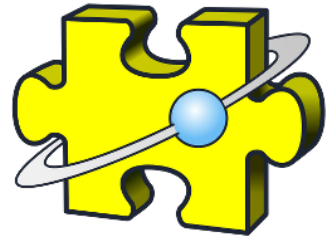


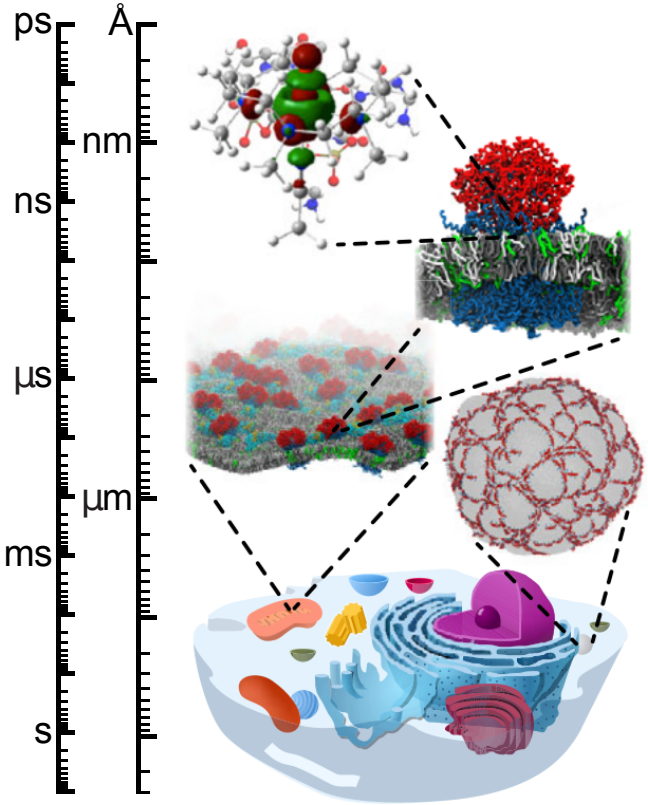
BlackBox Component Builder для надмолекулярного моделирования

Денисов Иван Андреевич

Сибирский федеральный университет (Красноярск)

День Оберона в Москве, 30 сентября 2017

Живая клетка



Квантовый
- атомы, электроны и электронные облака включены
- явный растворитель
- квантовая механика

Все-атомный
- все или большенство атомов влючено
- явный растворитель
- молекулярная динамика

Грубо-зернистый
- стержни, состоящие из нескольких атомов
- явный или неявный растворитель
- молекулярная динамика

Супер-грубо-зернисрый
- сайты взаимодействия из многих атомов, частей белков / белков
- неявный растворитель
- стохастическая механика

Непрерывный
- материалы как непрерывные массы
- неявный растворитель
- непрерывная механика

Рис. 1: Уровни моделирования живой клетки сегодня[1]

[1] H. I. Ingolfsson et al. “Computational ’microscopy of cellular membranes”. en. In: *Journal of Cell Science* 129.2 (2016), pp. 257–268.

Грубозернистые модели

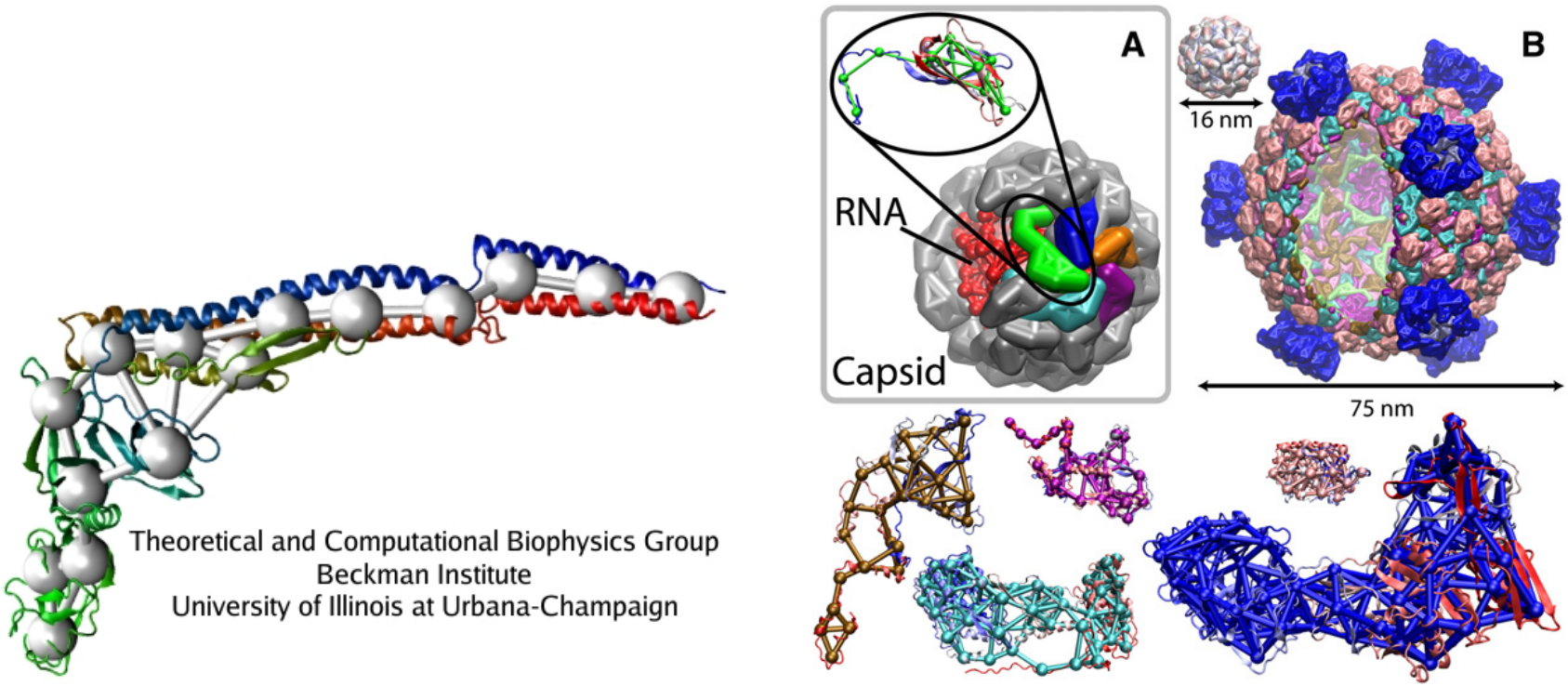


Рис. 2: Примеры грубозернистых моделей надмолекулярных систем [2]

[2] Anton Arkhipov, Peter L. Freddolino, and Klaus Schulten. "Stability and Dynamics of Virus Capsids Described by Coarse-Grained Modeling". In: *Structure* 14.12 (2006), pp. 1767–1777.

Переход между моделями разных уровней организации

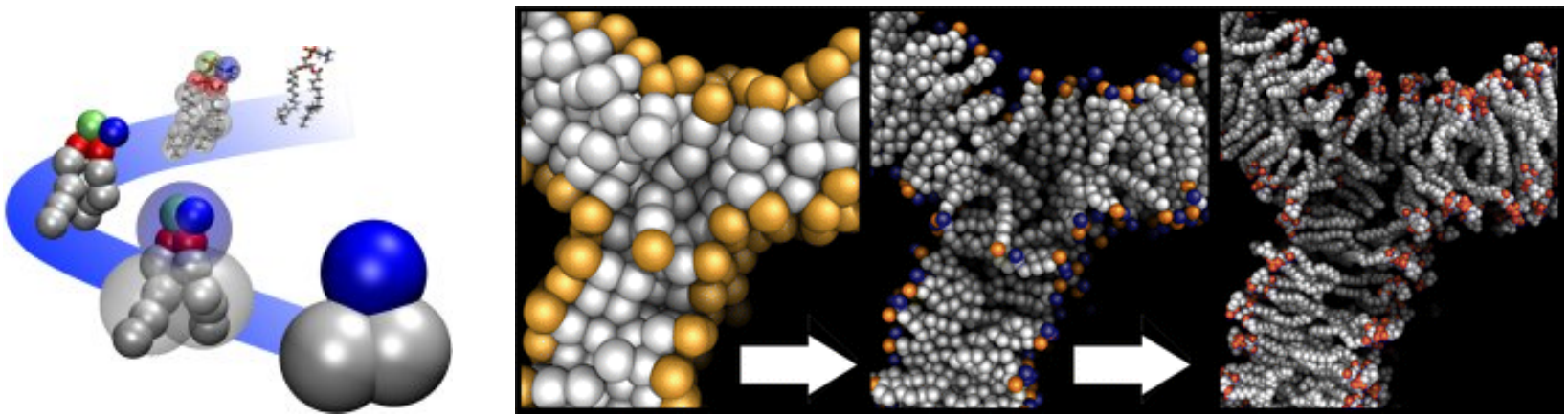


Рис. 3: Понижение и увеличение числа объектов в молекулярных моделях [3]

[3] Tsjerk A. Wassenaar et al. “Going Backward: A Flexible Geometric Approach to Reverse Transformation from Coarse Grained to Atomistic Models”. In: *Journal of Chemical Theory and Computation* 10.2 (2014), pp. 676–690.

Постановка задачи

Цель — поиск адекватного представления многоуровневых систем, создание эффективного аппарата математической биофизики клетки.

Требования:

- возможность решения уравнений движения на существующих вычислительных ресурсах для получения данных об эволюции и поведении надмолекулярных биологических систем;
- автоматическая подстройка сложности модели под контекст расчета;
- описание образование и разрыва связей на разных уровнях;
- возможность параметризации появления новых свойств в системе (эмерджентность).

Гипотеза: более точной абстракцией удастся меньшим числом переменных описать систему

Метод дополнительных частиц

Образования комплекса описываем путем добавления в систему объекта более высокого уровня организации. Появление объекта на уровне $L+1$ приводит к возникновению нового свойства у ансамбля объектов на уровне L (например связи).

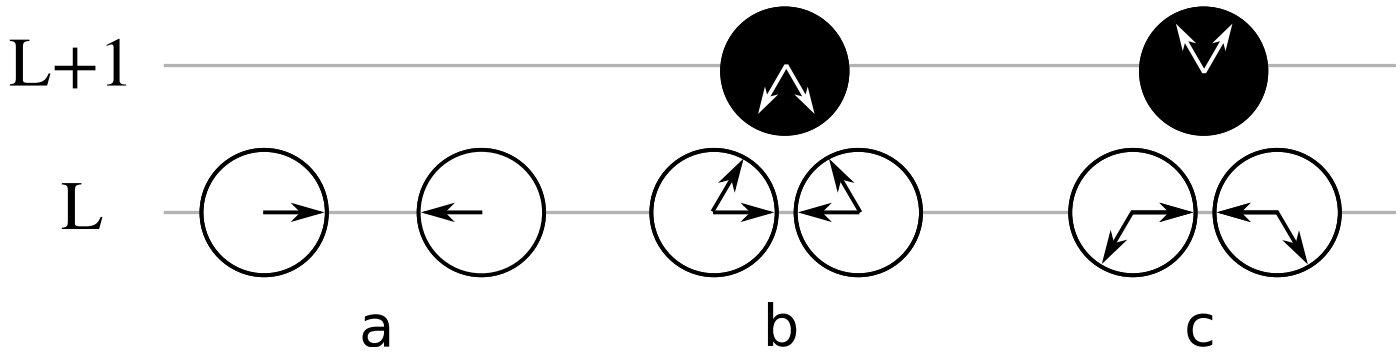


Рис. 4: (a) два объекта на уровне организации L сближаются, (b,c) при достижении заданного расстояния происходят изменения в системе, добавляется объект на уровне $L+1$; такой объект может как стабилизировать (b) так и дестабилизировать (c) связь объектов на уровне L

В рамках одноуровневой физической модели такой переход невозможен.

Уравнения движения

Учет диссипации энергии (сил трения, вязкость среды) и температуры среды в уравнении Ланжевена:

$$m\ddot{q} = F(q) - \gamma\dot{q} + \eta(t), \tag{1}$$

где $F(q)$ — силы, обусловленные потенциалами в системе;
$\gamma\dot{q}$ — сила вязкого трения, которая пропорциональна скорости частицы и отвечает за диссипацию энергии в системе;
$\eta(t)$ — шумовой член, выражающий температурную накачку системы.

Существование в модели накачки и диссипации энергии важно для моделирования самоорганизации, так как только при их наличии система перебирает возможные стационарные состояния и может найти минимум энергии.

Схема трехуровневой модели

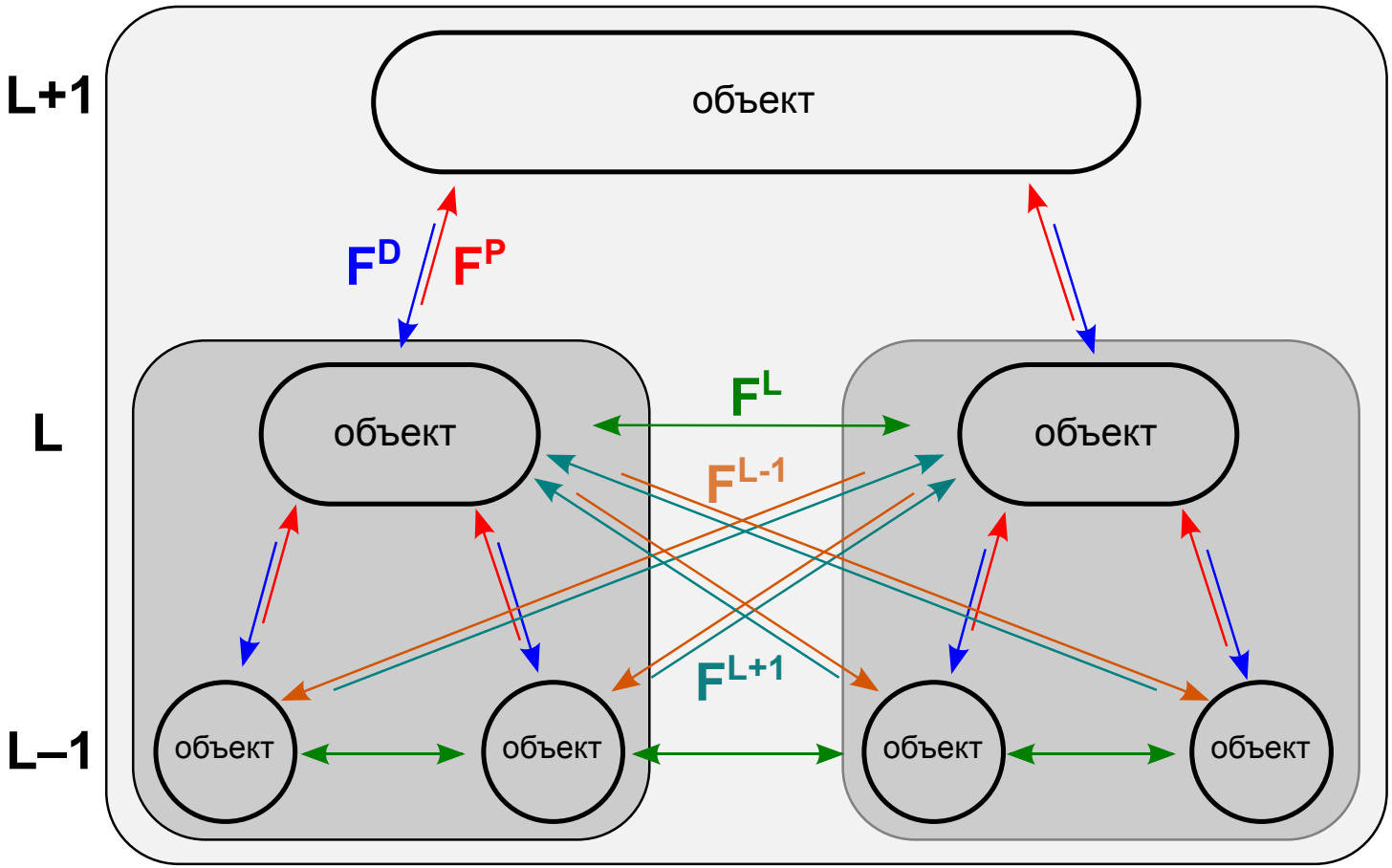


Рис. 5: Пять типов сил: F^L — между частицами одного уровня, F^{L+1} — с вышележащим уровнем, F^{L-1} — с нижележащим уровнем, F^P — с родительскими частицами, F^D — с дочерними частицами

Уравнение реконструкции

$$m_j^i \ddot{q}_j^i = \sum_{k \neq i} F_j^L(q_j^i, q_j^k) + \sum_{k \notin D_j^i} F_j^{L+1}(q_j^i, q_{j+1}^k)\mathcal{T}_{j+1}^k + \sum_{k \notin P_j^i} F_j^{L-1}(q_j^i, q_{j-1}^k)\mathcal{T}_{j-1}^k +$$
$$+ \sum_{k \in P_j^i} F_j^P(q_j^i, q_{j-1}^k)\mathcal{T}_{j-1}^k + \sum_{k \in D_j^i} F_j^D(q_j^i, q_{j+1}^k)\mathcal{T}_{j+1}^k - \gamma_j^i \dot{q}_j^i + \eta_j^i(t), \quad (2)$$

где q_j^i, m_j^i — координата и масса i-ого объекта на уровне j;
F_j^L — сила, действующая со стороны объектов уровня j;
F_j^{L+1} — сила, действующая со стороны объектов уровней $j+1$;
F_j^{L-1} — сила, действующая со стороны объектов уровней $j-1$;
F_j^P — сила, действующая со стороны родительских объектов уровня $j-1$;
F_j^D — сила, действующая со стороны дочерних объектов уровня $j+1$;
P_j^i, D_j^i — множество родительских/дочерних объектов для объекта i уровня j;
γ_j^i — коэффициент вязкости i-ого объекта на уровне j;
$\eta_j^i(t)$ — энергетическая накачка i-ого объекта на уровне j.

BlackBox для Linux/OpenBSD/FreeBSD/Windows

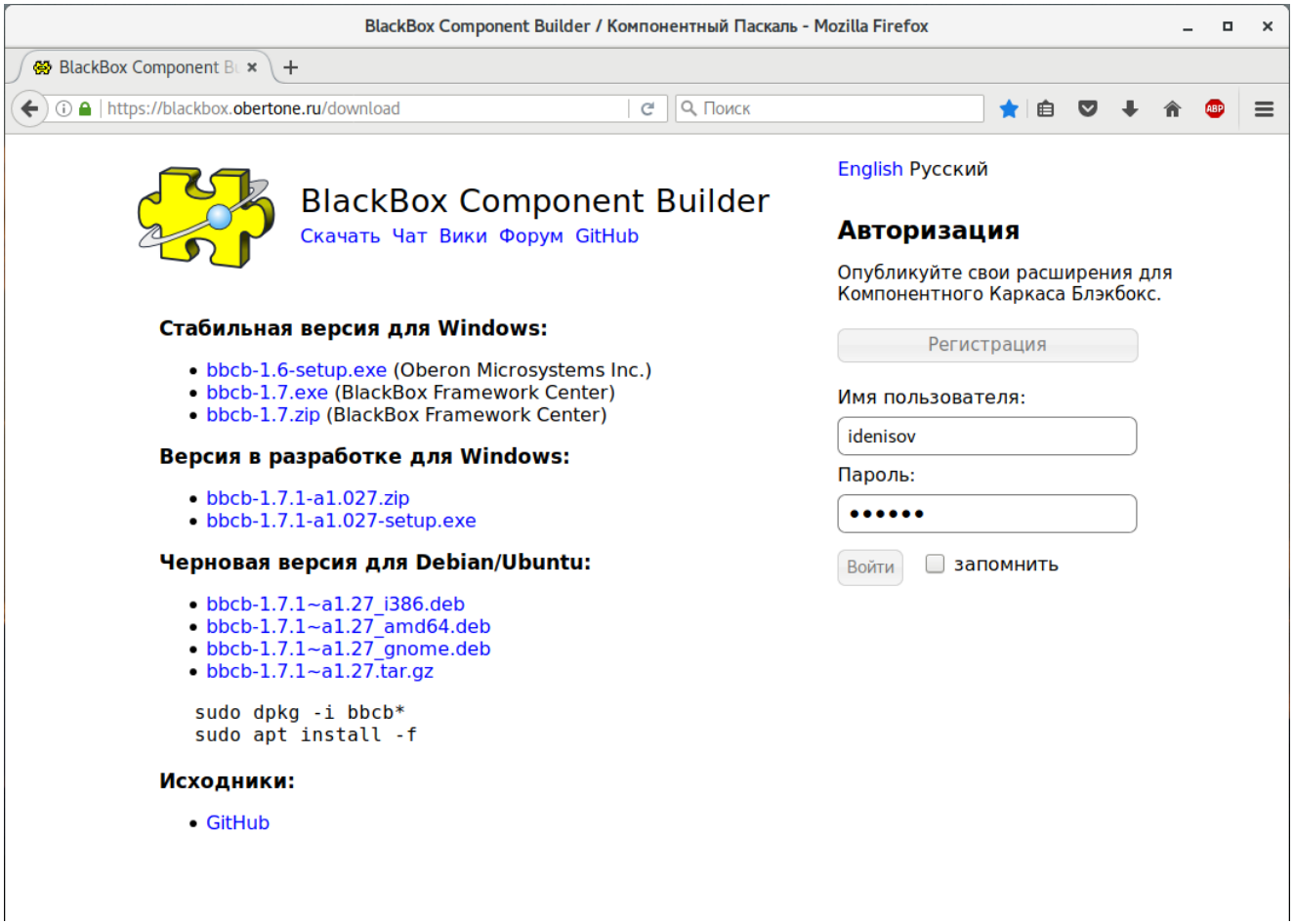


Рис. 6: Страница загрузки

BlackBox для Linux/OpenBSD/FreeBSD/Windows

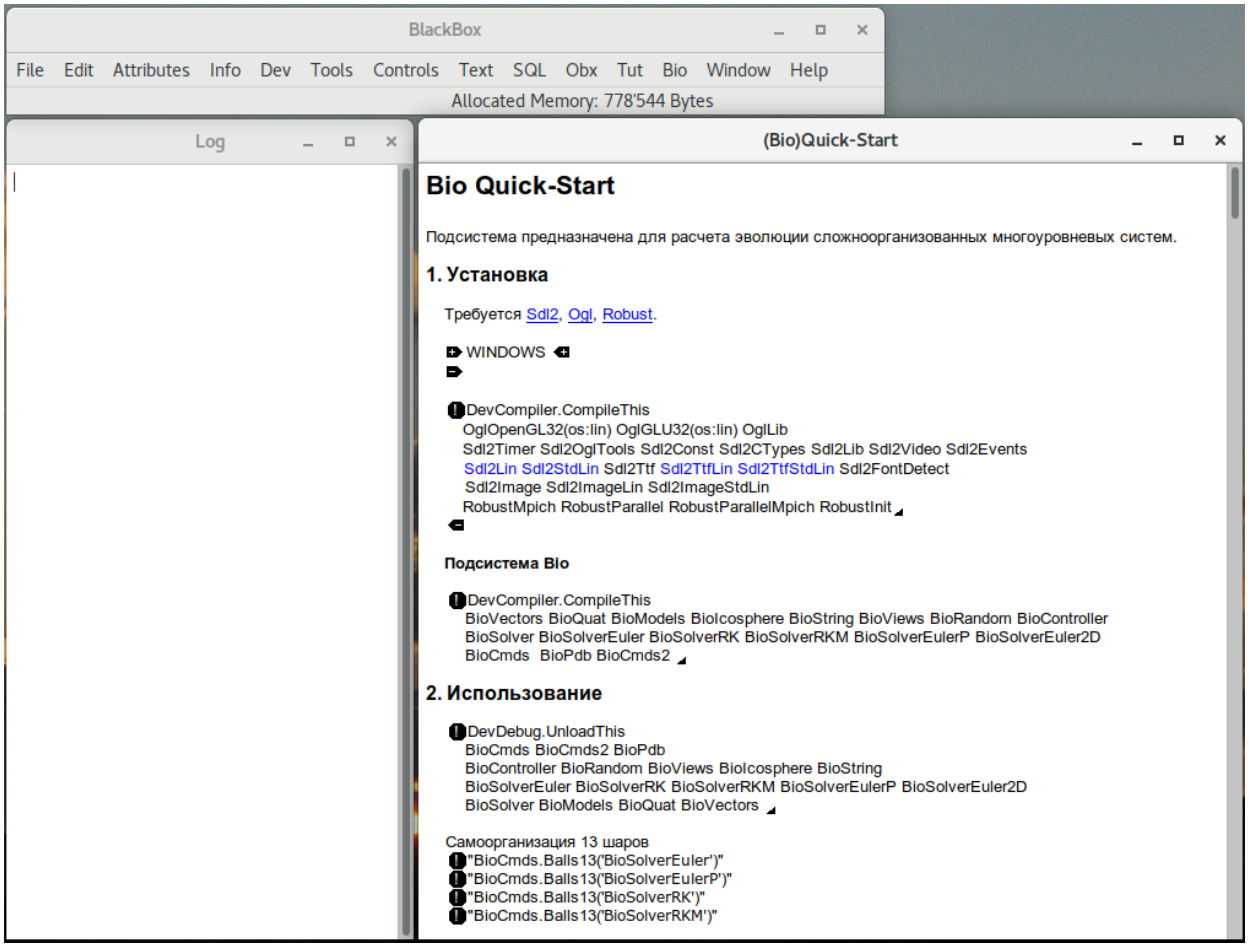


Рис. 7: Интерфейс Блэкбокс в Linux

Подсистема Dev2: компоновщик ELF & PE (спасибо И.Дехтяренко и А.Ширяеву)

```
(!) Dev2Linker1.LinkElfExe Linux blackbox := Kernel$+
    HostFiles HostGnome StdLoader ~

Dev2
  Dev2Linker
  Dev2Linker1
  Dev2LnkBase
  Dev2LnkChmod
  Dev2LnkLoad
  Dev2LnkWriteElf
  Dev2LnkWriteElfStatic
  Dev2LnkWriteElf_Got
  Dev2LnkWritePe
```

Поддержка [ccall16] (спасибо LuoWy)

```
  PROCEDURE PreCcall16* (val: INTEGER);  (* DevCPC486 *)
    VAR c, sp,cx: DevCPL486.Item;
  BEGIN
    CheckAv(CX);
    DevCPL486.MakeReg(cx, CX, Int32);
    DevCPL486.MakeReg(sp, SP, Int32);
    DevCPL486.GenMove(sp, cx);  (* MOVE ECX, ESP *)

    DevCPL486.MakeConst(c, -16, Int32);
    DevCPL486.GenAnd(c, sp);    (* AND ESP,-16*)
    DevCPL486.GenPush(cx);      (* push ecx *)
    val:= (-4 - val) MOD 16;
    IF val # 0 THEN
      DevCPL486.MakeConst(c, val, Int32);
      DevCPL486.GenSub(c, sp, FALSE) (* sub ESP, val *)
    END
  END PreCcall16;

  PROCEDURE PostCcall16* (val: INTEGER);
    VAR c, sp: DevCPL486.Item;
  BEGIN
    val := (-4 - val) MOD 16 + val;
    DevCPL486.MakeConst(c, val, Int32);
    DevCPL486.MakeReg(sp, SP, Int32);
    DevCPL486.GenAdd(c, sp, FALSE); (* ADD ESP, val *)
    DevCPL486.GenPop(sp)            (* POP ESP*)
  END PostCcall16;

...
  IF (x.obj.sysflag = ccall) OR (x.obj.sysflag = ccall16) THEN
    n:=CCallParSize(x.obj, NIL);
    IF (x.obj.sysflag = ccall) THEN
      AdjustStack(n)
    ELSE (* ccall16*)
      PostCcall16(n)
    END;
  END;
```

SDL2 для кроссплатформенных приложений

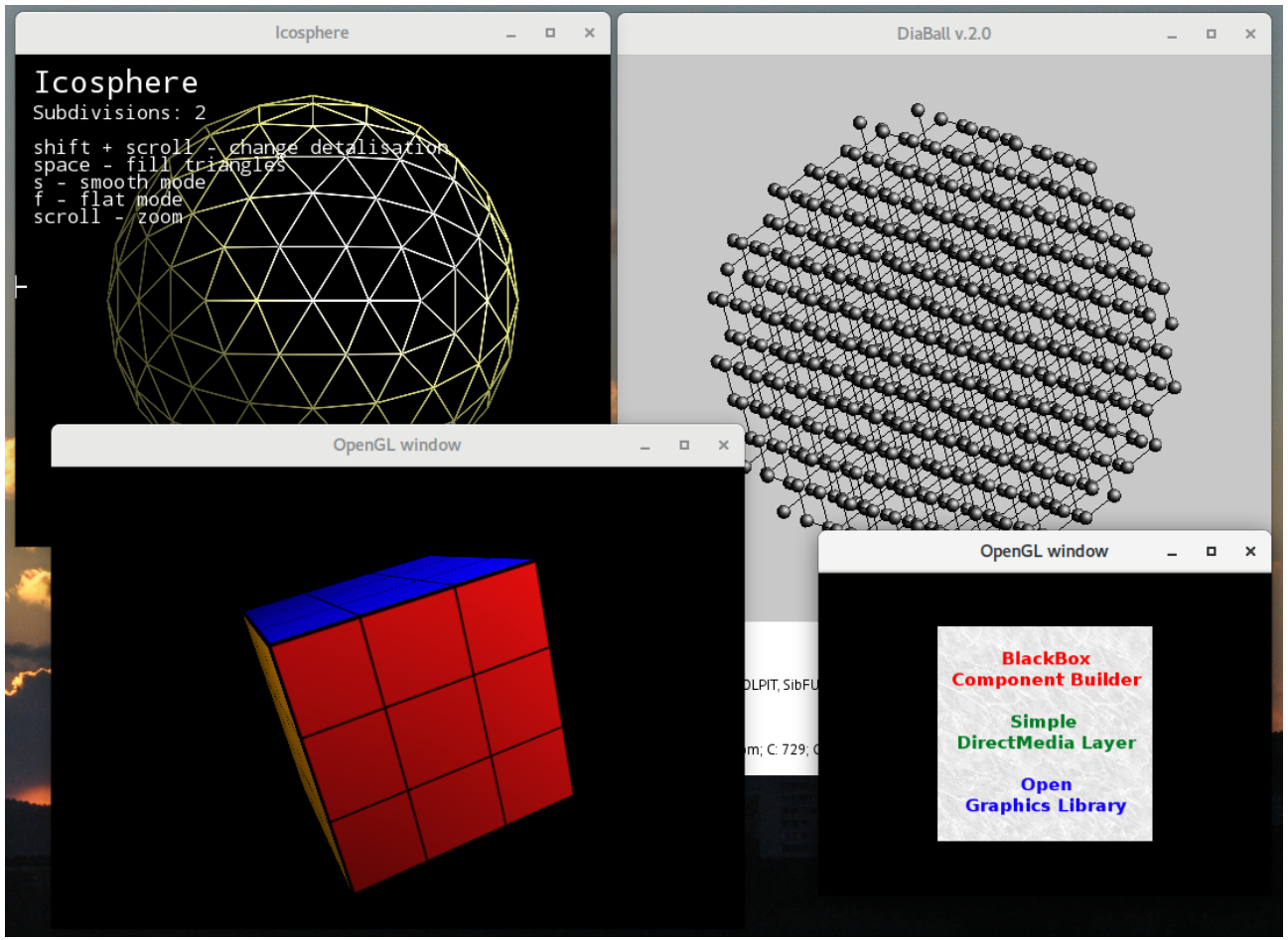


Рис. 8: Поддержка кроссплатформенных окон, загрузка изображений, шрифтов, обработка ввода пользователя

Результат

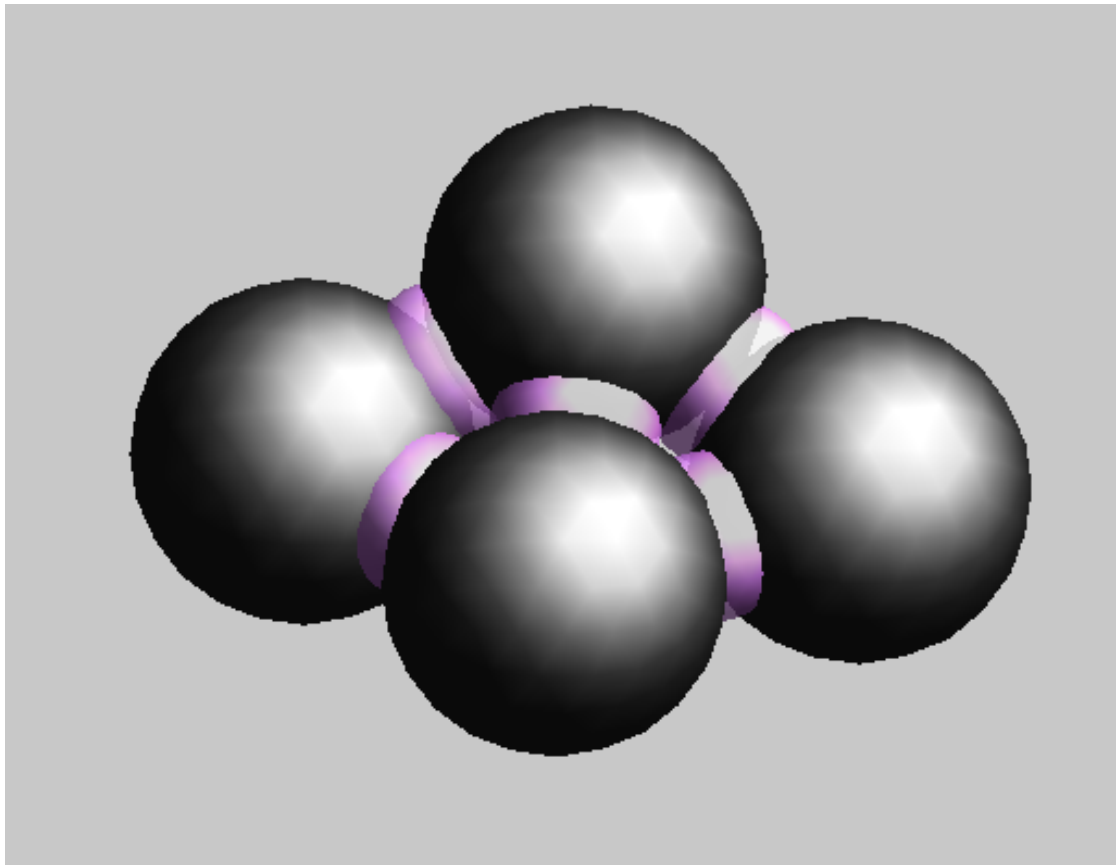


Рис. 9: Система их 4-х частиц связанных дополнительными частицами

Как выполнить программу на многих процессорах?

Как сделать, чтобы программа работала на обоих процессорах?

ЦИТАТА

Иван Денисов » Четверг, 08 Октябрь, 2009 19:23

Здравствуйте уважаемые участники форума. Несколько лет стараюсь следить за новостями и разбираться по мере сил со средой BlackBox. Написал программу, которая требует много ресурсов процессора. У меня двуядерный и хотелось бы загружать оба камня вычислениями, а не только один. С потоками и ядром Active BlackBox, пока не разбирался.
Подскажите с чего начать?
Как заставить программу работать на двух процессорах?
Есть ли доступный пример?
Как заложить масштабируемость в программу, на случай, если появится больше процессоров?

Иван Денисов

Сообщения: 1883
Зарегистрирован: Четверг, 08 Октябрь, 2009 19:00
Откуда: Красноярск

ПС

Рис. 10: Моя первая тема на OberonCore

MPI — коммуникационный протокол

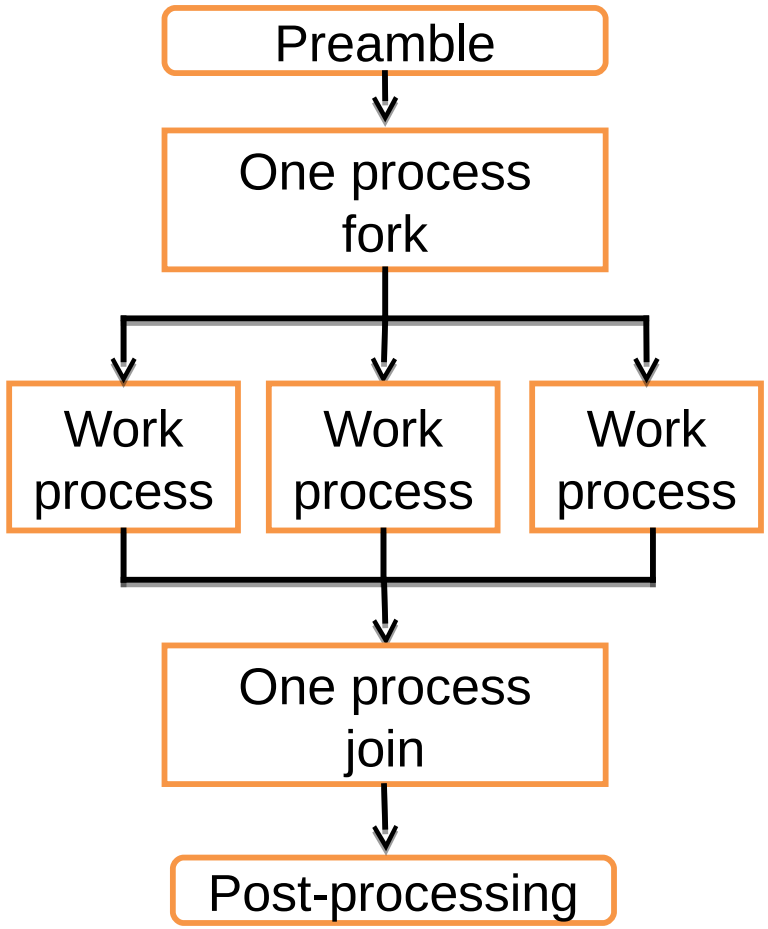
Усилия по разработке Message Passing Interface начались летом 1991 года. Версия 1.0 MPI была выпущена в июне 1994 года. Разработка MPI не была санкционирована какой-либо крупной организацией по стандартизации, тем не менее, интерфейс стал стандартом де-факто для связи процессов при работе параллельной программы на системе с распределенной памятью.

Программа, использующая MPI, легче отлаживается (сужается простор для совершения стереотипных ошибок параллельного программирования) и быстрее переносится на другие платформы (в идеале, простой перекомпиляцией).

Концепция MPI

Процесс — компьютерная программа в стадии своего выполнения.
Компьютерная программа является пассивным набором инструкций; процесс является активным выполнением этих инструкций.

MPICH — это эффективная реализация стандарта MPI.
http://www.mpich.org/



Формула параллельного вычисления:
инициализация [[отправка/прием]n, ожидание и расчет]n финализация

Интерфейс для параллельных вычислений

```
DEFINITION RobustParallel;
  IMPORT Stores;
  CONST tag = 1;
  TYPE
    Computon = POINTER TO ABSTRACT RECORD (Stores.Store)
      (c: Computon) Do, NEW, ABSTRACT
    END;
  VAR
    rank-: INTEGER;
    size-: INTEGER;
  PROCEDURE LogMsg (a: ARRAY OF CHAR);
  PROCEDURE Pull (rank, tag: INTEGER): Computon;
  PROCEDURE Push (store: Computon; rank, tag: INTEGER);
  PROCEDURE SetHook (h: Hook);
  PROCEDURE Stop;
  PROCEDURE Try (rank, tag: INTEGER): Computon;
  PROCEDURE Unload (module: ARRAY OF CHAR);
  PROCEDURE Wait;
END RobustParallel.
```

Пример использования компутона

```
TYPE
  Summator = POINTER TO RECORD (P.Computon)
    data: POINTER TO ARRAY OF REAL;
    result: REAL
  END;

PROCEDURE (a: Summator) Do;
VAR i: INTEGER;
BEGIN
  ASSERT(a.data # NIL, 21);
  FOR i := 0 TO LEN(a.data) - 1 DO
    a.result := a.result + a.data[i]
  END;
  a.data := NIL;
  P.Push(a, 0, P.tag)
END Do;
```

Пример использования компутона

```
PROCEDURE (a: Summator) Externalize (VAR wr: Stores.Writer);
VAR i: INTEGER;
BEGIN
  IF P.rank # 0 THEN wr.WriteReal(a.result)
  ELSE
    IF a.data # NIL THEN
      wr.WriteInt(LEN(a.data));
      FOR i := 0 TO LEN(a.data) - 1 DO wr.WriteReal(a.data[i]) END
    ELSE wr.WriteInt(0) END
  END
END Externalize;

PROCEDURE (a: Summator) Internalize (VAR rd: Stores.Reader);
VAR i, len: INTEGER;
BEGIN
  IF P.rank = 0 THEN rd.ReadReal(a.result)
  ELSE
    rd.ReadInt(len);
    IF len > 0 THEN
      NEW(a.data, len);
      FOR i := 0 TO len - 1 DO rd.ReadReal(a.data[i]) END
    ELSE a.data := NIL END
  END
END Internalize;
```

Пример использования компутона

```
PROCEDURE Do*;
VAR a: Summator; i, j: INTEGER;
BEGIN
  NEW(a);
  NEW(a.data, 10);

  FOR i := 1 TO P.size - 1 DO
    FOR j := 0 TO LEN(a.data) - 1 DO
      a.data[j] := i * j
    END;
    P.Push(a, i, P.tag);
  END;

  FOR j := 1 TO P.size - 1 DO
    Log.String(" p:"); Log.Int(j);
    Log.Real(P.Pull(j, tag)(Summator).result)
  END;
  Log.Ln
END Do;
```

Демонстрация, вопросы и обсуждение

Подвижные клеточные автоматы

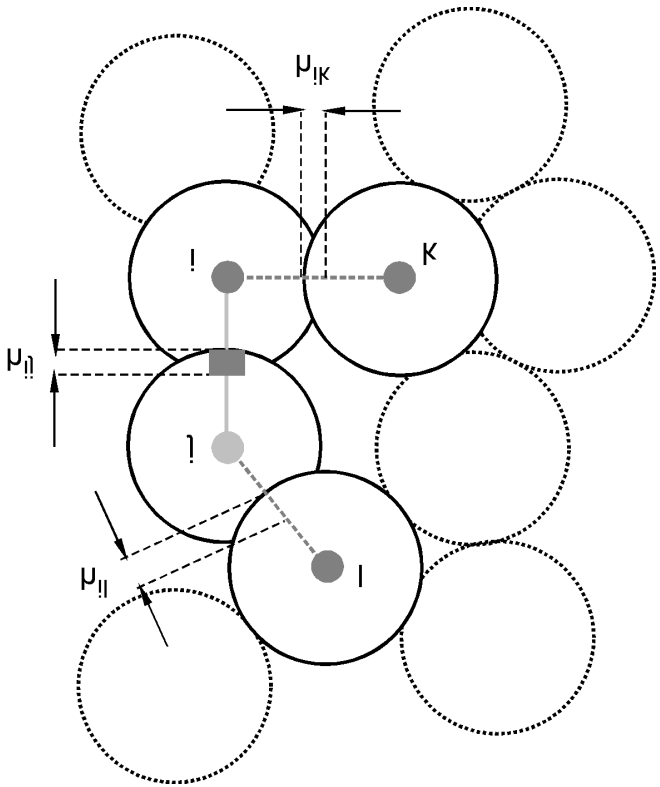


Рис. 11: Условия переключения автоматов[4]

Рис. 12: Применение метода Псахье для моделирования разрушения бетона

[4] С Г Псахье и др. “Метод подвижных клеточных автоматов как новое направление дискретной вычислительной механики. I. Теоретическое описание”. В: *Физическая мезомеханика* (2000), с. 5—13.

Функция триггера

Некоторые частицы уже заняты в ансамблях, поэтому простых прямоугольных функций недостаточно. Суперпозицию прямоугольных функций назовем *триггером* (*англ. trigger*) и обозначим как $\mathcal{T}$:

$$\mathcal{T}_j^i = \theta \left(\sum_{m,n \neq m, D_{j-1}^n \cup D_{j-1}^m = \varnothing} \prod \left(\frac{q_{j-1}^n - q_{j-1}^m}{2(r_{j-1}^n + r_{j-1}^m)} \right) \right), \qquad (3)$$

где i, m, n — номера частиц на уровне, а выражение $D_{j-1}^n \cup D_{j-1}^m = \varnothing$ означает, что у частиц n и m на уровне $j-1$ еще нет общих дочерних объектов.

Функция сукцессора

Функция наследования или сукцессор ($\mathcal{S}$, *от лат. succesio — преемственность, наследование*) — отображение свойств родительских объектов на свойства дочерних объектов. $\mathcal{S}$ определяет свойства нового объекта, эффективную массу, положение, потенциалы и алгоритмы триггера и сукцессора на следующем уровне. В математической форме сукцессор выражается следующим образом:

$$\mathcal{O}^i_{j+1} = \mathcal{S}_j \left(\mathcal{O}^n_j, \mathcal{O}^m_j \right), \qquad \{n, m\} \in P^i_{j+1} \tag{4}$$

где $\mathcal{O}$ — совокупность свойств объекта.

Допускает математическое описание эмерджентности как химического и биологического явления.