Прообразы программ и проактивные вычисления

Роганов В. А.

НИИ Механики МГУ

МГУ, 5 апреля 2016г.

Аннотация

Программы изначально были придуманы для машин, но хороший программный код сегодня пишется прежде всего для людей. На это противоречие долго не обращали должного внимания; оно маскировалось общим успехом программирования и прогрессом в части эволюции языков высокого уровня.

Пропрограммы – класс объектов, которые наиболее удобны для осмысления человеком и автоматизированного преобразования компьютером, но, тем не менее, все еще легко превращаемых в программы.

Вычислительные устройства в рамках предлагаемого подхода рассматриваются как управляемые динамические системы со свойствами, позволяющими обеспечить корректность результата при организации параллельных вычислениях.

План доклада

Контекст: HPC и передний край IT-индустрии

- HPC сегодня: платформы, тенденции, перспективы
- IoT, IoE: Интернет вещей, Интернет всего ...
- Искуственный Интеллект и успехи нейросетей

Проблема и предлагаемый подход к ее решению

- Качество ПО: формальные и неформальные аспекты
- Филосовская сторона вопроса: причины и условия
- Пропрограммы как решение основной проблемы

Сущности и методы, на которых основывается подход

Будут перечислены новые сущности и алгоритмы, на использовании которых построена практическая реализация разработанного подхода.

Parallella board

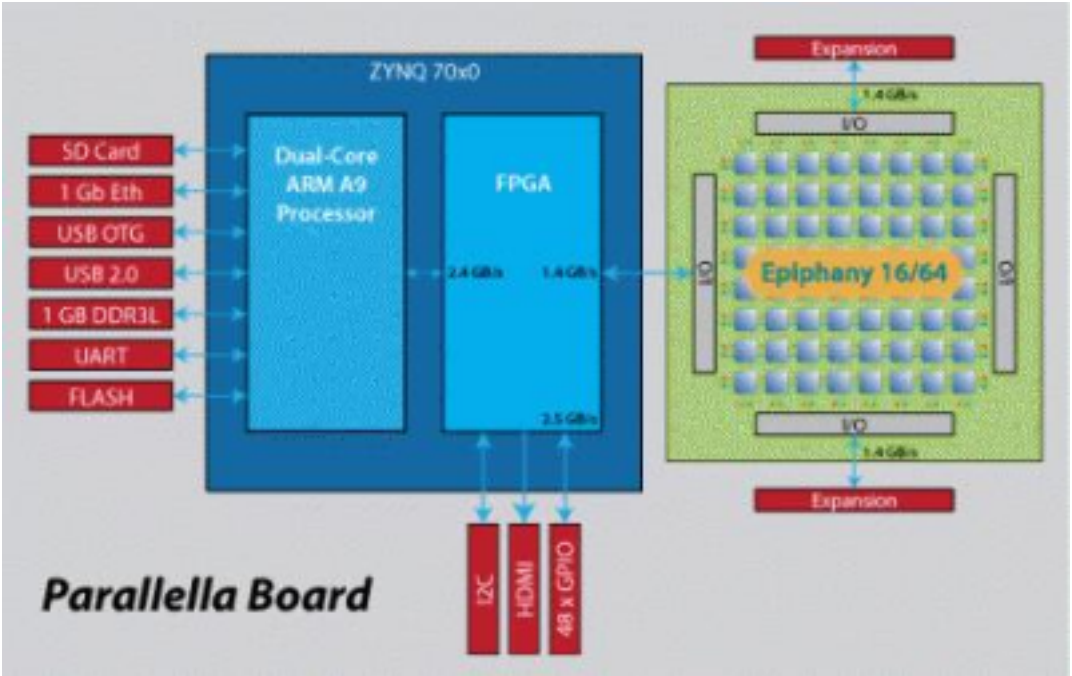


Рис. 1:

Parallella board

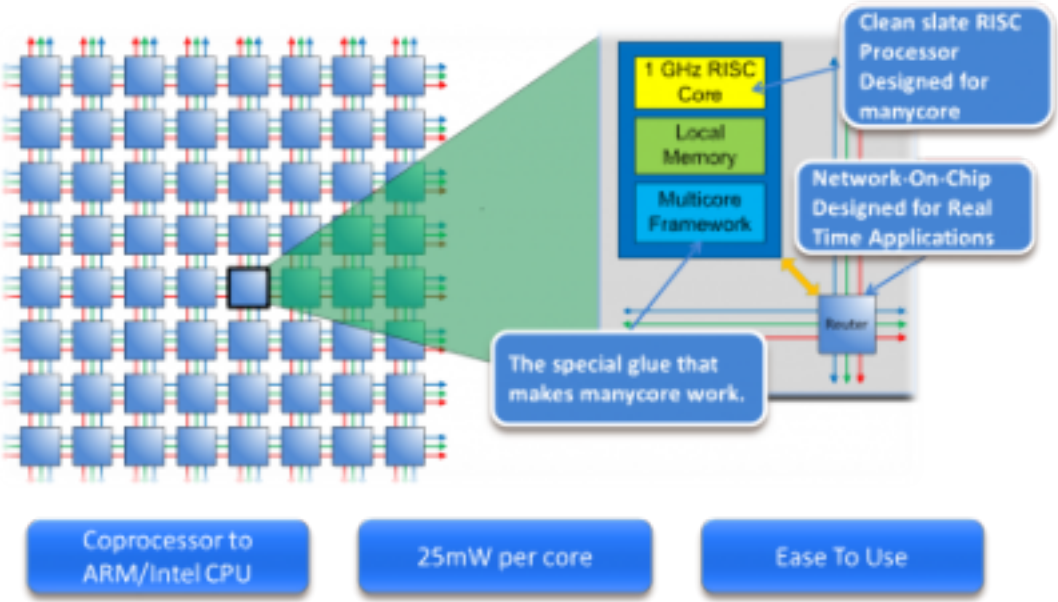
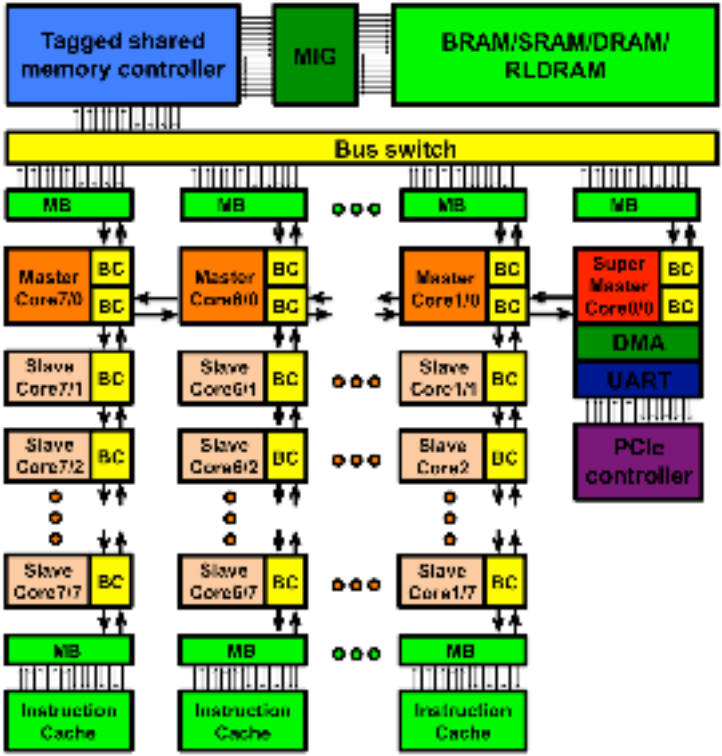


Рис. 2:

MALT System


Tagged shared memory controller
MIG
BRAM/SRAM/DRAM/ RLDRAM
Bus switch
MB
MB
MB
MB
Master Core7/0
BC
BC
Master Core6/0
BC
BC
Master Core1/0
BC
BC
Super Master Core0/0
BC
BC
DMA
UART
PCIe controller
Slave Core7/1
BC
Slave Core6/1
BC
Slave Core1/1
BC
Slave Core7/2
BC
Slave Core6/2
BC
Slave Core2
BC
Slave Core7/7
BC
Slave Core6/7
BC
Slave Core1/7
BC
MB
MB
MB
Instruction Cache
Instruction Cache
Instruction Cache

Примеры из (нашей!) истории

Рис. 4:

Переход к принципиально новому представлению чисел и способу вычислений позволил отечественным суперЭВМ на некоторое время в несколько раз превзойти все известные зарубежные аналоги по производительности. Идеи модулярной арифметики живы и актуальны для HW и по сей день.

Процессоры со сверхнизким энергопотреблением

Рис. 5:

Сверхрегенаритвные устройства со сверхнизким энергопотреблением

Рис. 6:

Сверхрегенератор профессора В.Т.Полякова

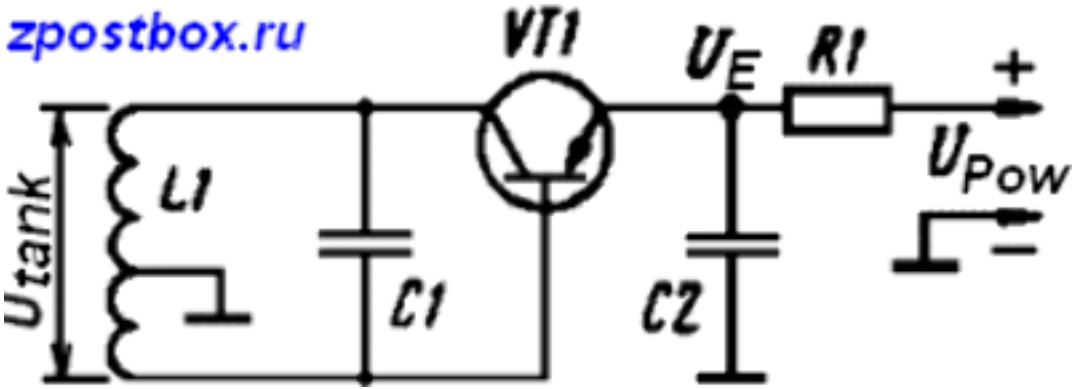
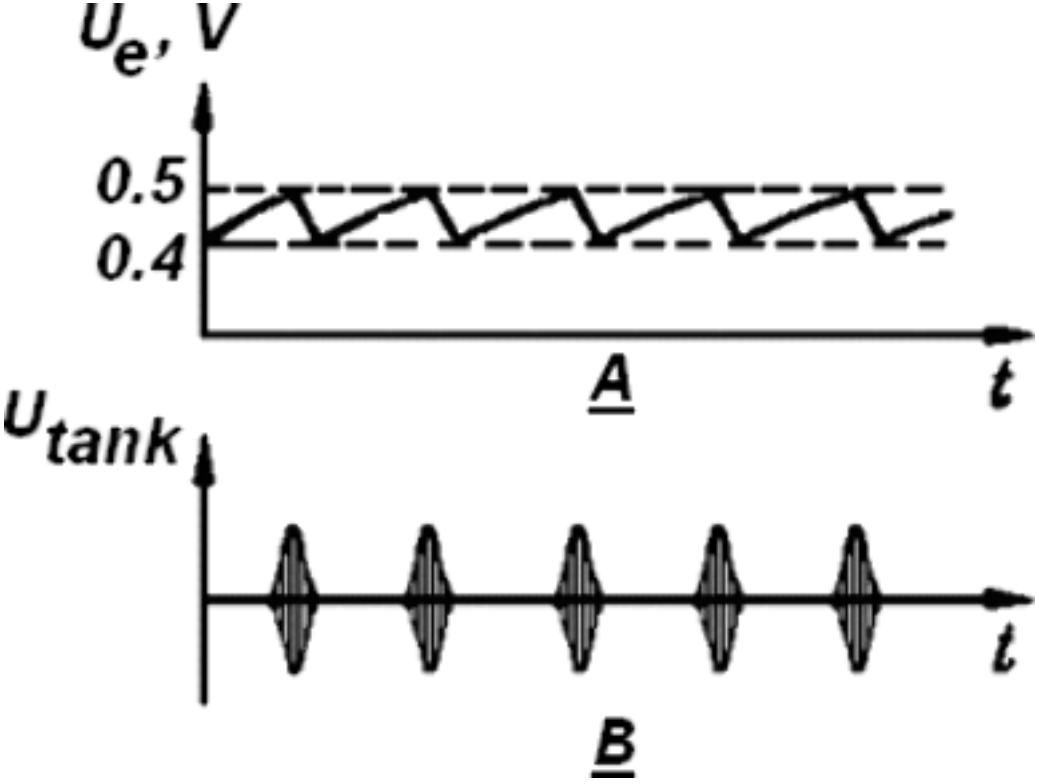


Рис. 7:

Форма сигнала (режим автогашения осцилляций)



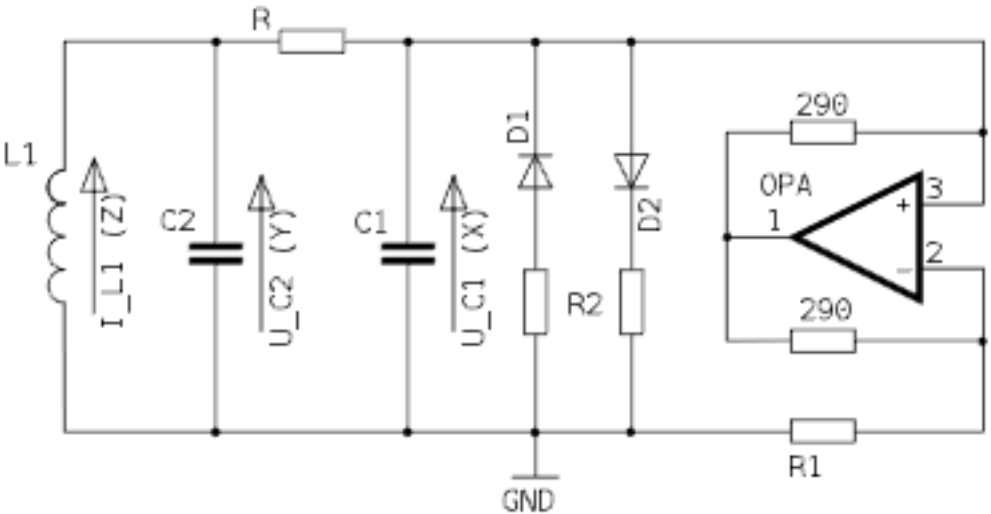


Рис. 9:

Аттрактор типа двойной завиток

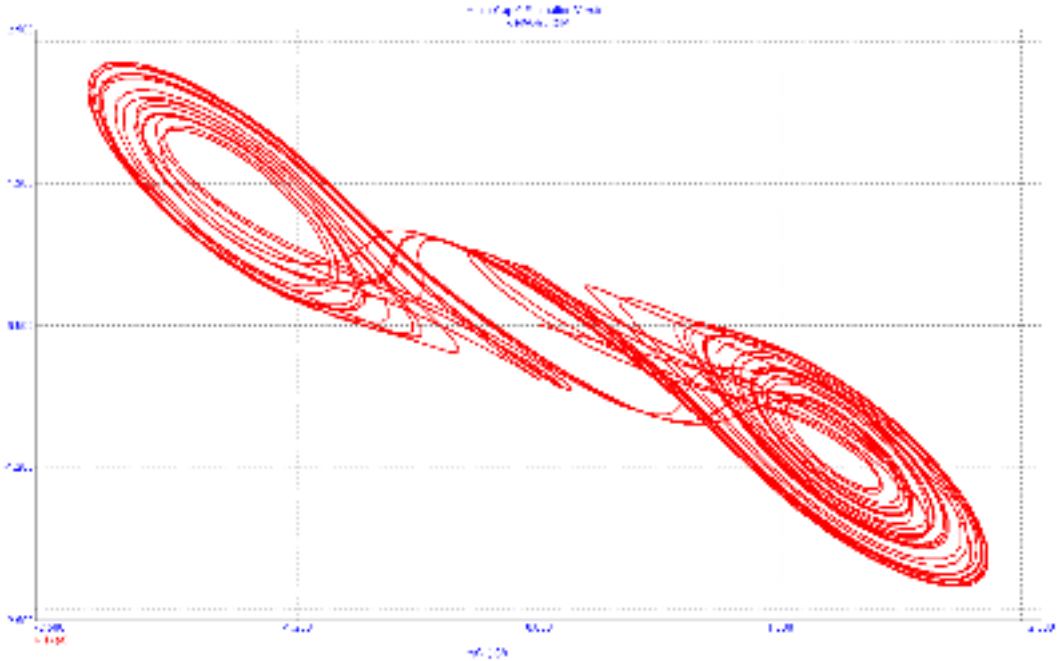
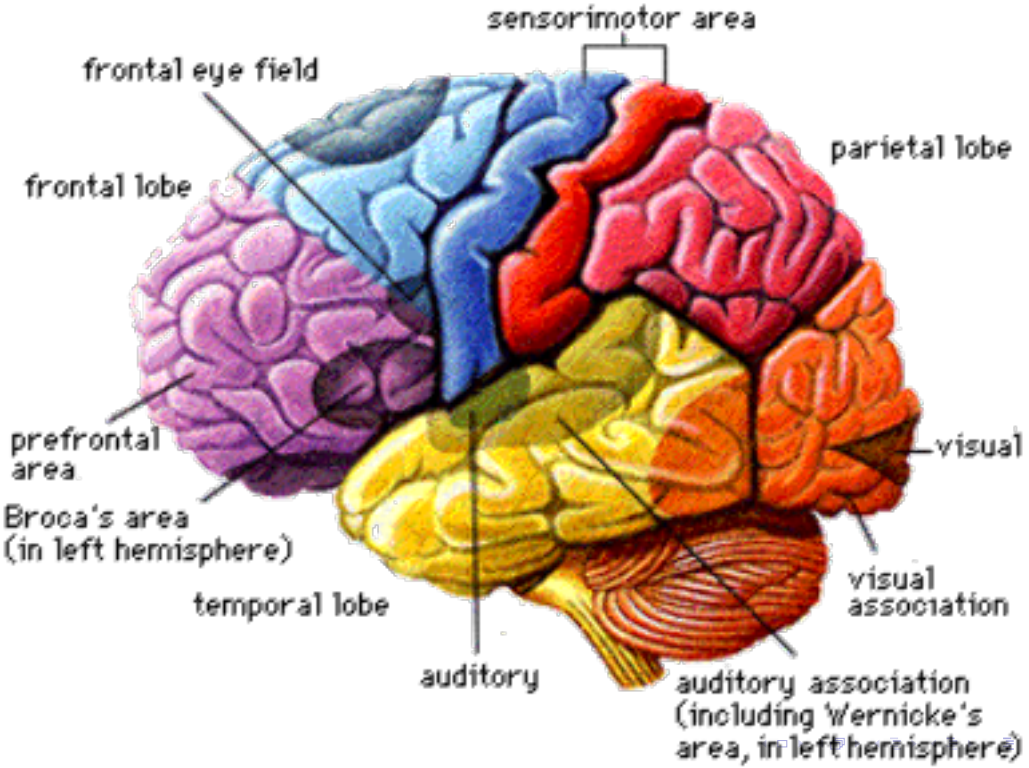


Рис. 10:

Успехи нейросетевых архитектур



Нейронные сети

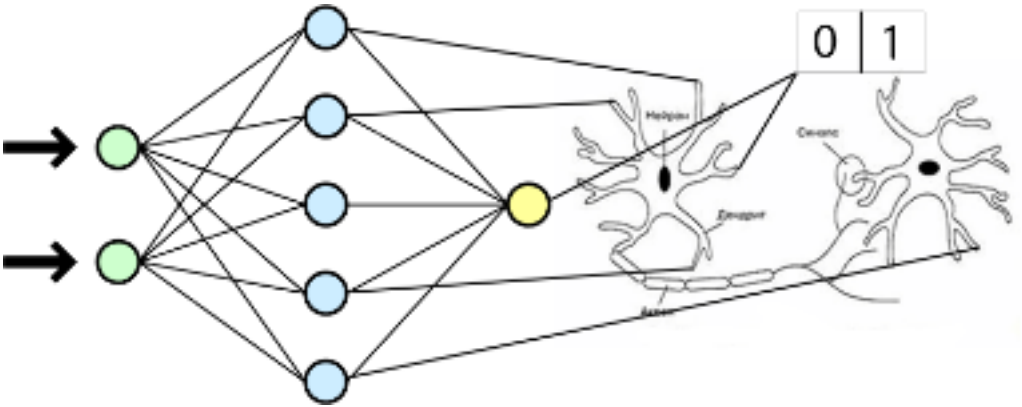


Рис. 12:

Сверточные нейронные сети

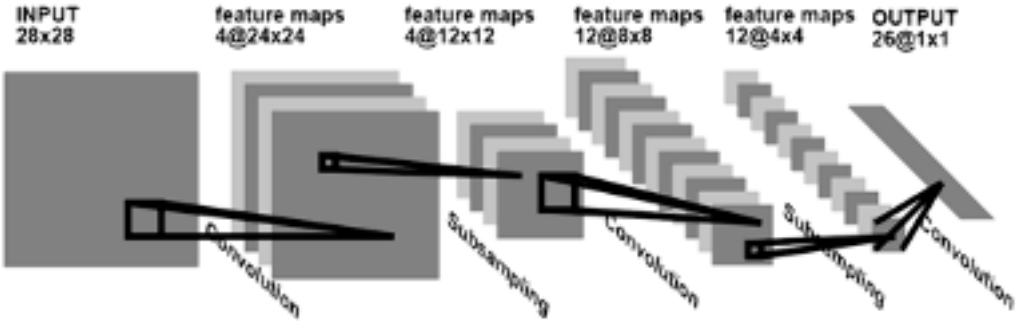
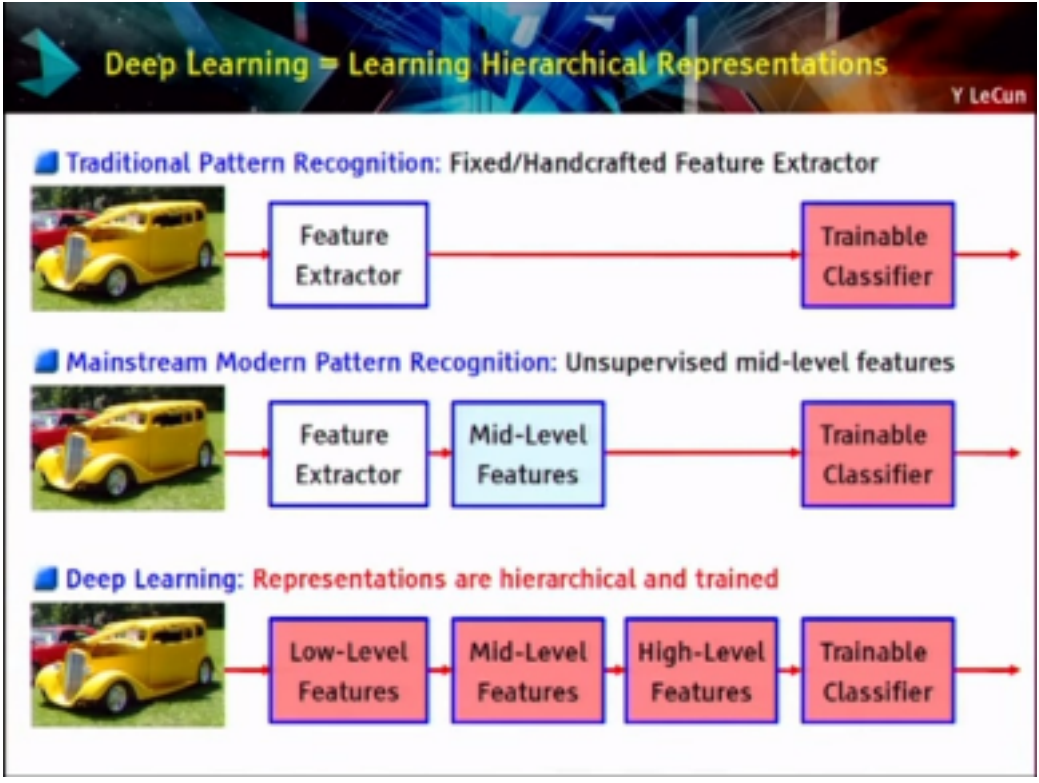


Рис. 13:

Концепция Deep Learning



Проблема качества ПО

Ка́чество програ́ммного обеспечения — способность программного продукта при заданных условиях удовлетворять установленным или предполагаемым потребностям (ISO/IEC 25000:2014).

Филосовский анализ: причины и условия (качество имеет отношения к условиям). Повышение качества ПО, таким образом, требует нового уровня гибкости:

1. Суть предлагаемого нового подхода состоит в переходе на слудующую ступень “про” – к пропрограммам.
2. В качестве вычислительных устройств допускается использование динамических систем произвольной природы, но с некоторыми обязательными свойствами (в частности, не обязательно детерминированным, но непременно контролируемым поведением). Программой и входными данными в этом случае будут их начальные условия.

Определение пропрограммы

Программа (от греч. προ - пред, γράμμα - запись) – термин, в переводе означающий «предписание», то есть предварительное описание предстоящих действий. По аналогии, пропрограмма (греч. προπρογράμμα) – это предварительное описание программы.

Формальное определение пропрограммы, таким образом – это объект, который известно как преобразовывать в программу. Поскольку нас в первую очередь интересуют преобразования пропрограмм, рассмотрим

Эквивалентные преобразования пропрограмм

Для заданного множества целевых языков программирования {Li}, соответствующих правил трансляции в них {TLi}, и правил эквивалентного преобразования (отображения множества пропрограмм в себя) {ETi} пропрограммы суть объекты, которые являются прообразами программ {P} при отображениях {TLi}, если выполняются следующие аксиомы A1 и A2:

Эквивалентные преобразования пропрограмм

A1

Для любых пропрограммы PP и правил трансляции TL1, TL2:

```
PP ->TL1-> P1
                        =>  P1 ~ P2
PP ->TL2-> P2
```

(применение разных правил трансляции к одной и той же пропрограмме приводит к эквивалентным программам).

A2

Для любых пропрограммы PP, правил экв.преобразований ET1, ET2 и правила трансляции TL:

```
PP ->ET1->TL-> P1
                        =>  P1 ~ P2
PP ->ET2->TL-> P2
```

(применение разных эквивалентных преобразований к одной и той же пропрограмме с последующей трансляцией приводит к эквивалентным программам).

Компетентные базы знаний

Чтобы оптимизировать пропрограммы для работы в нужных нам условиях и транслировать их в эффективные программы, нужны знания о том, как это делать.

Компетентной базой знаний для класса задач C называется БЗ, эффективно выдающая рецепты решения любой задачи из класса C.

Для корректного определения нужно задать критерии эффективности (например, зафиксировать лимит времени на “размышление”).

Пропрограмма может рассматриваться как замкнутая относительно операции декомпозиции подзадач надстройка над соответствующей компетентной БЗ.

Проактивные вычисления

Проактивные вычисления – это те же самые пропрограммы, обрабатываемые в динамике. Такая обработка без наличия соответсвующей компетентной базы знаний не представляется возможной – решения надо принимать “на лету”.

Иерархическое представление знаний в КБЗ

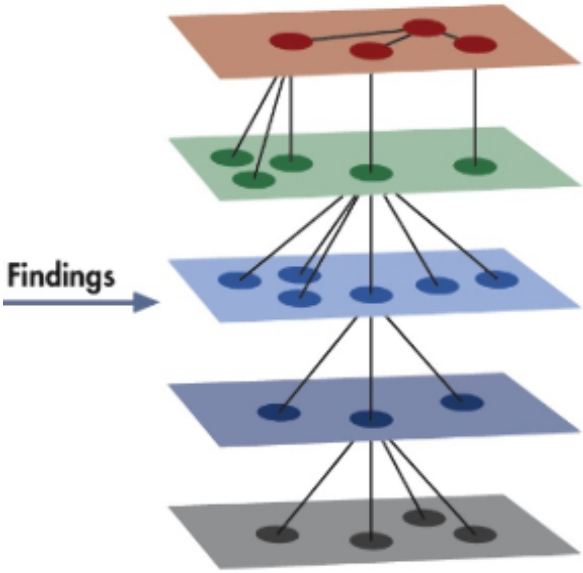


Рис. 15:

Многоуровневая КБЗ может быть построена для класса подзадач предметной области. Не напоминает нейронную сеть, повернутую на 90 градусов ?

Язык URAL (Universal Relational-Algebraic Language)

```
DEF(PreAmp2, PIN(in);PIN(out);PIN(g);PIN(vcc), {
  C_(ci);  ci=in; ci.model="100nF";
  R_(ri);  ri=ci; ri.model="100k";

  T_(t1);  t1.b=ri;
  R_(re1); re1=t1.e; re1=g; re1.model="470";
  C_(cf1); cf1=g;
  R_(rf1); rf1=vcc; rf1=cf1; rf1.model="510k";
  R_(rc);  rc=t1.c; rc=rf1; rc.model="39k";
  C_(cc);  cc=t1.c; cc=g; cc.model="100pF";

  T_(t2);  t2.b=t1.c;
  R_(rc2); rc2=out; rc2.model="2.4k";
  R_(re2); re2=t2.e; re2=g; re2.model="300";

  R_(rs2); rs2=t1.b; rs2.model="50k";
  C_(cs2); cs2=rs2; cs2=g; cs2.model="10nF";
  C_(cs1); cs1=t2.e; cs1.model="10nF";
  R_(rs1); rs1=cs1; rs1=rs2; rs1.model="50k";

  R_(rb); rb=t1.b; rb=t2.e; rb.model="360k";

  [...]
```

Сгенерированный SPICE-код

```
test circuit
vcc v 0 15v

vin 0 i1 sin(0 450mv 100hz 0 0) dc=0 ac=1
lp i1 i2 3.4H
rp i2 in 6.7k
cp 0  in 150pF

ci      ci.inner        in      100nF
ri      ri.inner        ci.inner        100k
q_t1    t1.c    ri.inner        t1.e    sbeta
q_t2    t2.c    t1.c    t2.e    sbeta
q_t4    0       t2.c    t4.e    f_pnp
cs2     0       rs2.inner       10nF
re2     0       t2.e    300
cc      0       t1.c    100pF
cf1     cf1.inner       0       1000u
rf1     cf1.inner       v       510k
rc      cf1.inner       t1.c    39k
q_t3    v       rc2.inner       t3.e    f_npn

[...]
```

Спектральный состав выходного сигнала

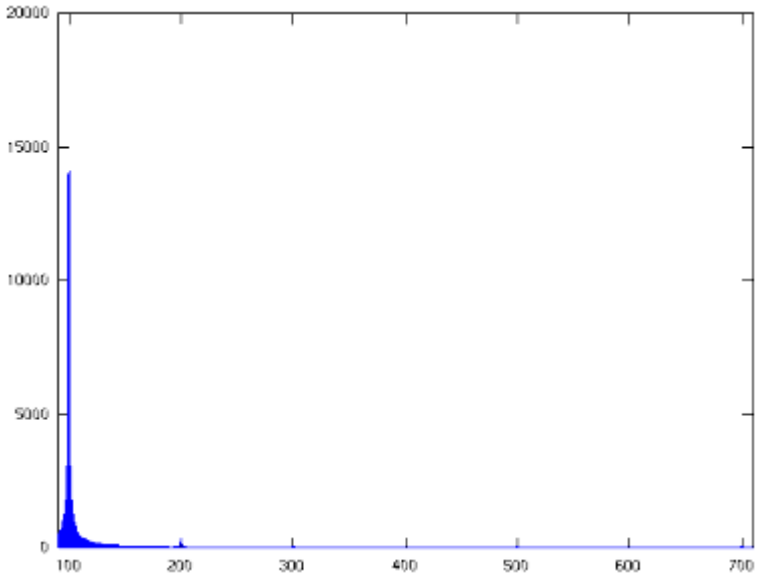


Рис. 16:

SEA-Интерпретатор

Дополним классический Eval/Apply-интерпретатор Mc Carthy's стадией Split:

- S - Split: декомпозиция объекта
- E - Eval: эволюция (рекурсивная) суб-объектов
- A - Apply: композиция (сборка) итогового результата

По аналогии с Eval/Apply-схемой, SEA-схема работает рекурсивно, но имеет важную дополнительную интеллектуальную стадию Split. Мы допускаем сложные вычисления на этой стадии, но с жестким ограничением на общее количество используемых вычислительных ресурсов.

SEA-Интерпретатор, дополненный простейшей базой знаний в виде SM-таблицы (Super Memo Table), во многих случаях хорошо подходит для непосредственного получения программы из пропрограммы, представленной в виде агрегации некоторых объектов.

В процессе обработки объекта могут получаться суб-объекты, свойства которых в базе знаний отсутствуют. В этом случае автоматически инициируется их исследование путем генерации и запуска соответствующих тестирующих программ, результаты которых и пополняют динамическую базу знаний.

SEA-Интерпретатор и Факторизация

Если смотреть на пропрограмму, как на описание задачи, для решения которой мы хотим синтезировать квазиоптимальную программу, то нетрудно увидеть связь между стадией Split и факторизацией иерархии множества задач, которые мы де-факто имеем для заданной предметной области.

Фактически на каждом шаге SEA-интерпретатор пробует ту или иную декомпозицию, спускаясь сначала рекурсивно вниз и собирая по пути недостающую информацию, а затем поднимается вверх, и оценивая эффективность полученных решений.

Выбор хорошей факторизации гарантирует финитность и эффективность данного процесса.

Отслеживая содиржимое SM-таблицы, и инициируя решения недостающих задач, можно добиться того, что для любой наперед заданной задачи из данной предметной области в нашей динамической базе знаний найдется ее округленное приближение. В этом случае SEA-Интерпретатор построил компетентную базу знаний, и решение любой задачи уже не протребует проводить экспресс-исследования свойств реальных объектов.

Мультидинамические системы

Динамическая система DS – множество элементов, для которого задана функциональная зависимость между временем и положением в фазовом пространстве каждого элемента системы, то есть закон эволюции DS(t).

Переход динамической системы в последующее состояние (для дискретного времени) можно связать с некоторой функцией эволюции – динамикой d:

```
DS_t' = d (DS_t)
```

Мультидинамическая система MDS – система с множеством динамик

```
d_1,..., d_n; n > 1
```

Переход в последующее состояние производится под действием одной из динамик:

```
DS_t' = d_i (DS_t)
```

На мультидинамическую систему можно смотреть как на динамическую, снабженную единственной, но управляемой извне динамикой d.

Автокоммутативные мультидинамическе системы (Абелевы соответствия)

Назовем мультидинамическую систему автокоммутативной, если все ее эволюционные динамики коммутируют между собой:

$\forall i \forall j (d_i \bullet d_j \equiv d_j \bullet d_i)$

Автокоммутативные MDS обладают свойством эквифинальности: порядок применения динамик d_i можно произвольно менять, что не скажется на конечном состоянии системы.

Обратное, вообще говоря, неверно. Эквифинальные MDS совершенно не обязаны быть автокоммутативными.

Надстройки над параллельными вычислительными системами

Специальная надстройка (добавление запаса хода) превращает чисто функциональную параллельную программу в автокоммутативную мультидинамическую систему.

Из того факта, что автокоммутативность есть лишь простой частный случай эквифинальности, легко прийти к выводу, что распараллеливание чисто фунцкиональных программ не является наиболее общим случаем бесконфликтного распараллеливания.

Лемма о катящемся шарике

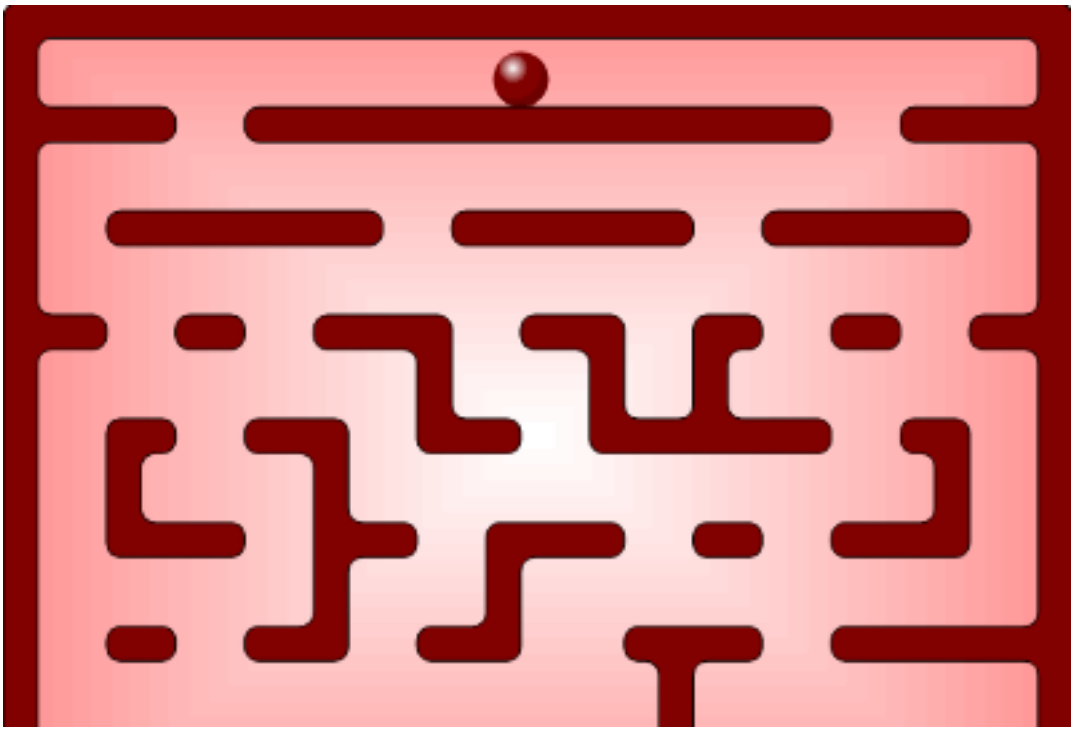


Рис. 17:

Профункциональное программирование

Имеется класс императивных программ, которые весьма интересны для HPC.

1. Известно, что фунциональные программы хорошо распараллеливаются.
2. Известно, что любой программе P можно сопоставить функциональную программу F(P), такую, что $F(P) \sim P$.

Введем класс профункциональных программ PFP, таких, что:

$P \in PFP \Leftrightarrow F(P) \approx P$

При этом известный способ распараллеливания F(P) может существенно приблизить нас к созданию параллельной версии P.

Выводы

Предложенный подход позволяет

- Существенно повысить гибкость программного обеспечения и прозрачность его разработки;
- Распределить работу по созданию высокопроизводительных приложений между специалистами по предметной области, аппаратному обеспечению и математиками.

Предложенные интеллектуальные алгоритмы позволяют

- Объединить целенаправленный поиск квазиоптимального решения с извлечением необходимых знаний о предметной области;
- “Встраивать” достижения интеллекта в обычные программы в виде компетентных баз знаний.

Вопросы безопасности ИИ

- Неверифицируемый искуственный интеллект может использоваться непосредственно только для поиска хорошей факторизации полугруппы операций в предметной области.

Спасибо за Внимание!

Вопросы ?