначали отдельно стоящую подпрограмму (процедуру), хотя процедуры, как объекты программы, вовсю использовались. Делалось это с помощью условных или безусловных переходов (IF (условие) GO TO метка или просто GO TO метка) и оператора возврата RETURN. Вот, например, генерация массива псевдослучайных чисел, равномерно распределенных в интервале (0, 1):

```
DIMENSION MASSIV(3)
ISEED=123457
N=3

! Вызов подпрограммы, которая возле метки 222
GO TO 222

PRINT 1,ISEED
PRINT 2,MASSIV
1  FORMAT(1X,'РЕЗУЛЬТАТ'/1X,I25)
2  FORMAT(1X,'МАССИВ = ',1X,3E16.7)

! Отсылка к концу программы
GO TO 444

! Здесь начинается подпрограмм
222 INTEGER I,D2P32M
        DOUBLE PRECISION Z,D2P31M,D2PN31,DMOD,DFLOAT
        DATA D2P31M/2147483647.D0/,D2PN31/
*   4.656612873077393D-10/,D2P32M/16807/
        Z=DFLOAT(ISEED)
        DO 10 I=1,N
        Z=DMOD(D2P32M*Z,D2P31M)
10      MASSIV(I)=Z*D2PN31
        ISEED=Z
```

! А тут подпрограмма заканчивается
RETURN

444 STOP
END

В тексте программы строка GO TO 222 говорит программе перейти на метку 222, с которой начинается подпрограмма генерации массива псевдослучайных чисел. Конечно, в данной программе такая подпрограмма излишняя, вот когда надо сто раз провести подобную генерацию, то код программы значительно сокращается, т. к. не надо сто раз писать один и тот же текст.

В общем и целом, Фортран получился языком простым, не требующий больших усилий для своего изучения и реализации, так что учёным, которым и так было чем заняться, он пришёлся очень по вкусу.

Однако были там и определённые недостатки или можно сказать нюансы, которые появились вместе с новым языком. Сначала посмотрите, каким образом писались фортрановские программы. Ну и совсем уже древний текст, исключительно для понимания, насколько везёт сегодняшним программистам, представлен на рисунке ниже.

Текст совсем древней программы на Фортране

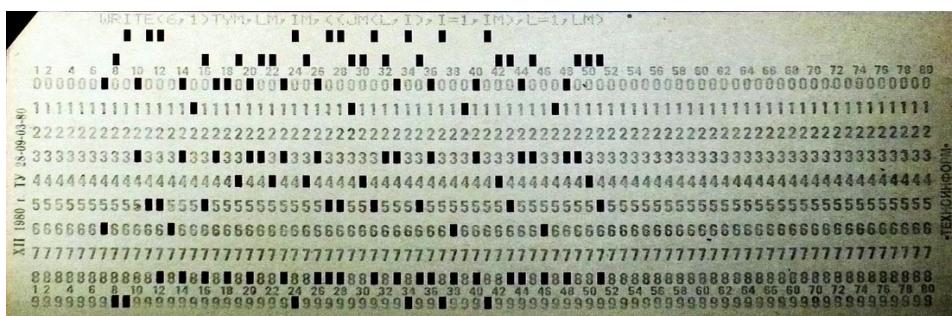
Прошу обратить внимание, что в рукописном тексте символ **0** (ноль) отличался от символа **O** (прописной буквы О) тем, что буква **O** была перечёркнута по диагонали – **Ø**. Правда такое перечёркивание относилось к американскому формату печати текста. В Европе и Советском Союзе было всё наоборот – перечёркивался именно ноль, а буква **O** печаталось просто так. В дальнейшем, с развитием аппаратов для печати текста, такая проблема отпала.

Это был так называемый «фиксированный» формат, который применялся до введения в Фортран-90 «свободного» формата.

FOR COMMENT		CONTINUATION	FORTRAN STATEMENT			IDENTIFICATION		
STATEMENT NUMBER			1	2	3	72	73	80
C			PROGRAM FOR FINDING THE LARGEST VALUE					
C	X		ATTAINED BY A SET OF NUMBERS					
			BIGA = A(1)					
			DO 20 I = 2,N					
			IF (BIGA - A(I)) 10, 20, 20					
10			BIGA = A(I)					
20			CONTINUE					

Типичная программа на Фортране в старом формате

Как видите, часто программа писалась на специальной карточке. Это делалось специально, чтобы можно было быстро найти тот кусок кода на откомпилированной карточке (см. рис. ниже), чтобы его можно было быстро заменить. Если пригля-



Перфокарта – это часть древней откомпилированной программы

деться, то под заголовком карточки увидите мелкие цифры (1, 5, 6, 7, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80). Это разметка номеров колонок. В первой ко-

лонке должен стоять символ комментария С, если текущая строка является комментарием. Колонки со 2 по 5 отводились для номера метки (на карточке можно увидеть две метки – 10 и 20). Колонка 6 нужна для знака «продолжение предыдущей строки», если текст не влезал в одну строку. Текст программы начинается с колонки 7 и продолжается до колонки 72. Если строчка не влезла, то в колонке 6 следующей строки карточки ставится значок X или *, что говорило о продолжении предыдущей строки. Колонки с 73 по 80 применялись для каких-нибудь служебных целей. Такой формат был ориентирован на перфокарты, на которых в нужных местах пробивались дырочки, а потом перфокарты скормливалисьчитывающему устройству компьютера. Правда, это всего лишь часть многочисленных легенд, которые любят рассказывать о Фортране, но, согласитесь, ведь очень похоже ☺. После введения автоматизации в компилирование программ (т. е. пробивания дырочек в карточке ☺) стало возможно писать текст кода на самой карточке с дырочками.

Такой стандарт для текстов программ продержался аж до начала 90-ых годов. Надо сказать, что больше никто из разработчиков языков программирования такого формата не придерживался. Сей формат записи позаимствован прямиком из Ассемблера. Организация циклов и подпрограмм – тоже заимствование из Ассемблера. И когда программы по своим размерам стали всё более и более увеличиваться, такое заимствование привело к ситуации, когда исходный код стал очень плохо читаем и для начинающих программистов – плохо понимаем. Ну, сами посудите, если в ста строках программы, на пятидесяти вас пересыпают с места на место с помощью GO TO («иди на...»), программистский пыл на высоком уровне долго не продержится. Может показаться – зачем вводить в язык столько неудобств, но это была плата за скорость и простоту разработки языка.

Небольшое лирическое отступление про перфокарты

Если вы думаете, что перфокарты появились только в середине прошлого века, как сопутствующий элемент ЭВМ, то вы сильно ошибаетесь ☺. Перфокарты изначально использовались как носители программ для полуавтоматических ткацких станков и такие станки появились ещё в 1725 году. Правда, первый программируемый станок работал не с перфокартами, а с перфолентой. На перфокарты перешли

несколько позже, т. к. с ними можно было оперативно изменять текущую программу станков. Т. е. для ЭВМ использовали идею, которая к тому времени существовала уже более двухсот лет.



Ручной протыкатель
перфокарточек –
компилятор

пользования программатором – перфопрограммой занялись электромагниты, придумали клавиатуру от печатной машины, добавили устройство автоматической подачи карточек. Карточки стали выпускать единого, стандартного размера, на них стали печатать буквами тот текст программы, которая дырявилась именно на этой карте и прочие удоб-

Сначала перфокарты дырявили вручную, а потом изобрели специальные машинки – механизированные перфораторы, потому что ручное дырявление было очень нудное и требовало постоянного и пристального внимания из-за того, что программы в одном экземпляре не делались. Из-за ручной работы постоянно возникали «ошибки компиляции».

Дальнейшие усовершенствования сводились к удобству ис-



Автоматизированная
машинка для протыкания
карточек фирмы IBM



Система автоматического
ввода данных в ЭВМ с
перфокарт EC-6012

ства. В конце концов, перфокарты стала делать сама ЭВМ.

На рисунке справа показана одна из совершенных машинок для перфорации карт от фирмы IBM.

Для ввода данных в ЭВМ с перфокарты, сначала тоже применялся ручной ввод, но потом изобрели специальные автоматические системы, например EC-6012, изображенную на рисунке ниже.

Данные с перфокарточки считывались двумя способами:

- Электромеханическим – когда карточка прогонялась металлическими роликами, а сверху были электрические щетки. Попадая в пробитое отверстие, щетки замыкали электрический контакт, и таким

образом рождалась логическая единица. Если же в нужном месте отверстия не было, то это был логический ноль.

• Фотоэлектрическим – карточка прогонялась тоже роликами, но уже резиновыми, но вот внизу были специальные элементы, типа фоторезистора, а сверху светили маленькие электролампочки. Как только попадалось отверстие, фоторезистор освещался лампочкой, ток, идущий через него, менялся, и это была логическая единица.

В считывателе перфокарт ЕС-6019 применялась вакуумная система подачи перфокарточек и работал он со скоростью 1 200 карточек в минуту.

Для перфорации карточек в машинах серии ЕС использовался аппарат ЕС-9010, который во время своей работы производил ужасающий шум и некоторые особо впечатлительные и юные дамы-операторы ЭВМ предпочитали во время работы этого «бармалея» удаляться в коридор или в дамскую комнату ☺.

Ещё мне рассказывали одну страшную историю про ввод с карточек, когда машинка ввода вместо того, чтобы спокойно скормить информацию с карточек в ЭВМ, вдруг начинала с ужасающим визгом разбрасывать карточки по всему залу. Вы только представьте, тысячи карточек, которые должны лежать в строго определённом порядке, чтобы правильно и последовательно вводить программу оказывались, раскиданы везде в страшном беспорядке и приходилось тратить несколько часов, чтобы собрать карточки и опять сложить их в нужном порядке. Подозреваю, что именно из таких историй рождались идеи фантастов про искусственный интеллект и обязательно следующее за этим восстание машин против человечества... ☺

1.2 Фортран стандартизованный – вперёд к звёздам

*О стандартах мы обычно не задумываемся,
за исключением тех случаев, когда их
отсутствие причиняет нам неудобства
Из обращения глав МЭК, ИСО и МСЭ
к Всемирному дню стандартов, 1998 год*

В преддверии к стандарту

Стандартизацией Фортрана занимается специальный комитет, который в США называется J3 [5] и который сейчас числится подкомитетом Международного комитета по стандартам информационных технологий (INCITS). В INCITS фортрановский комитет именуется WG5 [6]. Россия принимает самое живейшее участие в деятельности этого комитета.

Надо сказать, что первый из стандартов Фортрана появился довольно поздно – базовый стандарт приняли в 1966 году, а окончательный – в 1972-ом, который и стал называться Фортран-66. К тому времени, 1966 год, фирма IBM успела выпустить Фортраны версии II и IV. Кстати, если версией I считать первоначальную версию Бэкуса, то о версии III никто никогда не слышал ☺. В Европе уже успели ввести два стандарта на собственный язык Алгол. В самой Америке стандартизовали универсальный язык для всего на свете ПЛ/1, тоже разработки IBM, идеи которого слямзили как раз с Алгола. Правда сам ПЛ/1, как и в последствии Алгол-68, оказался проектом полностью нереализуемым из-за слабых технических возможностей тогдашних ЭВМ. Хотя транслайторов для него было сделано некоторое количество – например, он входил в стандартный комплект программ для IBM 360, которая у нас волшебным образом превратилась в ЕС ЭВМ ☺. Даже успел появиться язык программирования Бэйсик, который был сделан по образцу раннего Фортрана, но был несколько более структурирован в плане синтаксиса и не требовал для своей работы мощных технических ресурсов. Почему так? Не совсем понятно, видимо какие-то IBM'овские патентные заморочки.

Более интересен вопрос – а зачем вообще было стандартизировать Фортран? С Ассемблером понятно – нет единого для всего государства процессора, нет и стандарта. А вот Лисп, к

примеру, никто стандартизировать до 1994 года так и не со-брался. На сегодняшний день диалектов Лиспа выпущено довольно много. Как раз в таком разнообразии и кроется причина необходимости стандартизации.

По словам одного из популяризаторов Фортрана Валери Кальдербенка, после появления на свет Фортрана II, а особенно Фортрана IV, каждый вычислительный центр в США посчитал своим долгом иметь собственную реализацию Фортрана. Тем более что его реализация оказалась делом совсем не трудным. Сами понимаете, что каждая творческая личность, а все учёные и программисты как раз такими являются, посчитали своим священным долгом внести поправки в исходный язык, чтобы программировать стало намного лучше. Увы, после такого повального энтузиазма обмен программами между университетами и вычислительными центрами стал совершенно невозможен.

В 1963 году на очередной научной конференции была высказана идея о стандартизации Фортрана, после чего и началась работа по согласованию Фортранов. Конечно же, IBM, а особенно Джон Бэкус, настаивали на том, что Фортран должен быть именно таким, как у них. И им это удалось.

Фортран-66

Стандарт недействующий, признан устаревшим. Это, фактически, стандартизованный Фортран II/Фортран IV фирмы IBM. В сам стандарт ничего нового, по сравнению с этой IBM-овской версией, не ввели. Стандарт на основе Фортрана II получил наименование BASIC FORTRAN (этот стандарт принят в 1966 году), а доработанный стандарт, который впоследствии и назвали Фортран-66, был сделан на основе Фортран IV в 1972 году.

Что в нём стандартизировано (основные моменты):

- Типы данных: INTEGER – целочисленный, REAL – с плавающей точкой одинарной точности, DOUBLE PRECISION – с плавающей точкой двойной точности, COMPLEX – комплексный, LOGICAL – логический, CHARACTER – символьный. Этот набор типов действует до сих пор.

*À la guerre
comme
à la guerre!*

• В дополнение к типу появился параметр DIMENSION – массив с размерностью до трёх измерений и в дополнение к объявлению переменных появился модификатор области видимости – COMMON. По умолчанию все переменные имеют область видимости только в текущем контексте, т. е. объявив переменную в своей программе, в подпрограмме вы её уже не увидите. COMMON позволяет распространить область видимости на все дополнительные подпрограммы, которые используются в данной программе.

• Оператор DATA позволяет инициализировать переменные данными в более компактном виде. От себя скажу, что понятнее программу это не делает ☺.

• Отдельные синтаксические единицы для подпрограмм и функций (SUBROUTINE и FUNCTION), которые заканчиваются оператором END. Это сделало большие программы с процедурами\функциями более читаемыми.

• Целых три оператора перехода: GO TO просто так, GO TO назначаемый и GO TO вычисляемый. Если честно – это жесть, ну просто застрелиться и не встать.

• Два вида условных оператора IF: логический (это понятно, просто вычисляется логическое выражение ДА\НЕТ) и вычисляемый. Вычисляемый имеет вид IF (выражение) метка1, метка2, метка3. Если выражение меньше ноля, то осуществляется переход на метку1, если равно нолю, то на метку2, а если больше, то на метку3.

• Операторы ввода\вывода: READ\WRITE.

• Оператор FORMAT, который определяет, в каком виде вводить\выводить данные, если требуется что-то нестандартное.

• Не оператор, но очень полезная в программах штука – комментарии ☺.

От не стандартизированного Фортрана этот вариант отличался не сильно, но теперь все вновь создаваемые компиляторы должны были ему соответствовать, иначе бизнес в гору не пойдёт.

Фортран-77

Стандарт недействующий, признан устаревшим. В этой версии стандарта, который вышел в 1980 году, сделана попытка приблизить Фортран к большей человекообразности. Вот самое интересное:



- Сделана более понятной работа с файлами: `OPEN()` файл открывает, а `CLOSE()` – закрывает. Внутри этих функций введены ряд именованных параметров, определяющих режимы работы с файлами. Файлу присваивался определённый номер. Впрочем, работа с файлами была и с самого начала, только не очень понятная из-за того, что в UNIX файлы и устройства всегда проходили под номерами. И дальше уже все файловые манипуляции проводились с указанием этого номера. При взгляде на текст программы было непонятно, что имеешь дело именно с файлом, а не с устройством.

- Появился специальный оператор вывода на экран – `PRINT`. Если раньше в качестве первого параметра оператора `WRITE` указать звёздочку (`WRITE *,*`) то вывод данных осуществлялся на экран. Теперь можно написать `PRINT *,` каким-то текст, вместо звёздочки можно поставить формат вывода данных, если он должен быть каким-то особым.

- `IF` приобретает человеческий вид в своей логической инкарнации и теперь явно видно, где он начинается, а где заканчивается: `IF ... THEN ... ELSE ... END IF`.

- Количество измерений массива увеличено аж до семи штук (маловато будет – говорят, что наша вселенная родилась двенадцатимерной 😊), а у индексов массива любые ограничения вообще сняты. Конечно, желательно уместиться в оперативную память 😊.

- Добавилось много функций по работе со строками.

На основе этого стандарта в СССР введены два собственных – ГОСТ 23056-78 и ГОСТ 23057-78 [9], соответственно базовый Фортран и расширенный Фортран.

В качестве интересного момента можно добавить, что наряду с «гражданским» стандартом, был принят и «военный» стандарт, как дополнение к основному стандарту. В чём там разница я не вникал, но кому интересно, могут покопаться в первоисточнике [10].

Фортран-90

Стандарт недействующий, признан устаревшим. Принят в 1991 году. Практически была закончена работа над человеческим образом Фортрана – теперь он белый и пушистый. И по сравнению со старым Фортраном – это, пожалуй, новый язык. Из основного:



- Самое востребованное нововведение – оператор, который явно запрещает переменным самотипизироваться (`IMPLICIT NONE`), ранее можно было с помощью `IMPLICIT` определять, с какой буквы начинаются те или иные типы, но лентяи этим не пользовались.

- Очень хорошо добавлена внутренняя функциональность по работе с массивами, позволившая отказаться от большой группы внешних функций типа `BLAS\ LAPACK`. Теперь все арифметические операции можно делать как с переменными, так и с массивами или вперемешку.

- Присваивание по условию – `WHERE`. Возможность интересная, облегчающая код, позволяющая не создавать циклы с проверкой каждой ячейки. Но, конечно, от циклов совсем уйти не удалось, просто они теперь спрятаны в подпол к компилятору.

- Цикл `DO` приобрёл наконец-то человеческий вид. Теперь у него есть начало и явно видимое окончание `DO...END DO` и внутри появилась возможность, как прервать цикл совсем (`EXIT`), так и вернуться, не дойдя до конца, к началу (`CYCLE`).

- Работа с динамическими заранее неопределённой размерности массивами: выделить\забрать память по не сразу узнанному размеру, переопределить размер. До этого пользовались ловким трюком, когда определялся массив с размерностью в одну ячейку, а в программе в него заталкивали столько значений, сколько нужно. Теперь такое безобразие невозможно, стало ясно видно, какой размер у массива.

- Введено понятие модульности программ по типу как в Паскале. В модуле содержится набор процедур\функций

(возможно и с какими-то данными), которые относятся к какой-нибудь одной области. Модуль можно подключать и использовать как целиком, так и отдельные из него функции. В самом модуле переменные могут иметь как общую для всей программы видимость (PUBLIC), так и локальную (PRIVATE).

- Введено то, что в Си было с самого начала – переменная неизвестного типа POINTER. В процессе работы программы его приводят к определённому типу, как только он становится известным. Насколько это чудо природы в Фортране нужно, мне не очень понятно. Т. е. понятно, что может быть изначально неизвестно, какого типа должны быть данные, однако именно в вычислительных программах, на которые заточен Фортран, мне ни разу не встречались случаи, когда тип данных заранее неизвестен. Впрочем, у других такое могло и случаться.

- Для параметров процедур\функций введён дополнительный модификатор INTENT, который определяет, это входной параметр (IN), выходной (OUT) или и то и другое (INOUT).

- Плюс ещё куча всяких вычислительных и справочных функций.

Ещё одна особенность этого, прямо скажем, революционного стандарта – сделана попытка избавится от пресловутого GO TO вообще. Дело в том, что иногда возникает ситуация с многочисленными вложенными циклами, когда в самом внутреннем цикле после проверки условия выясняется, что дальнейшее циклирование уже нет нужды. Использование стандартного EXIT позволяет выйти только из текущего цикла. Поэтому в других языках используется GO TO с переходом на метку, которая находится вне этих циклов. А вот в Фортране сделали серьёзный шаг вперёд. От меток, правда избавится, не удалось, но теперь метку можно привязать к циклу, а в команде EXIT прописать, цикл с какой меткой имеется в виду, вот пример:

```
Метка1: DO i=1, 100
Метка2: DO j=1, 100
Метка3: DO k=1, 100
        ! Что-то вычисляем
```

```
        ! Проверяем условие выхода
        IF (Условие) THEN
```

```

    EXIT Метка1 ! Указание, какой
    ! именно цикл мы
    ! покидаем

        END IF
    END DO Метка3
    END DO Метка2
    END DO Метка1

```

Конечно, описание циклов стало несколько многословнее, но сам текст программы более логичен и более понятен.

Фортран-95

Стандарт недействующий, признан устаревшим. Пожалуй, именно с этого стандарта не стало большого количества вносимых изменений в новых стандартах. Всё основное уже практически сделано, дальше пошли бантики и кружева. Из интересного:

- Введён новый оператор FORALL для упрощения работы с массивами.
- Для пользовательских процедур можно добавлять признак PURE, который указывает компилятору избавить эти процедуры от проверок во время работы программы. Другими словами, PURE – это процедура без побочных эффектов. Таким образом, увеличивается скорость работы программы.
- Динамические массивы автоматически освобождаются (если, конечно, для них выделялась память) при выходе программы из блока видимости таких массивов.



Именно с этого года, из стандарта начали удалять устаревшие возможности. Удаление организовано таким образом. К примеру, в Фортране-90 какие-то возможности объявлены устаревшими, но из стандарта они не выводятся и пока остаются в действии. В следующем стандарте Фортран-95 устаревшие возможности из стандарта убираются. Итак, были убраны:

- В циклах с заранее определённым количеством повторений (DO I=начало, конец) тип переменной итерации цикла теперь не может быть дробным (раньше это позволялось).

- К END IF нельзя перейти снаружи блока. Действительно, практического смысла в пропуске условия и любых действий по этому условию нет никакого.

- Убраны команды ASSIGN и PAUSE. ASSIGN использовалась для присвоения номера метки какой-либо переменной. Однако начиная с Фортрана-90 ценность метки в программах, можно сказать, сведена практически к нулю. Так что удаление этого оператора логично, но пропадает возможность трансляция совсем уж старых программ современными компиляторами. Ну а PAUSE – это пауза в программе и есть.

Фортран-2003

А вот с этого момента пошли стандарты действующие. Точнее говоря, если вы хотите создать свой компилятор, то можете опираться на любой стандарт, начиная с этого.

Il est ais  de reprendre et difficile de faire mieux



Введён в 2004 году. В этом стандарте самым сногшибательным новшеством является введение ООП. Всё остальное, по сравнению с ним – сущие пустяки ☺.

Фортран-2008

Введён в 2010 году. Последний из ныне действующих стандартов. Вот основные нововведения:

- Пожалуй, самое главное – введено понятие COARRAY для параллельного программирования. Данные могут теперь быть не только массивом в одной программе, но и считаться массивом в разных экземплярах программы (т. е. когда одна и та же программа запущена на нескольких машинах и каждая обрабатывает свою порцию единых данных) и обмен данными между этими программами идёт так же, как в простом массиве. Конечно, для работы такого массива нужна платформа межпроцессного общения. Обычно это MPI.

- Введено некоторое количество встроенных функций, которые раньше были из внешних библиотек – функции Бессе-



Autres temps, autres m urs

ля, гамма-функция, обработки ошибок, работы с битами и ещё кое-чего научное.

- У массива теперь может быть до 15 измерений. Ура, товарищи! Появилась возможность просчёта нашей вселенной целиком и полностью. Ведь все знают, что вселенная 12-тимерная. Буквально уже завтра физики предложат вариант многомерных перемещений, и до звезды Альфа Центавра можно будет добираться не за четыре года, а в течение каких-нибудь пары секунд.

- Улучшено взаимодействие с языком Си.

- Ещё одно интересное нововведение – новая модификация цикла DO: DO CONCURRENT. По заявлению разработчиков эта штука должна помочь при оптимизации цикла компилятором, когда присваивание значений внутри цикла не зависит от порядка присваивания. Правда по отзывам программистов, когда оптимизация такого цикла у компилятора не получается, список ограничений довольно длинный. Так что если хотите использовать такой цикл, то нужно обязательно его сравнить по скорости работы с обычным циклом.

Фортран-2018

Введён буквально недавно. В предварительном варианте назывался Фортран-2015, но к 15-му году подготовить его не успели.

Если говорить совсем коротко, то основные нововведения по двум пунктам:

- Дальнейшее совмещение с языком Си.
- Добавление финтифлюшек для параллельного программирования COARRAY.

Устаревшее:

- Удалён вычисляемый IF, который пометили как устаревший ещё в Фортране-90. Так что, как только компиляторы обновятся до поддержки Фортрана-2018, довольно большое количество текстов старых программ перестанет компилироваться, т. к. в старые времена вычисляемый IF использовался довольно регулярно. Теперь остался только логический IF.

- Так же запрещён оператор DO без явного указания конца цикла в виде END DO. Таким образом, всё, что писалось до Фортрана-90, перестанет компилироваться. Не исключено, что



Qui Cherche Trouve

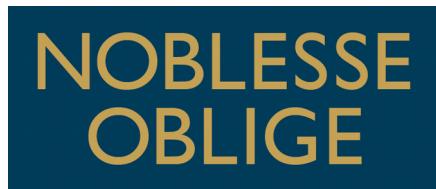
производители компиляторов введут какой-нибудь ключ совместимости со старыми текстами. Так же не исключено, что производители компиляторов выделят Фортран-77 в отдельную ветку без дальнейшей модификации.

- Объявлена устаревшей конструкция FORALL. Объясняется это тем, что она не оправдала возложенное на неё надежды и признана избыточной.

Я думаю, что это второй, после Фортрана-90, революционный стандарт, т. к. в нём отменяются основополагающих два кита старого синтаксиса.

Фортран-202x

Смотрим не в завтра, а в послезавтра... В июне 2017 года началось планирование разработки стандарта Фортран-2020, который обещают сдать на всеобщее обозрение в июне 2020



года. Однако в связи с тем, что стандарт 2015 съехал на 2018-ый год, название этого стандарта тоже поменялось и стало 202x, что вполне разумно.

Пока что планы стандарта полностью не определены. Вот список работы комитета:

- Начало планирования дальнейших возможностей ревизии: 2017-06.
- Выбор вопросов, которые требуют изучения: 2018-06.
- Предварительный выбор технического контента: 2019-08.
- Окончательный выбор технического контента: 2020-06

На сегодняшний день известно:

- Будет дальнейшее улучшение «ООП совместимости» - джинерики и шаблоны, как в Си++.
- Возможно появится автоматическое выделение памяти под «безразмерные» массивы, когда программа попытается заполнить ячейки данными. Эта штука мне кажется очень хорошей, поскольку сильно упростит код программы.
- Обработка исключительных ситуаций примет блочный вид, как это делается во многих современных языках. Т.е. потенциально опасный код будет заключаться в специальную

- программную конструкцию, типа `try ... catch` в Си++ или `Try ... Except` в ObjectPascal.
- Возможно появится новый подтип — беззнаковое целое.
 - Облегчение использования «безразмерных» строк, которые в языке Си оканчиваются на специальный символ `NULL`.
 - И многое другое...

Почитать о планах на новый стандарт можно здесь [\[\]](#).

О стандартах, в общем и целом

Начиная со стандарта Фортран-95, проводятся всевозможные улучшения по взаимодействию с языком Си. Самого меня это пока мало волнует, потому что и просто с Си работаю редко, и мне практически совсем не требуются вызовы синых функций из моих чисто вычислительных программ. Но кому интересно – имейте в виду. Так же, начиная с Фортрана-90, каждый раз прибавляется какое-то количество функций взаимодействия с операционной системой, чтобы можно было не обращаться к библиотечным синым функциям непосредственно.

Если вы планируете писать программы на Фортране, то следует писать их в «свободной» форме, которая введена в Фортране-90. Это делает код программ ближе и понятней для тех, кто привык программировать на других языках. Однако, если вы хотите включить в тексты своих программ какие-то процедуры\функции, которые писались в старом, «фиксированном» формате, то компилятор их без возражения проглотит. Проблема может быть только с совсем старыми тестами программ, которые когда-то писались на диалектах Фортран II или IV в начале 60-ых годов. Однако и в таких случаях бывает нужно исправить буквально пару строк в тех местах, где встречаются отменённые операторы.

На самом деле, несмотря на то, что постоянно в стандартах появляются сообщения, что те или иные операторы объявлены как устаревшие и, по идее, в следующем стандарте их должны были бы отменить, самих отменённых операторов очень мало. Такая ситуация сложилась из-за того, что подавляющая часть вычислительных функций писалось ещё в 60-ых годах и переписывать их на современный синтаксис никто не

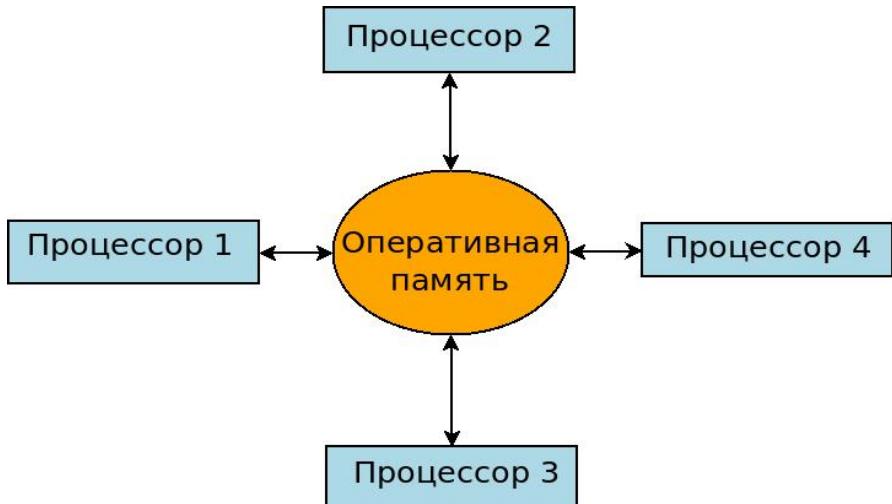
торопится. А что? Всё работает же и ладно, тем более, что алгоритм расчёта нисколько не поменялся...

1.3 «Параллельные» стандарты – OpenMP и MPI

*Из двух ошибок выбирают: сделать их последовательно или параллельно.
Один известный философ, попавший в «Матросскую тишину»*

Эти стандарты предназначены для создания программ, которые позволяют распараллеливать обработку больших объёмов данных, тем самым уменьшая время их обработки. Формально они не являются стандартами самого Фортрана. Однако в этих стандартах прямо оговорена разработка программ для языков Си и Фортран, поэтому можно сказать, что с современным Фортраном они очень тесно связаны. Рассмотрим их совсем чуть-чуть.

OpenMP – это наиболее лёгкий «параллелизм» в своём использовании. Распараллеливание вычислений происходит в виде потоков для многоядерных\многопроцессорных машин с общей памятью.



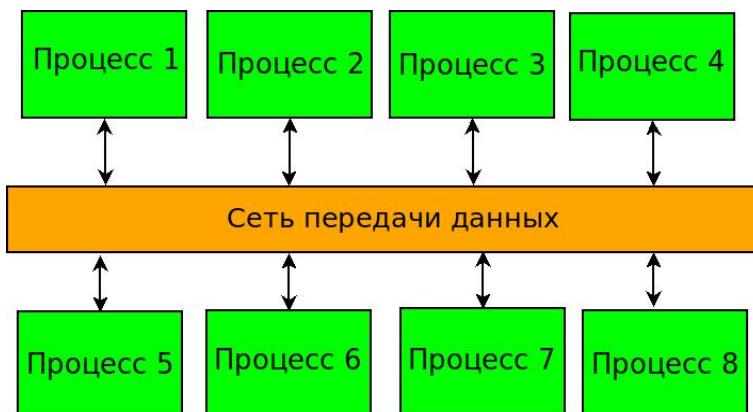
OpenMP – несколько процессоров работают с общей памятью

Такое программирование аналогично использованию отдельных потоков в рамках одного процесса встроенными функциями того или иного языка. Однако в отличие от этих встроенных функций, реализуется с помощью директив компилятору и кое-каких служебных функций. Это более высокоравневое многопоточное программирование, т. к. всеми внутренними потребностями для управления потоками заведует компилятор, а не программист. Таким образом, время подготовки программы значительно сокращается. Ещё одна «правильная» особенность – потоков создаётся не более, чем существует вычислительных ядер в системе, таким образом устраняется проблема временных затрат на переключение между потоками, если их сильно много.

У этого метода распараллеливания есть ограничение, потому что в современных процессорах ядер сейчас встраивается не сильно много. Это связано с технологическими особенностями производства – дело в том, что наряду с вычислительными устройствами в процессоры закладываются и управляющие, а управляющие намного сложнее вычислительных. Если сравнить обычные универсальные процессоры, к примеру, с видеопроцессорами, то обнаружится, что у самого крутого универсального процессора содержится до 28 ядер, а если учесть ещё и Hyper-Threading (т. е. аппаратную многопоточность процессоров), то это количество можно умножить на два, т. к. на два вычислительных ядра приходится всего одно управляющее.

Для видеопроцессоров, где управляющих систем чрезвычайно мало, количество вычислительных ядер может быть и несколько сотен. И уже довольно давно эти видеоядры используют для вычислений, конечно, если они не используются по прямому назначению. Поэтому возможности распараллеливания в рамках одной системы довольно ограничены. Вот только единого стандарта для видеопроцессоров пока ещё не придумали, поэтому у NVIDIA свои многопоточные функции, у AMD свои, а у INTEL – свои. Из-за этого написать одну программу, которая бы работала без изменения кода с любым видеопроцессором, как с обычным вычислительным устройством, пока затруднительно. Хотя попытки такие уже делаются, например, в проекте OpenACC, где нет разницы между параллельной работой на центральном процессоре или на видеопроцессоре.

MPI – это уже другой тип распараллеливания, когда несколько отдельных процессов со своими ресурсами обрабатывают каждый свою часть общего объёма данных, при этом каждый кусок данных находится в отдельном вычислительном узле. Здесь одна и та же программа запускается на нескольких, заранее определённых машинах, каждая со своими процессорами и своей памятью. В этом случае количество параллельных программ может быть и бесконечно много, главное, чтобы они успевали общаться друг с другом в разумный период времени. Тогда распараллеливание можно проводить как на уровне данных, когда один массив данных делится на количество его обрабатывающих процессов, так и на уровне функций, когда каждый процесс занят обработкой отдельной подзадачи со своими данными. Таким образом, распараллеливание может быть очень гибким.



Каждый процесс имеет свой процессор и память

Не обязательно запускать каждый процесс на отдельном компьютере, операционная система всегда выделяет каждому процессу своё адресное пространство и, по возможности, отдельный процессор или ядро. Но при работе на одном компьютере намного больше опасности, что процессы сожрут всю память и вместо увеличения скорости будет её уменьшение.

Конечно же, никто не мешает совмещать эти два метода между собой, что позволяет дополнительно ускорить выполнение расчётов.

Надо сказать, что у этих стандартов особенности работы немного различаются. OpenMP направлен на распараллеливание именно данных, упрощённо говоря, в цикле происходит обработка какого-либо массива и каждую часть массива обрабатывает отдельный поток. MPI – это распараллеливатель более универсальный – с помощью него можно распараллеливать как на уровне данных, так и на уровне функций, когда каждый процесс обрабатывает какую-то функцию, а не просто свою часть массива данных.

И у того и у другого стандарта есть как преимущества, так и недостатки. У OpenMP преимущества в скорости обмена данными между потоками. Недостаток – ограниченность в распараллеливании. Достоинство у MPI в теоретически безграничном распараллеливании, но вот обмен данными между процессами более долгий.

У OpenMP есть два специфических косяка, которые могут сильно осложнить жизнь любому программисту:

1. Попытка нескольких потоков овладеть одним единственным ресурсом системы, который под многопоточную работу вовсе не рассчитывался. Может создаться ситуация взаимной блокировки, когда один поток блокировал один ресурс, а второй поток – второй ресурс. И каждый из них принимается ожидать освобождения другого ресурса, который заблокирован его «собратом». Естественно, ожидать не только увеличения производительности, но и вообще окончания работы программы за время жизни нашей Вселенной становится делом бессмысленным.



Для доступа к ресурсам, которые обязательно нужно использовать единолично используют специальные функции блокировки. Казалось бы, удобно – заблокировал ресурс и пользуйся им, больше никто не помешает. Здесь и возникает взаимная блокировка. Классический шахматный пат, когда никто больше не может ничего сделать. Причём, в одной и той же программе такая ситуация может возникнуть, а может и не возникнуть просто из-за того, что каждый поток действует не всегда одно и то же время. Из-за разных причин, время выпол-

нения может быть разным. Такую ситуацию отследить бывает сложно. Поэтому блокировки ресурсов нужно проводить только тогда, когда без этого никак не обойтись и на как можно более короткий период.



2. Попытка нескольких потоков забраться в один и тот же ресурс, который в принципе против этого не возражает (например, текстовый файл), но вот результатом будет какой-нибудь дикий винегрет.

Лирическое отступление про сложность поиска ошибки в многопоточных программах

Сразу оговорюсь, что в приведённом мною ниже жизненном примере причина была не программная, а железная. Однако проблема возникла именно из-за многопоточности ☺.

Случилась эта леденящая душу история давным-давно, ещё в прошлом веке, когда компьютеры были большие, как пирамида Хеопса и занимали целый спортзал, в котором запросто можно было бы играть в баскетбол, если бы зал был пустой. И вообще, компьютеры звались не компьютерами, а Электронно-Вычислительными Машинами (именно так, с большой буквы) или сокращённо ЭВМ.

В одной крупной организации купили мощную, современную по тем временам ЭВМ, техники её смонтировали, контролёры проверили – всё хорошо, всё работает. Пришла пора давать машине настоящую работу. И что бы вы думали? Машина иногда считает правильно и выдаёт на-гора то, что надо, а иногда неправильно и выдаёт чёрте что, и сбоку бантик. Местные умельцы чесали репу, чесали, придумать ничего не смогли. Пришлось вызывать представителя завода-изготовителя.

Представители приехали быстро, видимо нигде больше в это время ихние машины не ломались. Провели подробнейшие и научно разработанные тесты – ЭВМ работает не просто хорошо, а без преувеличения, идеально. Опять запустили настоящую работу. Мать честная, опять посыпались проблемы! Провели тесты – всё проходит без сучка и задоринки. Начали обычную работу – опять ужасы нашего городка. Никто ничего не понимает. Репу чешут все, даже местная уборщица тётя Глаша.

Вот, кстати, тётя Глаша и показала, где пряталась проблема ☺. «Сынки, – говорит. – Я тут, когда мою пол, и нечаянно задеваю шваброй вот этот шкаф, он, по-моему, слегка нетрезвый». Все тут же начали трогать руками шкаф, и выяснилось, что у него была повреждена одна ножка. А шкаф этот содержал пять дисководов, каждый по 90 килограмм весом, и когда работали все пять одновременно – слегка покачивался. Если работали четыре – стоял не двигаясь. В общем, это покачивание вредно оказывалось на головках дисководов, мешая пра-

вильно читать данные. Как только ножку починили, машина заработала на ура.

Вот такая вот была проблема из-за многопоточности 😊...

1.4 Фортран в СССР

*СССР – это 22 400 000 квадратных километров
без единой рекламы кока-колы!
Габриэль Гарсиа Маркес*

*Здесь еще большое поле работы.
Иосиф Виссарионович Сталин*



Николай Николаевич
Говорун, инициатор
использования
Фортрана в СССР

физики давали задание математикам, те писали программы, потом их совместно тестировали, а в CERN'е физики сами писали программы, без помощи математиков, что существенно экономило время.

Собственный транслятор Фортрана совместно с операционной системой начали создавать в специально со-

Появлению Фортрана в СССР способствовали физики. В 60-ых годах более перспективным языком программирования считался Алгол. Однако физики, которые работали в Объединённом институте ядерных исследований (это был открытый, в отличие от Курчатовского института, международный исследовательский проект в области ядерной физики), в то время очень плотно сотрудничали со швейцарским CERN'ом, в котором программы писались как раз на Фортране. Решающим фактором в выборе именно этого языка послужило то, что у нас



Советская суперЭВМ БЭСМ-6, на которой рассчитывались в 70-е годы прошлого века все запуски космических ракет, при помощи программ на Фортране [44]

зданной Лаборатории вычислительной техники и автоматизации (ЛВТА). Однажды, начальник лаборатории Н. Н. Говорун вернулся из командировки в CERN с большим количеством программ и описаний Фортрана, после чего была создана рабочая группа по созданию собственного транслятора.

После того, как основная часть проектных работ была выполнена, транслятор начали отлаживать на машине БЭСМ-6. Эта машина была создана буквально только что в Институте Точной Механики и Вычислительной Техники в Москве, под руководством С. А. Лебедева и имела на тот момент ужасающе высокую скорость вычислений. Кстати, на рисунке представлена не одна ЭВМ, а две – на втором плане виднеются два пульта управления. Эти машины работали параллельно в локальной сети.

Мало кто верил, что у не математиков такой серьёзный проект удастся. Что интересно, отладку проводили с участием довольно таки старенькой даже на тот момент американской машины CDC-1604 фирмы CRAY, которую купили, видимо потому (а новые машины покупать было нельзя, запрет американцев), что её уже давно сняли с производства. Получился



Сергей Алексеевич Лебедев
– создатель советской
суперЭВМ БЭСМ-6



Американский суперкомпьютер CDC-1604 фирмы CRAY

очень крутой транслятор, который назвали ФОРТРАН-ДУБНА, который потом разошёлся по всей стране.

Стоит отметить, что вначале 70-ых Фортран существовал и на только что появившихся тогда машинах серии ЕС, однако наши разработчики здесь были не совсем при делах, т. к. ЕС'ка была позаимствована без спросу у американцев в виде IBM 360 и, чтобы время не терять, с полным комплектом программного обеспечения. Впрочем, говорить о том, что наши разработчики совсем уж не прикладывали руку к ЕС'кому Фортрану тоже нельзя. ЕС вовсе не была идентичной IBM 360, кое-что пришлось изменять под нашу советскую реальность. Поэтому, как операционную систему для ЕС, так и языки программирования приходилось слегка переделывать.

Трансляторы Фортрана делались так же и для машин СМ и Эльбрус. Однако на них они использовались ни шатко, ни валко, т. к. эти машины в основном служили несколько другим целям: СМ – для экономических задач, для чего применялся сынок Фортрана Бэйсик, а Эльбрус – больше относился к военке и на нём чаще применяли Алгол.

Если говорить только о наших разработках транслятора, то именно наши Фортраны существовали для машин БЭСМ-6, Минск-2 (-22, -32) и Эльбрус. Для остальных машин Фортран позаимствован у американцев.

На сегодняшний день количество организаций в России использующих Фортран и маленько его пропагандирующих крайне невелико – несколько федеральных университетов, например, Нижегородский и парочка академических институтов, например, институт прикладной математики имени Келдыша. Печаль, однако... Подавляющее большинство предпочитают пользоваться Си, подключая внешние вычислительные библиотеки и увязая в пучине отладки, вместо того, чтобы просто реализовать алгоритм на Фортране и радоваться жизни.

Небольшое лирическое отступление – за что в СССР любили вычислительную технику

Хочу отметить, что работа в сфере связанной с вычислительной техникой, наряду с военкой, была наиболее любимой в СССР. Кроме того, что это была интересная интеллектуальная работа, существовал и ещё один фактор, резко повышавший популярность ЭВМ. Дело в том, что сия техника не могла долго жить без профилактических мероприятий. А эти мероприятия проходили при непосредственном участии этилового спирта. Причём не просто какого-либо технаря, а спирта высшей очистки. Спиртом должны были протирать многочисленные

вычислительные детальки и вообще всё оборудование, которое было связано с ЭВМ, поэтому никаких сивушных масел в спирте быть не должно. К примеру, на заводе, где я работал когда-то, вычислительный центр отнюдь не был велик и сильно уж насыщен техникой. Но давайте посмотрим, сколько по советским нормативам полагалось для него спирта, см. таблицу.

**Нормативы расхода этилового спирта по ГОСТ 18300-72
на протирку при эксплуатации электронных управляющих
вычислительных комплексов, вычислительных машин
общего назначения и дополнительных устройств к ним**

№	Название	Количество, шт.	Расход спирта на одно обслуживание, л.	Количество обслуживаний в год	Итого, л.
1	ЭВМ СМ-1600	2	3,943	4	15,886
2	— НМЛ к нему	2	0,01	250	7,50
3	— НМД к нему	8	0,375	4	12,00
4	ЭВМ СМ-1425	3	0,555	4	6,66
5	АЦПУ СМ-6315	1	0,035	12	0,42
6	Видеотерминал ВТА2000	12	0,005	250	15,00
ВСЕГО:					57,466

Увы, ни разу не видел, чтобы наш начальник вычислительного центра получал именно такой объём спирта. Обычно со склада давали 1,5 литра, не более. Уж даже и не знаю, в чём была проблема. Видимо происходила естественная усушка этой в высшей степени летучей жидкости из-за того, что пробки в складских баках были завинчены не очень плотно ☺. Тем не менее, этого количества хватало на профилактические работы — мы к материалам всегда относились крайне бережно ☺. И оставалось к празднованию Нового года готовить рябиновую настойку — возле нашего заводоуправления росло несколько деревьев рябины. А чтобы праздник был разнообразнее, приготовлялся второй напиток — «Клюковка», уже настоящий на ягоде клюкве ☺.

Отдельно пару слов про компьютерные сети. Американцы утверждают, что первая сеть была создана именно у них в 1969 году, в Калифорнийском университете. Одни раз я вам уже доказал, что американцы — вруны те ещё. Если им верить — то обязательно останешься без портока ☺. Впервые докладную записку об использовании ЭВМ для управления движением ракетой в реальном времени, подал в президиум Академии Наук СССР профессор Лебедев в 1951 году, ещё задолго до того, как усился изобретать БЭСМ-6. Он прекрасно понимал: всё, что мы создадим в то время будет иметь низкую скорость вычисления. Поэтому большие и сложные задачи нужно было делить на подзадачи и распределять их по нескольким вычислительным узлам. Увы, в то время мы только-только начали производить АВМ (Аналоговые Вычислительные Машины), которые для ра-

боты в сети вообще не годились. Ну, сами подумайте – из одной машины выходит 5 вольт, а через 100 км стоит другая машина, и туда из-за сопротивления проводов уже приходит 0,5 вольт, потому что вряд ли кто-то для такой сети будет делать золотые провода. Да и пролежат они недолго ☺... А цифровые машины пока ещё не вышли за пределы бумажных рисунков и предварительных расчётов.

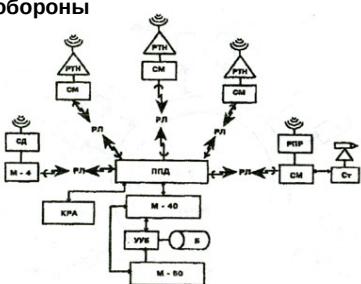


Григорий
Васильевич
Кисунько –
один из
разработчиков
сетевой
системы
противоракетн
ой обороны

В 1956 году в Казахстане, недалеко от озера Балхаш, под руководством выдающегося конструктора, разработчика противоракетных систем В. Г. Кисунько, начал создаваться специальный полигон для испытания системы противоракетной обороны, расчёт работы которой в реальном времени должны делать с десяток ЭВМ объединённые в одну сеть для совместной работы.

Примерная схема работы такой системы представлена на рисунке ниже.

Хочу обратить внимание, что для связи ЭВМ с передвижными стартовыми



РТН – радиолокаторы точного наведения
 СМ – специальные вычислительные машины
 СД – станция дальнего обнаружения
 РПР – радиолокатор противоракеты (передача сигналов на противоракету)
 СТ – мобильная стартовая установка противоракет
 ППД – процессор приема и передачи данных
 М-4, М-40 и М-50 – электронные вычислительные машины
 Б – запоминающее устройство на магнитном барабане
 УУБ – устройство управления барабаном
 КРА – контрольно-регистрирующая аппаратура

Схема вычислительной сети советской экспериментальной ПРО, развернутой в 1959-1960 годах

комплексами использовалась беспроводная высокочастотная сеть (Wi-Fi, ау! ☺). Расстояние между вычислительными машинами было до нескольких сотен километров. Всеволод Бурцев, ученик Лебедева, который руководил разработкой вычислительного сегмента, по поводу скорости приёма-передачи информации по сети вспоминал: «Общий темп поступления информации через радиорелейные линии превышал 1 МГц». В пересказе на

современные термины это можно приблизительно оценить, как 1 Мбит/с, что очень неплохо для беспроводных сетей даже по современным меркам. Американская ARPANET имела скорость по проводам 56 кбит/с. Почувствуйте разницу! ☺ Wi-Fi в СССР в 1960 году!

Идеи использования единой компьютерной сети на всей территории СССР появились в 1962 году. Руководитель отдела технической физики киевского Института физики АН УССР Александр Харкевич опубликовал в журнале «Коммунист» статью, в которой он выдвигал идею создания единой компьютерной сети. Он проводил параллели между такой сетью и сетями транспортировки грузов или электроэнергии и предлагал создать Единую общегосударственную систему передачи информации, которая осуществляла бы работу на основе уже имеющихся каналов электрической связи.

В статье идея описывалась довольно подробно и самое интересное, что предлагалось учитывать возможность выхода из строя, как отдельных узлов, так и линий передачи данных, автоматически переключая передачу на другие линии\узлы до восстановления повреждённых. Интернет нынче работает точно так же, не правда ли? ☺

В том же 1962 году директор киевского Института кибернетики Виктор Глушков предложил председателю Совета Министров СССР Косыгину создать единую управляющую сеть, которая бы позволила в полной мере реализовать единое планирование экономики по всей стране. Идея строилась на создании в крупных городах больших вычислительных центров, куда бы стекалась информация с мест, с отдельных терминалов и с малых вычислительных машин.

В качестве малой вычислительной машины можно было использовать первую советскую персоналку «МИР-1», проде-



Первая советская персональная ЭВМ
«МИР-1»

монстрированную в 1967 году на выставке в Лондоне. Меду прочим её за бешеные бабки купила фирма IBM ®.

Увы, после тщательных расчётов оказалось, что компьютерное обеспечение плановой экономики в СССР оказалось очень дорогим. Для сравнения цен, ориентировочно 1986 год:

- пирожок с повидлом – 0,06 руб.
- беляш с мясом – 0,11 руб.
- сахар-рафинад 1 кг в пачке – 1,04 руб.
- бутылка вина «Клюквенное» – 1,05 руб.
- колбаса варёная «Любительская» – 2,90 руб.
- утюг – 5,00 руб.
- микропроцессор KP580ВМ80А – 5,00 руб.
- кофеварка электрическая – 10,00 руб.
- электронная игра «Ну, погоди!» – 25,00 руб.
- микрокалькулятор «Электроника С3-33» – 35,00 руб.
- авиабилет Москва-Красноярск на Ту-154 или Ил-62 – 68,00 руб.
- джинсы мужские «RIFLE», Италия – 90,00 руб.
- мопед «Рига-12» – 183,00 руб.
- средняя зарплата ведущего инженера – 190,00 руб.
- люстра хрустальная – 240,00 руб.
- денежное довольствие лейтенанта танковых войск – 285,00 руб.
- денежное довольствие подполковника службы тыла – 300,00 руб.
- цветной телевизор «Рубин-Ц260» – 750,00 руб.
- зарплата ministra СССР – 800,00 руб.
- АЦПУ СМ-6315 – 14 836,00 руб.

АЦПУ это конечно не ЭВМ, но и по этой цене можно представить, каковы были бы затраты на оборудование нескольких тысяч вычислительных центров по всей стране. Для того чтобы увидеть цену техники для среднего вычислительного центра, стоимость АЦПУ можно умножить на 300. Для большого центра – умножить на 1 000. Если говорить о районных вычислительных центрах, где будет стоять только персоналка типа «МИР-1», то там, конечно, такая мощная АЦПУ не нужна. Но и с маленьким АЦПУ, как на рисунке выше, стоимость всё равно будет или лучше сказать была в районе 100 000 рублей. В общем повторилась ситуация с Конрадом Цузе и научным сообществом Германии, только здесь уже невозможность создания мирной компьютерной сети жестко обосновывалось расчётами экономистов. Увы, не военка, а значить и не первой необходимости.

Эх, печалька...