Параллельное и распределенное программирование. Современное состояние и вызовы.

Аветисян Арутюн Ишханович
arut@ispras.ru

Вызовы:

- Сложность создаваемых систем (авиационная и космическая промышленность, атомная энергетика, кораблестроение, автомобильная промышленность и др.)
- Экспоненциальный рост объема данных (биология, генетика, физика, Интернет и др.)

Массовое внедрение технологий параллельного и распределенного программирования – ответ на эти вызовы

Традиционный научно-технический подход:

1) Разработка теории или проектирование на бумаге
2) Поставить эксперименты или построить систему

Ограничения

- Слишком трудно – строить большие аэродинамические трубы
- Слишком дорого – строить пассажирский лайнер на выброс
- Слишком медленно – ждать пока произойдет эволюция климата или галактики
- Слишком опасно – вооружения, разработка медикаментов, эксперименты над климатом

Современный подход:

3) Использование параллельных и распределенных компьютерных систем для моделирования изучаемого явления,
опираясь на известные законы физики, эффективные численные методы и адекватные компьютерные модели

Пример: Глобальное предсказание погоды

Территория России 17,075,200 км2. Пусть атмосфера над территорией России разделена на ячейки размером в 1 км × 1 км × 1 км до высоты 50 км (50 ячеек в высоту) – всего около 108 ячеек.

Пусть каждый узел характеризуется 10 параметрами, каждый из которых вычисляется в среднем за 30 действий. Параметры изменяются во времени, так что их значения нужно вычислять с некоторым шагом по времени. Согласно сделанным предположениям вычисления в каждой ячейке для одного шага по времени требуют 300 операций с плавающей точкой. Таким образом, для каждого шага по времени необходимо выполнить ~1011 операций с плавающей точкой.

Для прогнозирования погоды на 7 дней с помощью вычислений с шагом в 1 мин на компьютере производительностью 1 Gflops (109 операций с плавающей точкой/сек) потребуется 106 сек или более 10 дней.

Чтобы выполнить эти вычисления за 5 мин требуется компьютер с производительностью 3.4 Tflops (3.4 × 1012 операций с плавающей точкой/сек). Требуемая память для одного экземпляра сетки более 64 Гб.

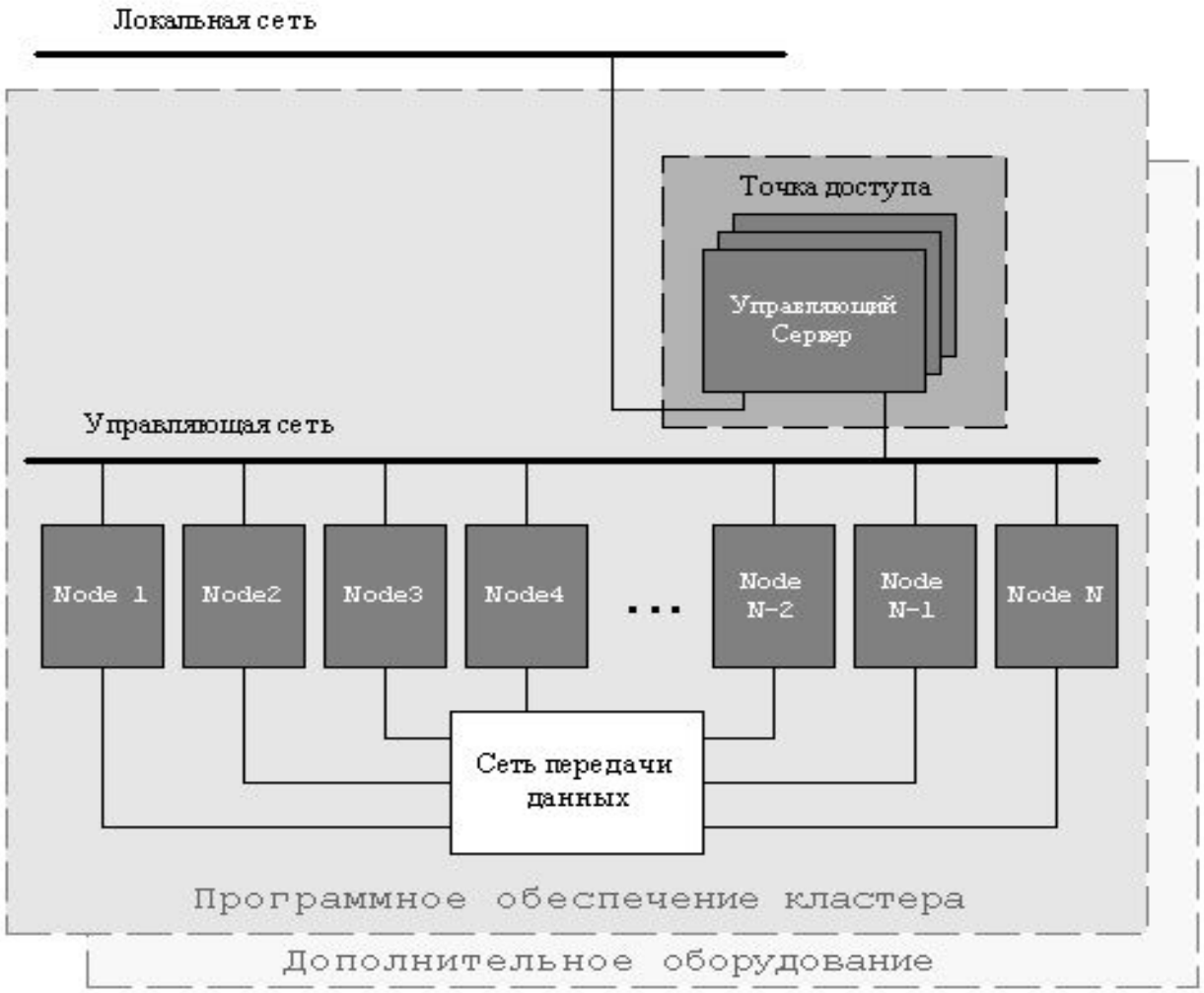
Современные высокопроизводительные вычислительные системы

- За последние 20 лет рост производительности компьютеров, входящих в список «Top500», был большим, чем предсказанный согласно «закону Мура» (удвоение производительности каждые полтора – два года):

Top500 1993 г:	Top500 июнь 2010 г:
№ 1 = 59.7 GFlop/s	№ 1 = 2331.00 TFlop/s
№ 500 = 422 MFlop/s	№ 500 = 54.79 TFlop/s

- Все системы с **распределенной** памятью
- Не более 10 из 500 (июнь 2010) являются **суперкомпьютерами**
- Остальные построены по **кластерной** технологии

Высокопроизводительный кластер (I)



Высокопроизводительный кластер (II)

Кластер* строится из стандартных (commodity) аппаратных компонент, на которых установлено стандартное системное программное обеспечение

Аппаратура узла: Xeon, Opteron, PowerPC, …, гибридные системы (например, на базе Tesla)

Коммуникационное оборудование: GigE, SCI, Myrinet, Quadrics, InfiniBand, …

Операционные системы: Windows, Linux (RedHat, Suse, Debian, …), Solaris, …

*R. Martin, A. Vahdat, D. Culler, T. Anderson. The Effects of Latency, Overhead and Bandwidth in a Cluster of Workstations. In Proc. 24th Int. Symp. on Com. Arch. (ISCA'97), June 2 - 4, 1997, pp 85 – 97.

Высокопроизводительный кластер (III)

За счет чего достигается рост производительности:

- рост производительности узлов
- развитие сети (масштабируемость - ~1000 узлов, пропускная способность – от 100 Мбит до 10 Гбит)

Рост производительности процессора

- повышение тактовой частоты и увеличение пропускной способности памяти
- параллелизм на уровне одного процессора:
  - конвейеризация,
  - суперскалярные вычисления (SSE, SSE2, SSE3 в процессорах Intel и AMD, Altivec Power IBM),
  - использование широких машинных слов (например в Itanium до 6 команд в слове)

Важное замечание: Без существенных успехов в области оптимизации программ во время компиляции усложнение архитектуры процессоров было бы невозможным из-за трудностей программирования

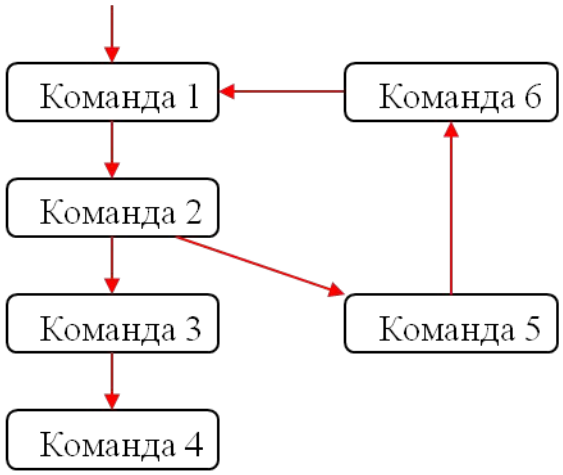
! Задачи оптимизации программ на уровне одного процессора успешно решаются современными компиляторами

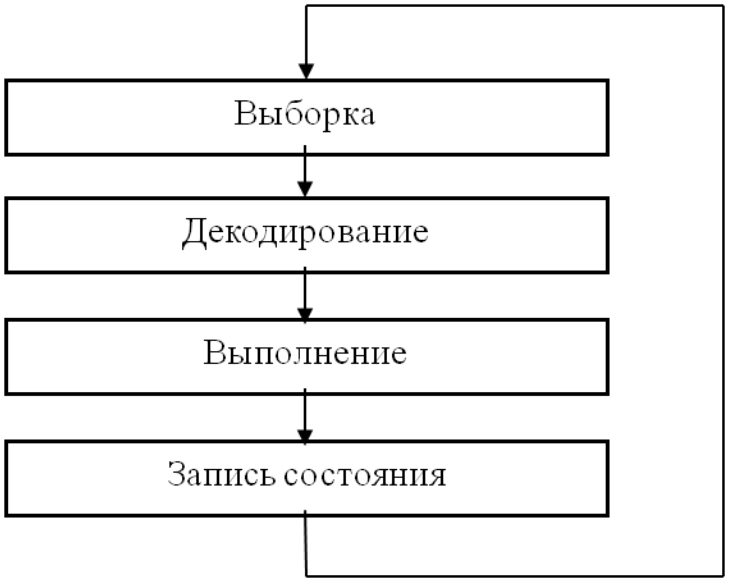
Сравнение одного из первых компьютеров Eniac и современного ПК

	Eniac	**ПК**
Год выпуска	1945	2007
Количество вентилей	18,000	6,000,000,000
Вес (*кг*)	27,200	1-3
Объем ($м^3$)	68	~0.005
Мощность (*вт*)	20,000	~50-300
Стоимость (US $)	4,630,000	~1,000
Объем памяти	~200	1,073,741,824
Производительность (Flop/s)	***800***	***5,000,000,000***

Архитектура фон Неймана

- Один поток вычислений, следующий изначальному порядку команд программы
- Каждая команда выполняется полностью перед началом выполнения следующей





Параллелизм на уровне команд (I)

- Часто последовательное выполнение является искусственным ограничением

```
x = y + z;
f = d - 100;
```

- Можно выполнить:
  - В любом порядке
  - Параллельно
  - С перекрытием
- Задача ILP: Выполнить как можно больше команд параллельно, убирая ненужные ограничения
- Ограничение: Необходимо сохранить корректность программы

Параллелизм на уровне команд (II)

Подходы к использованию ILP:

- Конвейеризация вычислений: параллельно работают разные стадии выполнения соседних команд
- Предсказание переходов: предугадать направление перехода и заранее (спекулятивно) выполнить команды по адресу перехода
- Аппаратное переупорядочивание команд (out-of-order): выдача команд в порядке, увеличивающем пропускную способность

```
r2 = load[r3]
jumpnz r2, label
mul r4, r4, r1
add r5, r5, r4
r6 = load [r5]
label:
mul r2, r4, r1
add r2, r5, r4
```

Решение аппаратуры:

```
r2 = load[r3]
jumpnz r2, label
label:
mul r2, r4, r1
add r2, r5, r4
```

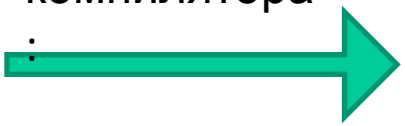
Параллелизм на уровне команд (III)

Архитектура с длинным командным словом:

- Компилятор упаковывает команды, которые должны выполняться параллельно, в одну “суперкоманду”
- Направление переходов подсказывается компилятором
- Спекулятивное выполнение команд тоже делается компилятором
- Плюсы: не нужна сложная аппаратура, компилятор может принять лучшие решения, т.к. вычисляет более точную информацию
- Минусы: более сложный компилятор

```
r2 = load[r3]
jumpnz r2, label
mul r4, r4, r1
add r5, r5, r4
r6 = load [r5]
label:
mul r2, r4, r1
add r2, r5, r4
```

Решение компилятора:



```
r2 = load[r3]
mul r2, r4, r1
add r2, r5, r4
jumpnz r2, label
mul r4, r4, r1
add r5, r5, r4
r6 = load [r5]
label:
check
```

Параллелизм на уровне команд (IV)

Параллелизм по данным (векторизация):

- Однотипные операции над элементами вектора можно выполнять параллельно
- Пример – множества команд SSE, SSE2, SSE3 в современных процессорах Intel и AMD
- Для этого компилятор должен преобразовать цикл обработки вектора (векторизация)

Было:

```
for(i=0; i<N; i++){
a[i] = a[i] + b[i];
}
```

Если можем обрабатывать 4 элемента сразу:

```
for (i=0; i<N; i+=4) {
a[i:i+4] = a[i:i+4] + b[i:i+4];    <-----   Одна машинная команда
}
```

Параллелизм на уровне команд (V)

Возможности использования параллелизма на уровне команд достигли предела!

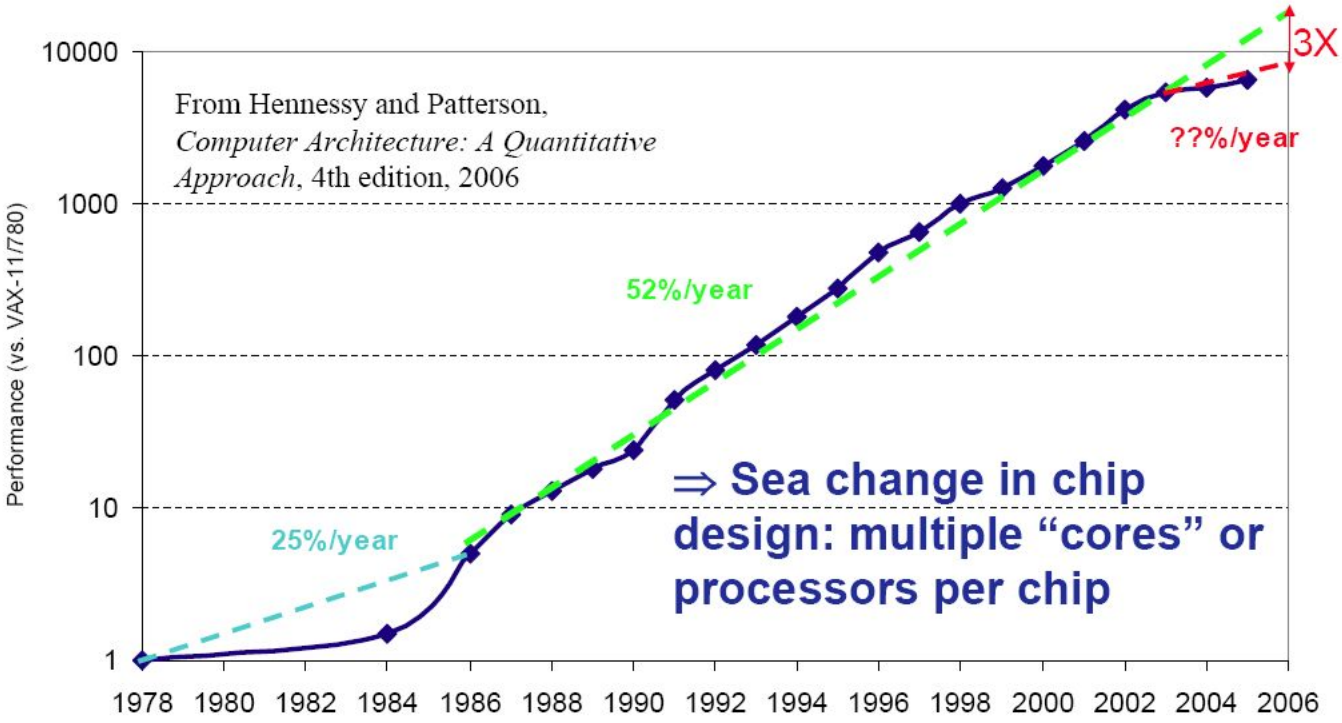
Улучшение компилятора/архитектуры не привели к существенному повышению производительности

Наш опыт: Для GCC был реализован агрессивный планировщик команд с поддержкой конвейеризации

- Основан на подходе селективного планирования
- Поддерживает ряд преобразований команд, выгодных для архитектуры EPIC (спекулятивное и условное выполнение, переименование регистров)
- Поддерживает конвейеризацию циклов
- На наборе тестов SPEC FP 2000 получено ускорение в среднем около 5%, на отдельных тестах до 10%
- Один из самых больших проектов в back-end’е GCC, патч включен в компилятор GCC 4.4.0 в качестве основного для уровня оптимизации –O3 для платформы Itanium
- Ускорение на Cell на некоторых тестах достигает 6-7%

Текущий тренд (I)

Производители (Intel, IBM, Sun) сделали ставку на параллелизм через multicore/manycore, т.к. развитие однопроцессорных систем уперлось в ряд фундаментальных проблем



Текущий тренд (II)

Фундаментальные проблемы:

- Было: энергия дешева, транзисторы дороги
  Стало: транзисторы дешевы, энергия дорога
  - на чип помещается больше транзисторов, чем можно запитать
- Было: память быстрая, вычисления дороги
  Стало: вычисления дешевы, память медленная
  - 4-6 циклов на FP операции, ~200 циклов на память
- Было: параллелизм на уровне команд (ILP) можно использовать через компилятор/архитектуру
  Стало: больше ILP выжать уже нельзя

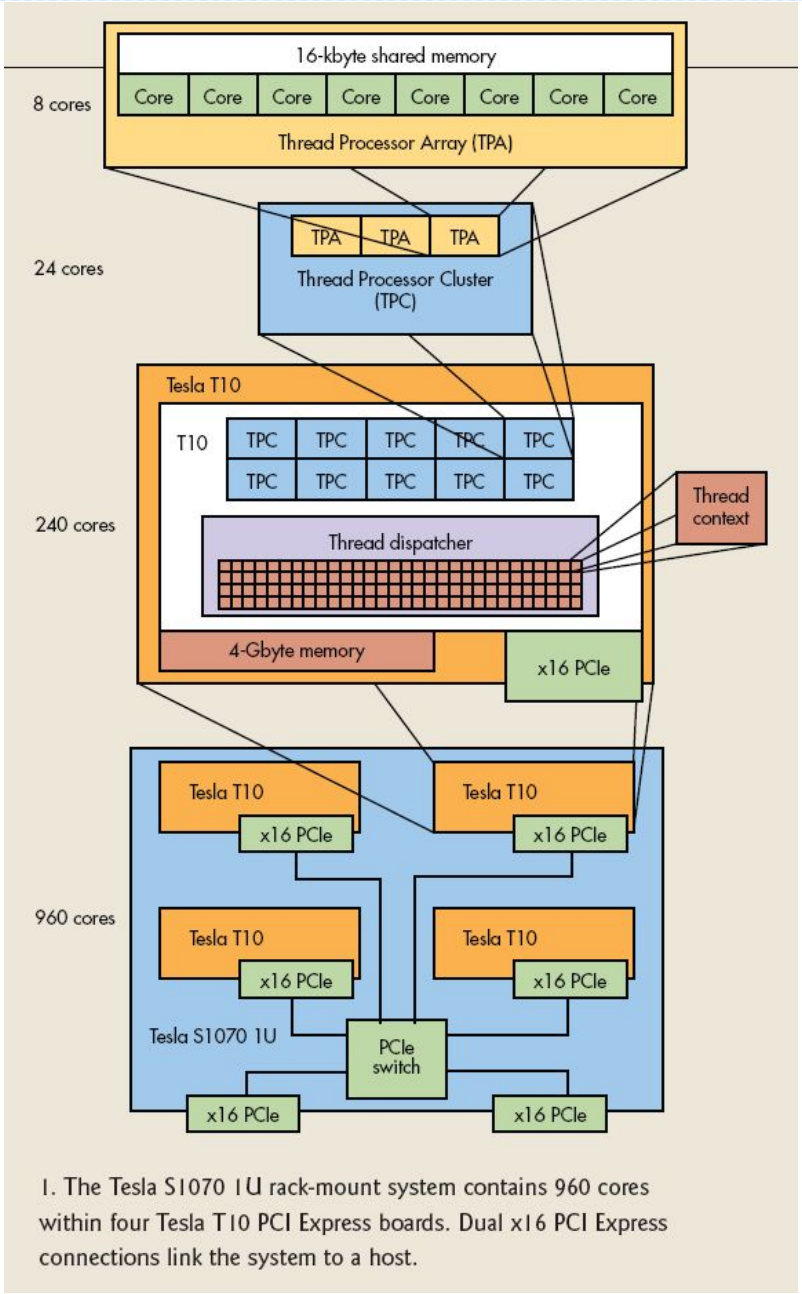
Текущий тренд (III)

- Расширение спектра задач решаемых с использованием графических акселераторов (Nebulae, Китай - 2-е место в Топ500 июнь, 2010 – на базе платформы Tesla, Nvidia)

- Попытки создать специализированные решения, например, Cell (IBM RoadRunner – 3-е место в Топ500 июнь, 2010 - построен на серверах на базе Cell)

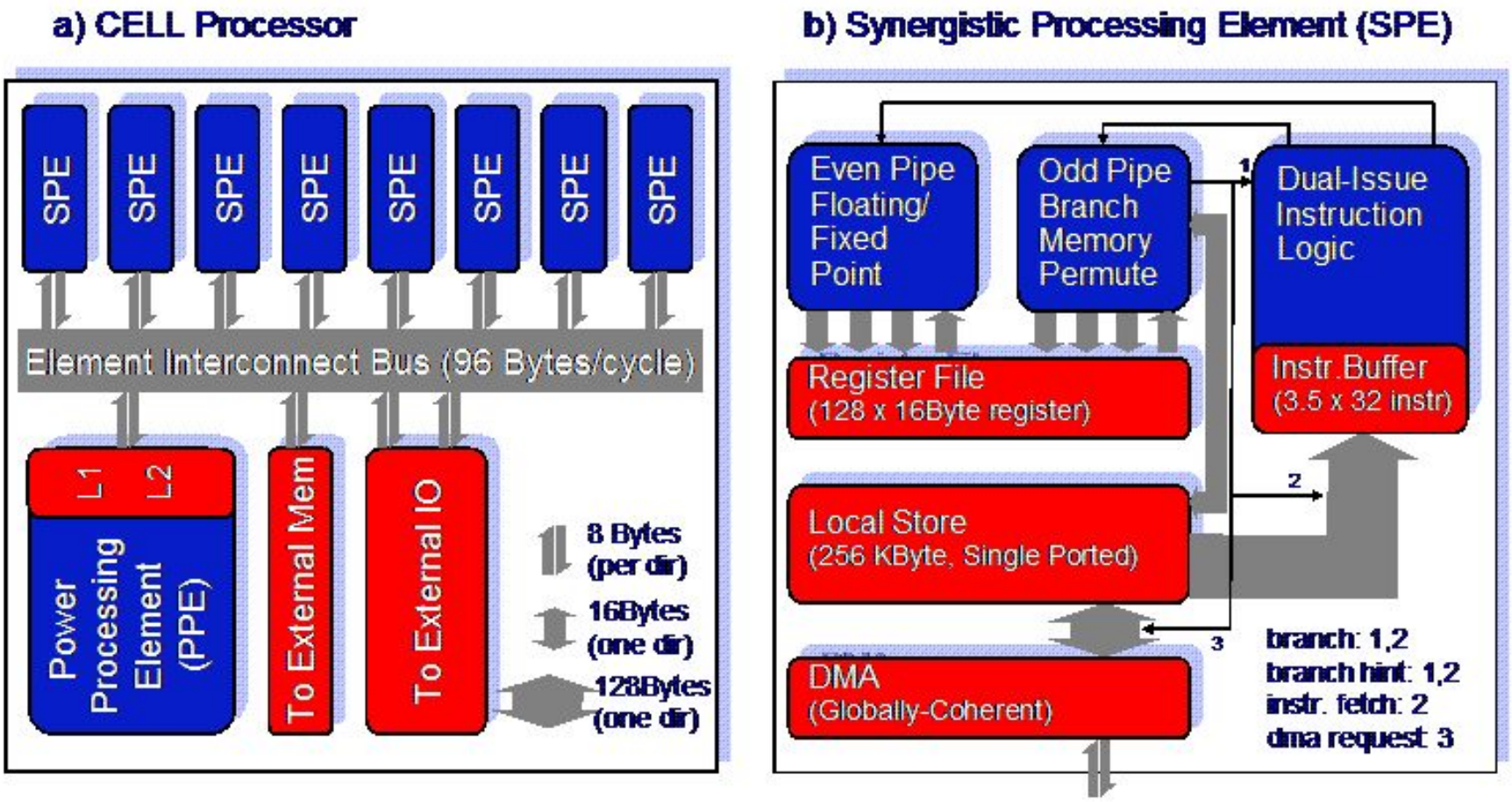
- Специализированные решения на базе FPGA

Архитектура Tesla



1. The Tesla S1070 1U rack-mount system contains 960 cores within four Tesla T10 PCI Express boards. Dual x16 PCI Express connections link the system to a host.

Архитектура Cell



Средства разработки ПО (I)

Проблема: обеспечение высокопродуктивных вычислений (компромисс между стоимостью разработки и сопровождения и эффективностью выполнения)

Мечта! Создание языка высокого уровня, позволяющего эффективно использовать существующие возможности аппаратуры, как в случае последовательных программ

Попытки создания параллельных языков высокого уровня не прошли: *HPF, Cilk (MIT), Unified Parallel C (Java version – Titanium) (Berkeley)* и др.

Пока не оправдались надежды, которые возлагались на языки нового поколения: *Fortress (Sun), Chapel (Cray), X10 (IBM)*

Средства разработки ПО (II)

// MPI вариант

```
while ((iterN--) != 0)
           {
             for(i = 2; i <= N; i++)
                   for(j = ((myidy == 0) ? 2:1); j < countsy[myidy] - 1; j++)
                   A[i][j] = (localA[i][j+1]+ 2 * localA[i][j] + localA[i][j-1])*0.25
                                                          + (B[i+1][j]+2 * B[i][j] + B[i-1][j])*0.25;
             for(i = 2; i <= N; i++)
                  for(j = ((myidy == 0) ? 2:0); j < countsy[myidy] - 1; j++)
                     localA[i][j] = A[i][j];
             if (myidy != 0)
                {
                  for (i = 0; i < N; i++)
                  upOut[i] = localA[i][1];
                  MPI_Send(upOut, N, MPI_DOUBLE, myidy - 1, TAG, MPI_COMM_WORLD);
                }
            if (myidy != sizey - 1)
                {
                  MPI_Recv(downIn, N, MPI_DOUBLE, myidy + 1, TAG, MPI_COMM_WORLD, &Status);
                  for (i = 0; i < N; i++)
                  downOut[i] = localA[i][countsy[myidy] - 2];
                  MPI_Send(downOut, N, MPI_DOUBLE, myidy + 1, TAG,MPI_COMM_WORLD );
                }
            if (myidy != 0)
                {
                  MPI_Recv(upIn, N, MPI_DOUBLE, myidy - 1, TAG, MPI_COMM_WORLD, &Status);
                  for (i = 0; i < N; i++)
                  localA[i][0] = upIn[i];
                }
            if (myidy != sizey - 1)
                {
                  for (i = 0; i < N; i++)
                    localA[i][countsy[myidy] - 1] = downIn[i];
                }
```

// HPF вариант

```
FORALL (J = 2:N, I=2:N) &
&       A(I,J)=(A(I,J+1)+2*A(I,J)+A(I,J-1))*0.25 &
&       + (B(I+1,J)+2*B(I,J)+B(I-1,J))*0.25
```

Средства разработки ПО (III)

Почему HPF не оправдал надежд:

- отсутствие компиляторных технологий, позволяющих генерировать эффективный параллельный код,
- отсутствие гибких стратегий распределения данных по узлам,
- отсутствие инструментария и др.

Ken Kennedy, Charles Koelbel, Hans Zima. “The Rise and Fall of High Performance Fortran: An Historical Object Lesson”// Proceedings of the third ACM SIGPLAN conference on History of programming languages, San Diego, California, Pages: 7-1 - 7-22, 2007.

Средства разработки ПО (IV)

В настоящее время фактическим языковым стандартом разработки промышленных прикладных программ является использование одного из языков программирования высокого уровня *(Fortran, C/C++)* с использованием

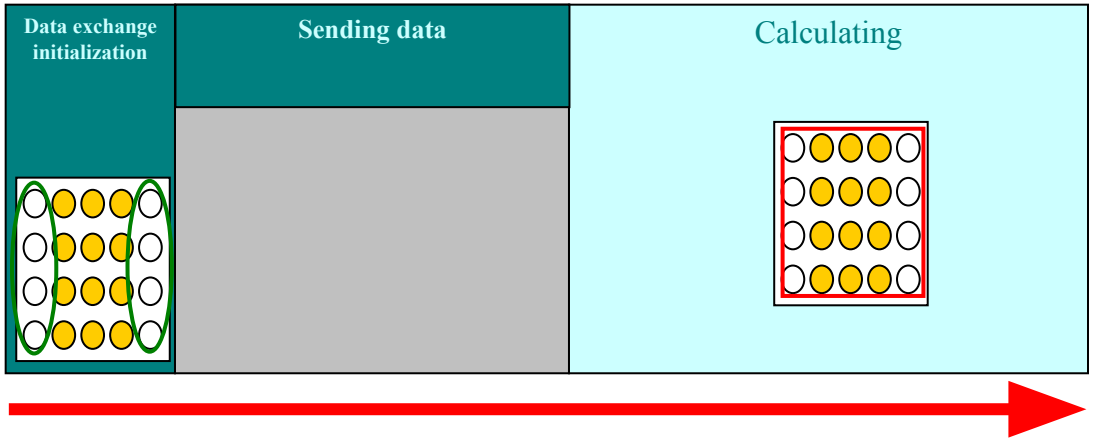
MPI (распределенная память),

OpenMP (общая память) или

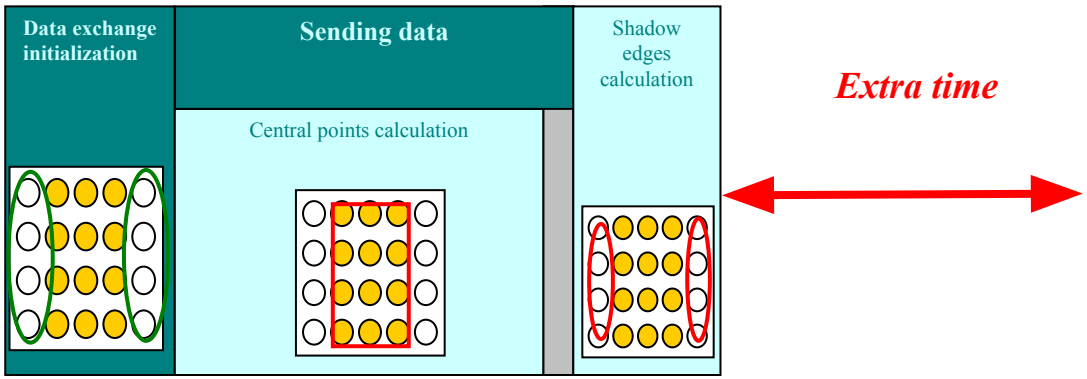
Cuda (OpenCL) для *GPGPU*

Программирование является искусством!

Пример доводки MPI-программы (1)

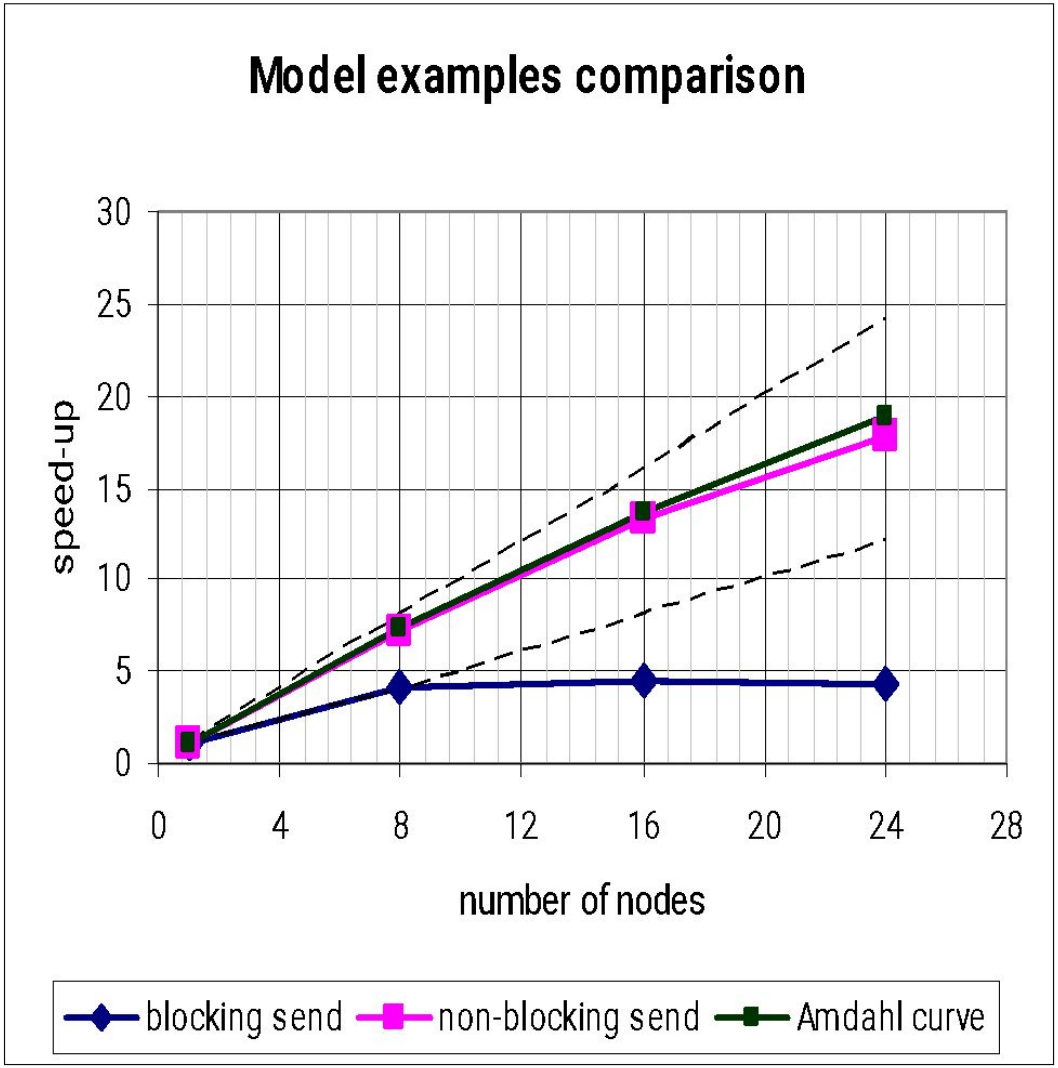



```
//sending
Send
Recv
//calculating
for (i = beg_i; i < end_i; i++)
    for (j = 0; j < N; j++)
        B[i][j] = f(A[i][j]);
```

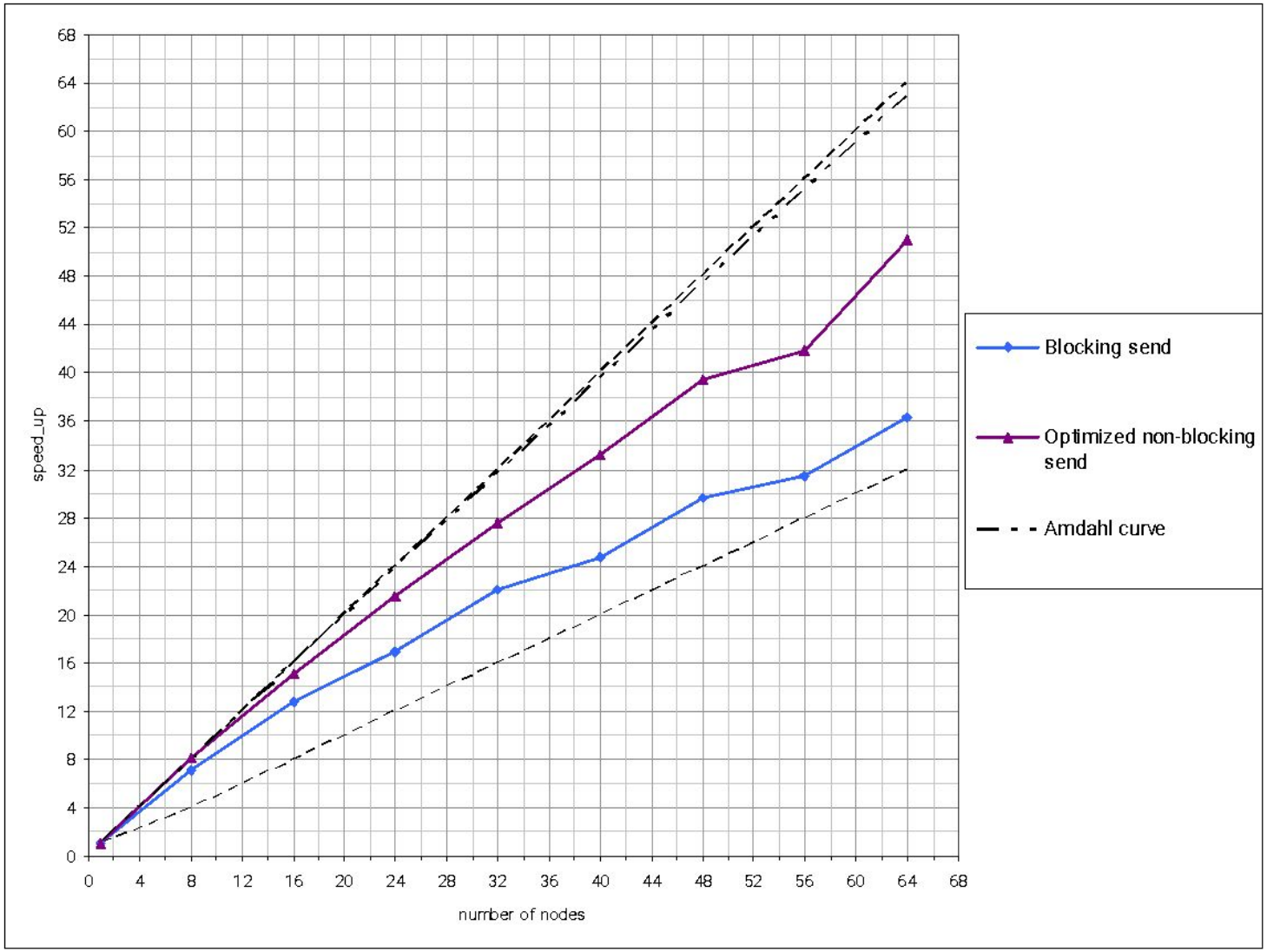



```
//sending
ISend
IRecv
//calculating
for (i = 1; i < N - 1; i++)
    for (j = 0; j < N; j++)
        B[i][j] = f(A[i][j]);
//waiting
Wait();
//calculating last columns
if (myid != 0)
    for (j = 0; j < N; j++)
        B[0][j] = f(tempL[j]);
if (myid != proc_size - 1)
    for (j = 0; j < N; j++)
        B[N - 1][j] = f(tempR[j]);
```

Пример доводки MPI-программы (2)



Пример доводки MPI-программы (3)
Моделирование торнадо



Эффективные программы для GPU

- Необходимо учитывать ряд особенностей
  - Количество нитей
  - Загруженность мультипроцессора, зависящая от:
    - Количества регистров на нить
    - Объем разделяемой памяти на блок нитей
    - Количество нитей в блоке
  - Конфликты разделяемой памяти
  - Слияние обращений к памяти (memory coalescing)
- Умножение плотных матриц: оптимальный размер блока зависит от пропускной способности как памяти, так и арифметических устройств GPU

Разреженные матрицы

- Специализированные форматы хранения
- Типичная задача – умножение на плотный вектор (SpMV)

$$y = y + Ax : y_i = y_i + \sum_{j=1}^{N} A_{ij} x_j$$

- 2 FLOPS на ненулевой элемент – ограничено пропускной способностью памяти
  - Производительность непосредственно зависит от формата хранения матрицы
- Результаты Nvidia (Bell, Garland 2009)
  - 10-кратное ускорение по сравнению с 2-ядерным Opteron

Предлагаемый формат (“sliced” ELLPACK)

Значения Столбцы

$$\left(\begin{array}{cc|c} a & b & \\ & c & d \\ \hline & e & \\ & & f \end{array}\right) \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 0 & 2 \\ 1 & 3 \end{pmatrix} \quad \begin{pmatrix} e \\ f \end{pmatrix} \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

sliceptr	0				4	6
column	0	1	2	3	1	3
value	a	c	b	d	e	f

- Матрица делится на полоски из строк
- Каждая полоска хранится в формате ELL
  - Количество ненулевых элементов разное для разных полосок -> можно настроить
  - Один блок нитей на полоску
  - Простой поток управления (важно для GPU)
  - Можно настроить количество нитей в блоке

Улучшения алгоритма (I)

- Переупорядочивание строк
  - Хочется расположить рядом строки с одинаковым количеством ненулевых элементов
  - Дорогая оптимизация, имеет смысл при многократном умножении на одну и ту же матрицу (для итеративных алгоритмов)
  - Переупорядочивание матрицы ухудшает локальность доступа к элементам вектора x
- Полоски переменной высоты
  - Полезны для сильно неоднородных матриц, когда переупорядочивание помогает лишь частично
  - Формировать узкие полоски из строк с большим количеством ненулевых элементов и наоборот

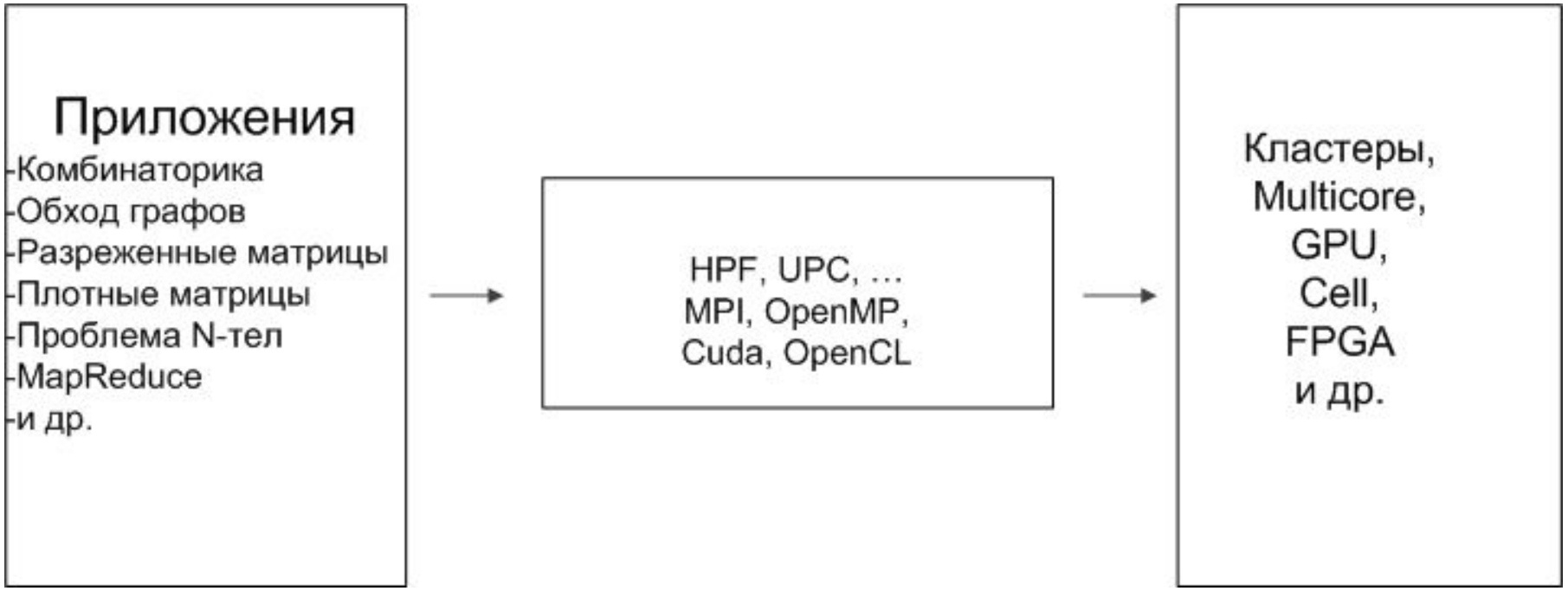
Улучшения алгоритма (II)

- Параметры реализации:
  - Зависящие от устройства
    - Количество строк на полоску
    - Количество нитей на блок
  - Зависящие от матрицы
    - Свобода переупорядочивания
    - Полоски переменной высоты
- Оптимальные значения параметров неочевидны
- Поддерживается автоматическая настройка (во время выполнения выбирается наиболее быстрый вариант)

Результаты тестирования

- Варианты алгоритма сравниваются с реализацией от NVIDIA как базовой
- Тесты – набор матриц, использующийся в работах NVIDIA и университета Беркли
- Все варианты включают автонастройку для конкретного устройства и матрицы
- Используется карта NVIDIA GTX280
- Лучший вариант, выбранный автонастройкой: ускорение до двукратного (47% в среднем)
- Достигается 24-97% теоретической пропускной способности памяти для неблочных методов (в среднем 64%)

Пути обеспечения высокопродуктивных вычислений (I)



- Создание технологий для классов приложений
- Разработка библиотек и пакетов прикладных программ
- Организация междисциплинарных команд

Пути обеспечения высокопродуктивных вычислений (II)

Программная модель Map/Reduce

Вычисления производятся над множествами пар (k, v), где k – ключ, v – значение
Функция Map в цикле обрабатывает каждую пару из множества входных пар и производит множество промежуточных пар:

Map(k1,v1) -> list(k2,v2)

Среда MapReduce группирует все промежуточные значения с одним и тем же ключом I и передает их функции Reduce, которая получает значение ключа I и множество значений, связанных с этим ключом:

Reduce(k2, list (v2)) -> list(v3)

В типичных ситуациях каждая группа обрабатывается (в цикле) таким образом, что в результате одного вызова функции образуется не более одного результирующего значения

Пути обеспечения высокопродуктивных вычислений (III)

```
// Функция, используемая рабочими узлами на Map-шаге
// для обработки пар ключ-значение из входного потока
map(String name, String document):
    // Входные данные: ключ - название документа
    //                 значение - содержимое документа
    // Результат:      ключ – слово, значение - всегда 1
    for each word w in document:
        EmitIntermediate(w, "1");

// Функция, используемая рабочими узлами на Reduce-шаге
// для обрабоки пар ключ-значение, полученных на Map-шаге
reduce(String word, Iterator partialCounts):
    // Входные данные: ключ – слово, значение - всегда 1.
    // Количество записей в partialCounts и есть требуемое значение
    // Результат: общее количество вхождений слова word во все
    //            обработанные на Map-шаге документы
    int result = 0;
    for each pc in partialCounts:
        result += parseInt(pc);
    Emit(AsString(result));
```

Пути обеспечения высокопродуктивных вычислений (IV)

Существуют эффективные масштабируемые реализации MapReduce (компания Google, система Hadoop) в рамках распределенной среды:

- ***десятки тысяч узлов*** (вполне вероятны отказы отдельных узлов) например, каждый запрос обрабатывается в Google в среднем ***20000 серверов***

- для управления данными, хранящимися на этих дисках, используется распределенная файловая система – Google (GFS), Hadoop (HDFS) – ***петабайты данных***;

Пути обеспечения высокопродуктивных вычислений (V)

- Системы автоматизации инженерного анализа (CAE системы)

ANSYS, CFX, LS_DYNA, ICEM CFD, STAR-CD, FLUENT, Diffpack, COMET/Acoustics, FlowVision, OpenFoam..

- Химия

Gaussian, Dirac, HyperChem, GROMACS, PC-GAMESS, NWChem,..

- Численный анализ

MatLAB, Wolfram Mathematica, SCALAPACK, BLAS, MKL,..

- Нефтегаз

TEMPEST, ECLIPSE

- Метеорология

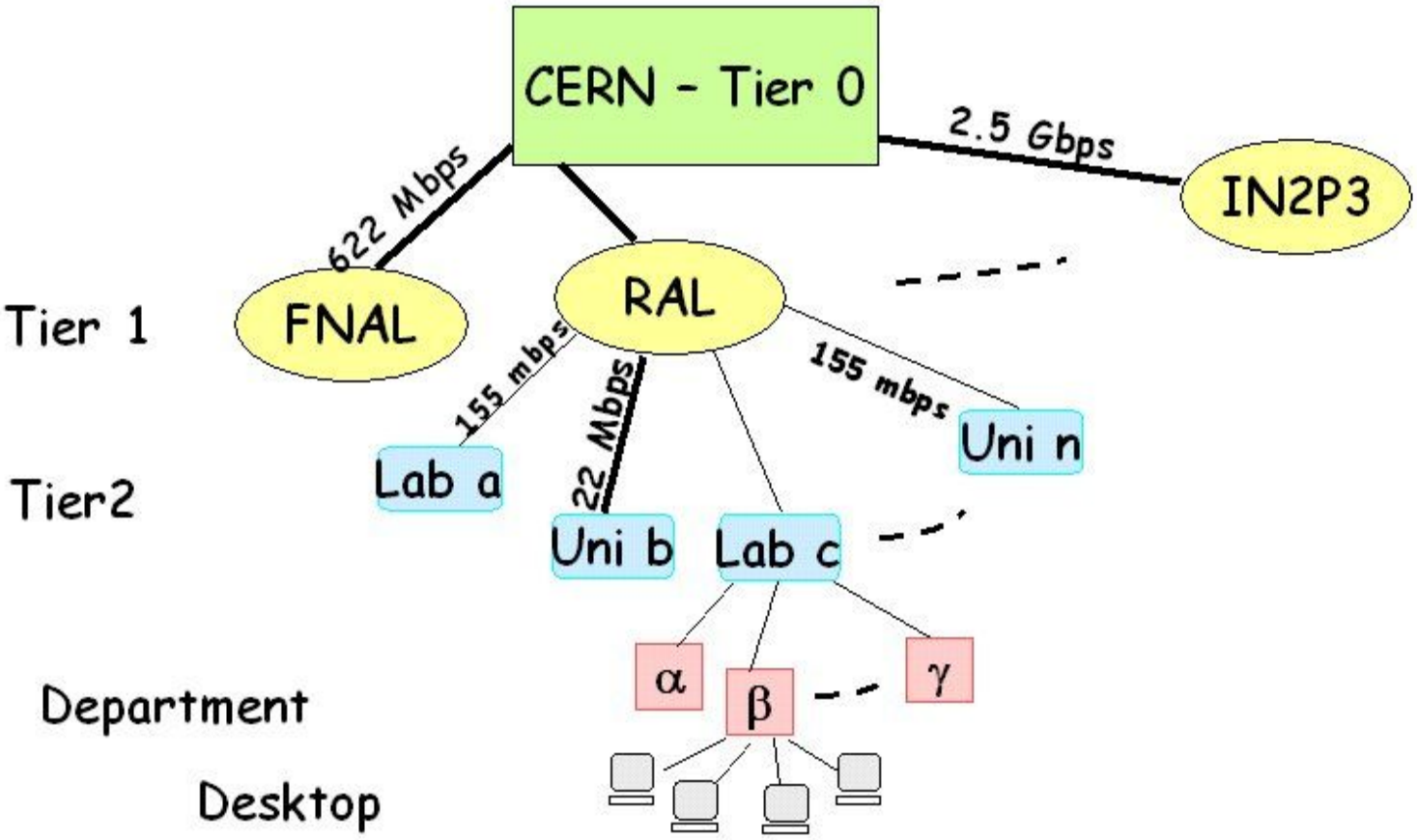
Модель ММ5, Модель ECHO-G …

Цели Grid*

- Обеспечение возможности решения крупномасштабных научно-технических задач
- Организация работы распределенных (административно и географически) коллективов
- Оптимизация использования вычислительных ресурсов

*объединение вычислительных ресурсов («виртуальные организации») для их совместного использования (ресурсы могут принадлежать разным организациям и могут быть расположены в различных географических и административных областях)

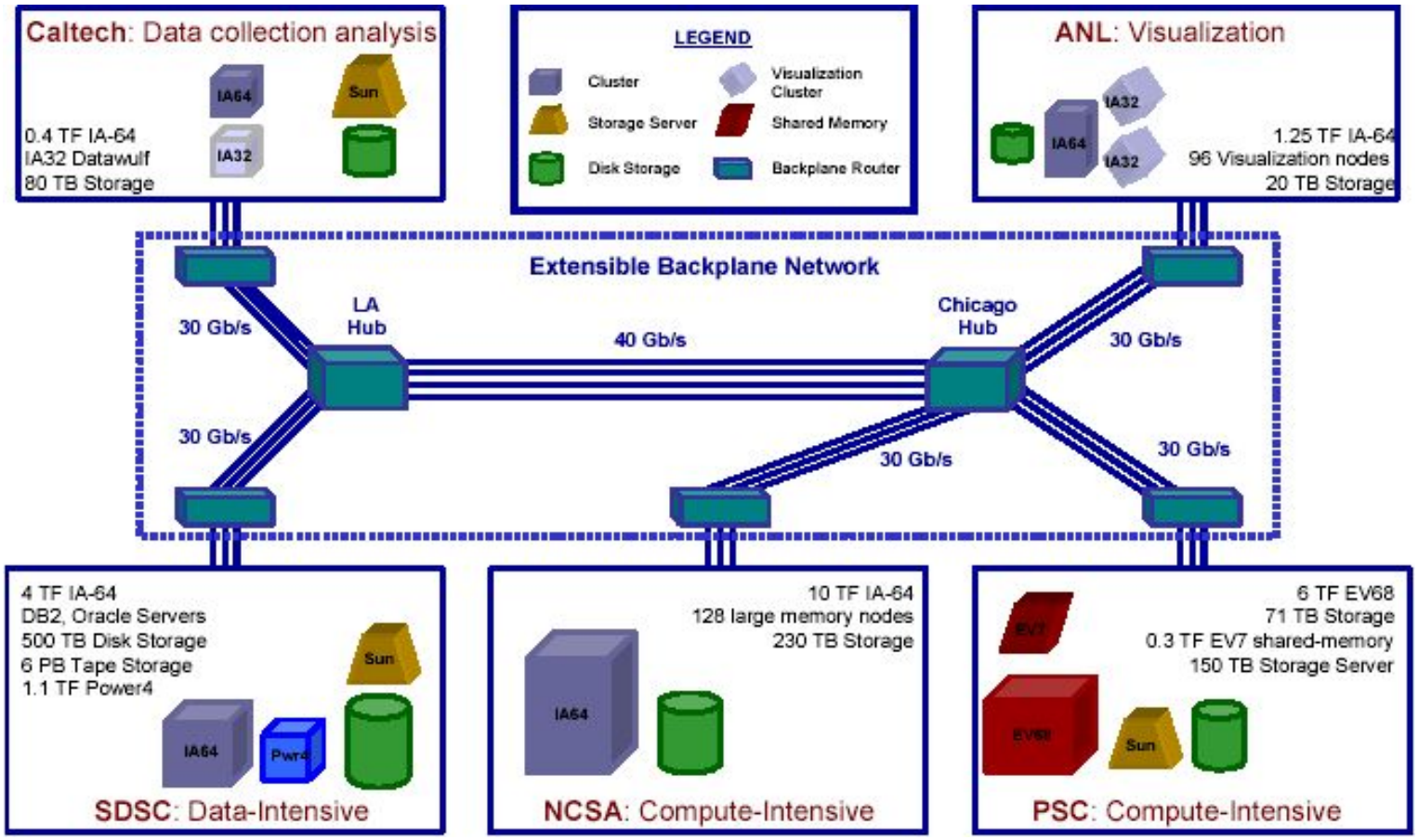
Проект CERN


CERN - Tier 0
2.5 Gbps
IN2P3
622 Mbps
FNAL
RAL
Tier 1
155 mbps
155 mbps
22 Mbps
Lab a
Uni n
Tier2
Uni b
Lab c
α
β
γ
Department
Desktop

Globus Toolkit

- Реализует стандарт OGSI
- Реализует аутентификацию на основе стандарта X.509
- Содержит контейнер Grid служб
- Содержит сервер и клиента GridFtp
- Поддерживает запуск пользовательских задач (Globus Resource Allocation Management)

Пример стенда – TeraGrid



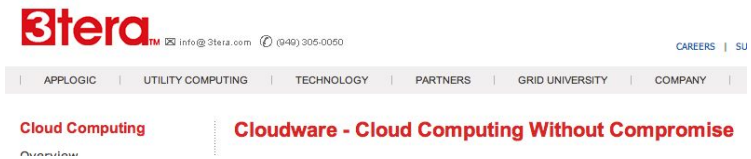
Концепция «Облачных вычислений»

- Все есть сервис (XaaS)
  - AaaS: приложения как сервис
  - PaaS: платформа как сервис
  - SaaS: программное обеспечение как сервис
  - DaaS: данные как сервис
  - IaaS: инфраструктура как сервис
  - HaaS: оборудование как сервис

- Воплощение давней мечты о компьютерном обслуживании на уровне обычной коммунальной услуги:
  - масштабируемость
  - оплата по реальному использованию (pay-as-you-go)

Компании предлагающие «Облачные» решения (небольшая выборка)

Amazon Elastic Compute Cloud (Amazon EC2) - *Beta*



Ожидаемый рост рынка облачных вычислений к 2015 г. до 200 млрд. долларов

Примеры внедрения «Облачных» решений

- Nebula – «облачная» платформа NASA
- RACE – частное облако для DISA (Defence Information Systems Agency)
- ВВС США – заказали и подписали контракт с IBM на разработку защищенной инфраструктуры облачных вычислений, способной поддерживать оборонительную и разведывательную сеть
- Panasonic – предоставление сервисов на основе IBM cloud, для эффективного взаимодействия с поставщикам
- Муниципалитет города Los Angeles переводит свою IT-инфраструктуру в облако, в частности, электронную почту в *Gmail*
- Муниципалитет города Miami совместно с Microsoft разработал систему регистрации и отображения на карте неаварийных ситуаций (*Microsoft Windows Azure*)

Правительственные инициативы по «Облачным» решениям

- G-Cloud – Правительственное облако Великобритании, которое опирается на инициативу: *«Deliver on Open Source, Open Standards and Reuse Strategy»*
- Kasumigaseki Cloud – правительственное облако Японии, которое, в том числе, используется для реализации элементов электронного правительства
- Federal Cloud Computing Initiative (США) – различные аспекты применения облачных вычислений в государственных учреждениях и бизнесе
- Европейское агентство по охране окружающей среды (EEA) разработало платформу Eye On Earth, которая позволяет собирать информацию о большом количестве климатических и экологических факторов и отображать их на карте

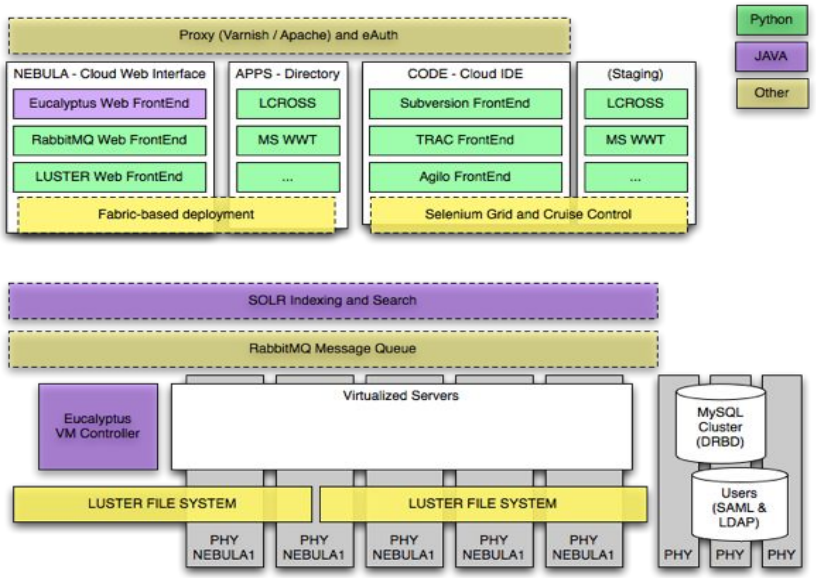
NEBULA Cloud Computing Platform

Nebula (туманность) – это проект который разрабатывается в Исследовательском центре Эймс а (NASA) целью которого является интеграция компонент свободного ПО в единую инфраструктуру обеспечивающую высококачественный вычислительные сервисы по предоставлению мощностей, хранению данных и сетевых подключений. Nebula в настоящее используется в открытых образовательных и исследовательских проектах.

В основе проекта лежат открытое ПО и предлагаются следующие сервисы:

- Infrastructure as a Service
- Platform as a Service
- Software as a Service

http://nebula.nasa.gov/



Свободное ПО и «Облачные вычисления»

Одно из основных направлений развития

- Стандартный стек системного ПО
- Распространение свободного ПО: Linux, Xen, Tashi, Hadoop, VNC, десятки прикладных пакетов и др.

Существующий уровень свободного ПО дает возможность организации «облачных вычислений» на всех уровнях

- Nebula – «облачная» платформа NASA реализована на основе компонент из свободного ПО
- Компания Yahoo! объявила, что в 2011 г. вся используемая ее платформа будет иметь статус свободного ПО

Почему сейчас?

Создание чрезвычайно крупномасштабных центров обработки данных

- в ~10 раз снижение стоимости (использование систем построенных из компонент общего назначения, дешевые помещения, масштаб и др.)

Кроме того:

- Всеобъемлющий широкополосный Интернет
- Быстрая виртуализация (зависимость программы от платформы существенно ослаблена)
- Стандартный стек системного ПО
- Распространение свободного ПО

Примеры применения

◊ Конвертирование большого количества файлов из одного формата в другой (пакетная обработка)

Washington post: 17.5 тыс. стр. документации – 1500 серверчасов – 200 EC2

◊ Обработка запросов в Google (MapReduce)

несколько тысяч запросов в секунду, каждый запрос – 20000 серверов

◊ Перенос в «облако» приложений, выполняемых на ПК *Matlab*

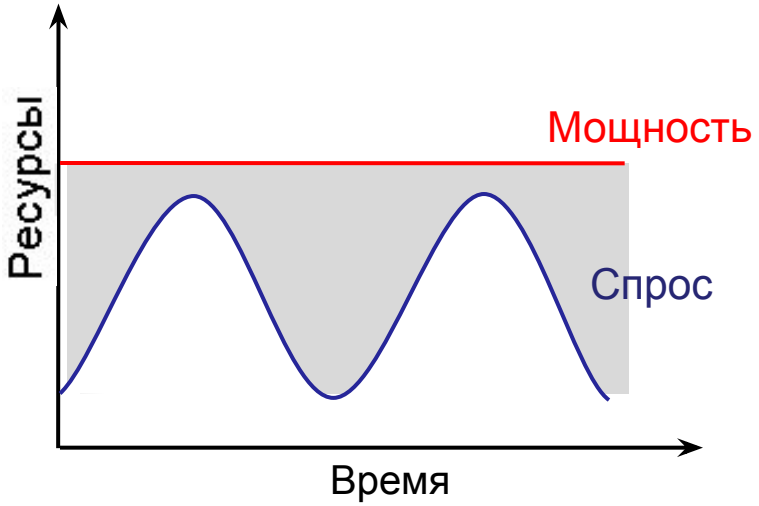
◊ Доступ к прикладным пакетам, рассчитанным на высокопроизводительные вычисления

NanoHub

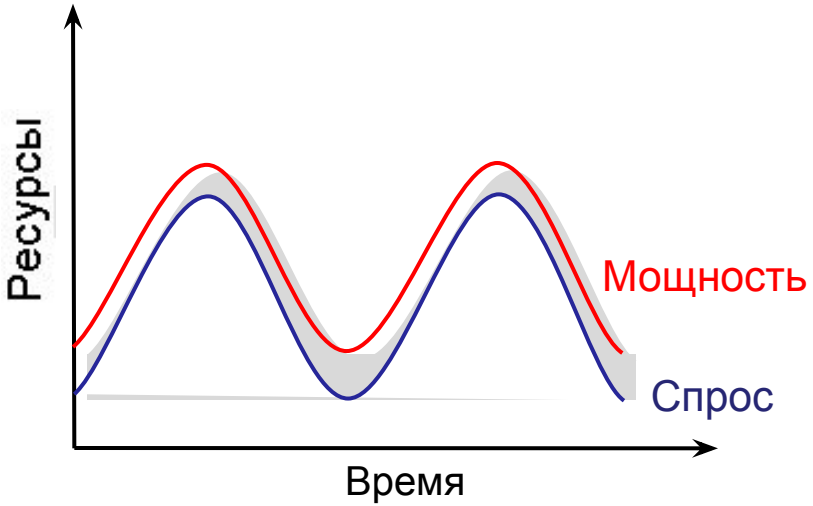
Краткосрочные пиковые нагрузки

Преимущества «облачного» ЦОДа

- Гибкость предоставления ресурсов *может* обеспечить беспрецедентную экономию – минимум неиспользуемых ресурсов



Обычный вычислительный центр



Облачный ЦОД

Неиспользуемые ресурсы

«Облачные вычисления» в науке и образовании (1)

Возможность создания web-ориентированных лабораторий (хабов) в конкретных предметных областях (объединение современных концепций web 2.0 с возможностью доступа к прикладным моделям):

- *интерактивный доступ к инструментам моделирования;*
- *поддержка распределенной разработки (система контроля версий, инструмент управления проектами и отслеживания ошибок);*
- *механизмы добавления новых ресурсов;*
- *информационные ресурсы (wiki, презентации и др.);*
- *поддержка пользователей;*
- *визуализация результатов и др.*

«Облачные вычисления» в науке и образовании (2)

- Принципиально новые возможности для исследователей по организации доступа, разработке и распространению прикладных моделей (*следствие возможность создания сообществ профессионалов в специализированных областях, стандартизация используемого инструментария, форматов хранения данных и др.*)

- Принципиально новые возможности по передаче знаний: лекции, семинары (практические занятия), лабораторные работы и др.

Общая схема организации «хаба»

Интерактивные инструменты моделирования

Механизм интеграции ресурсов

Средства поддержки пользователей

Информационные Wiki разделы, блоги

Поддержка разработки новых инструментов моделирования

Документация, интерактивные курсы

Социальные сети пользователей

Linux, Apache, LDAP, PHP, Joomla, MySQL, Sendna (ИСП РАН), Xen, Handoop, VNC, Rappture Toolkit,

Globus, Condor-G, gLite,…C

Open Science Grid

TeraGrid

МИНИСТЕРСТВО СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное агентство по делам молодежи

УНИВЕРСИТЕТСКИЙ КЛАСТЕР

ИСП РАН

hp invent

СИНТЕРРА

«Университетский кластер» (1)

Программа учреждена 4 сентября 2008 года Российской академией наук (ИСП РАН и МСЦ РАН), компаниями HP и «Синтерра»

Цель:

- повышение уровня **компетенций** в параллельных и распределенных вычислениях в образовательной и научно-исследовательской деятельности
- создание **сообщества** специалистов использующих и разрабатывающих современные технологии
- **передача** знаний и технологий в Российскую индустрию (энергетика, машиностроение, транспорт, связь и пр.)

«Университетский кластер» (2)

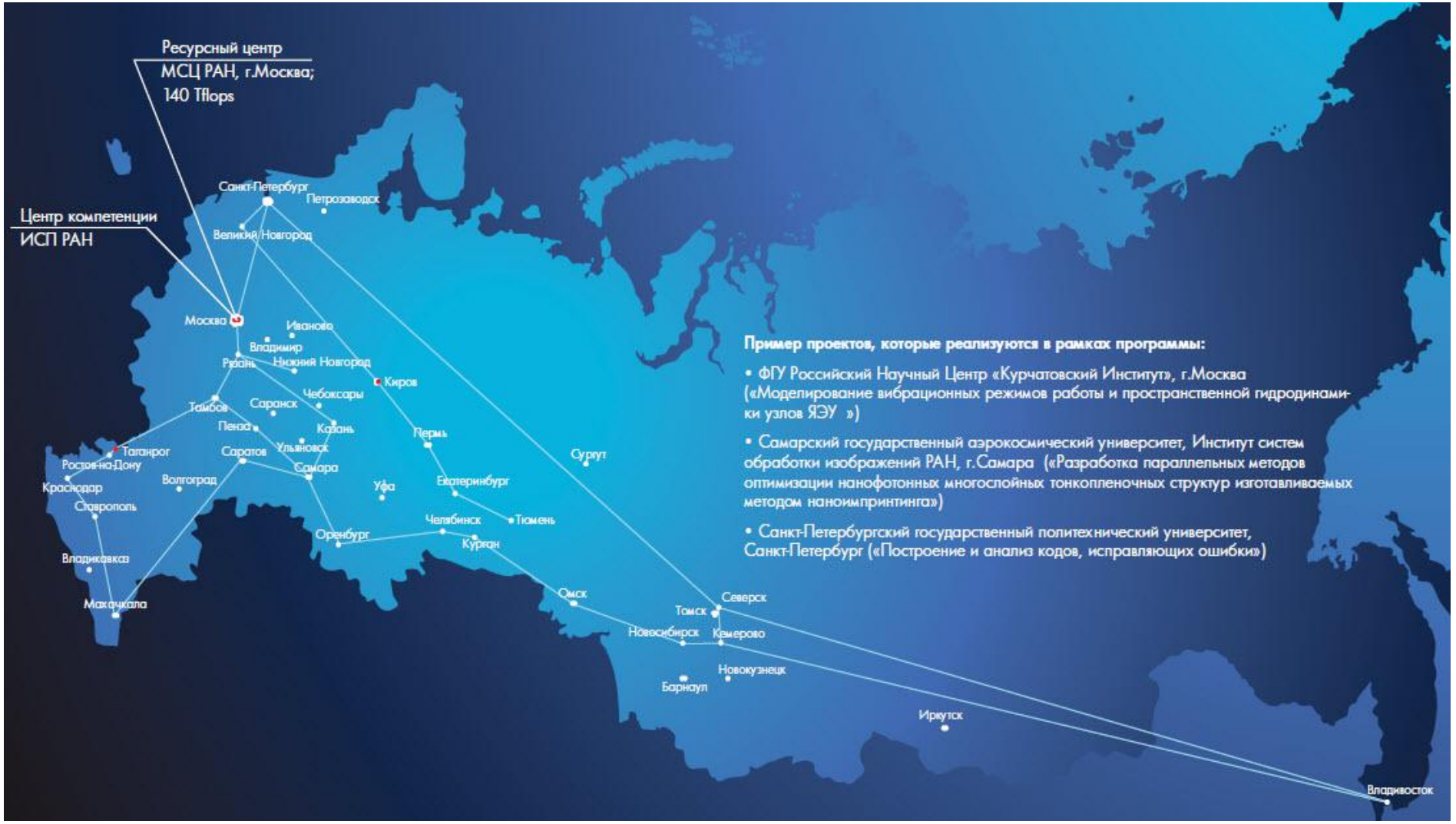
Для достижения целей Программы решаются следующие задачи:

- построение, развитие и поддержка **вычислительной инфраструктуры**;
- создание и развертывание на базе вычислительной инфраструктуры **сервисов** различных уровней (в модели «облачных вычислений»);
- развертывание на базе вычислительной инфраструктуры **испытательных стендов**, на которых можно будет осуществлять проверку эффективности, разработку и доводку, новых концепций и парадигм программирования, новых информационных технологий
- создание учебных планов, учебных программ и средств поддержки учебных курсов
- создание и развертывание предметно-ориентированных научно-исследовательских **web-лабораторий** («хабов»)

«Университетский кластер» (3)

Инфраструктура включает в себя современные аппаратные, программные, сетевые технологии, а также компетенцию передовых научных центров

«Университетский кластер» (4)



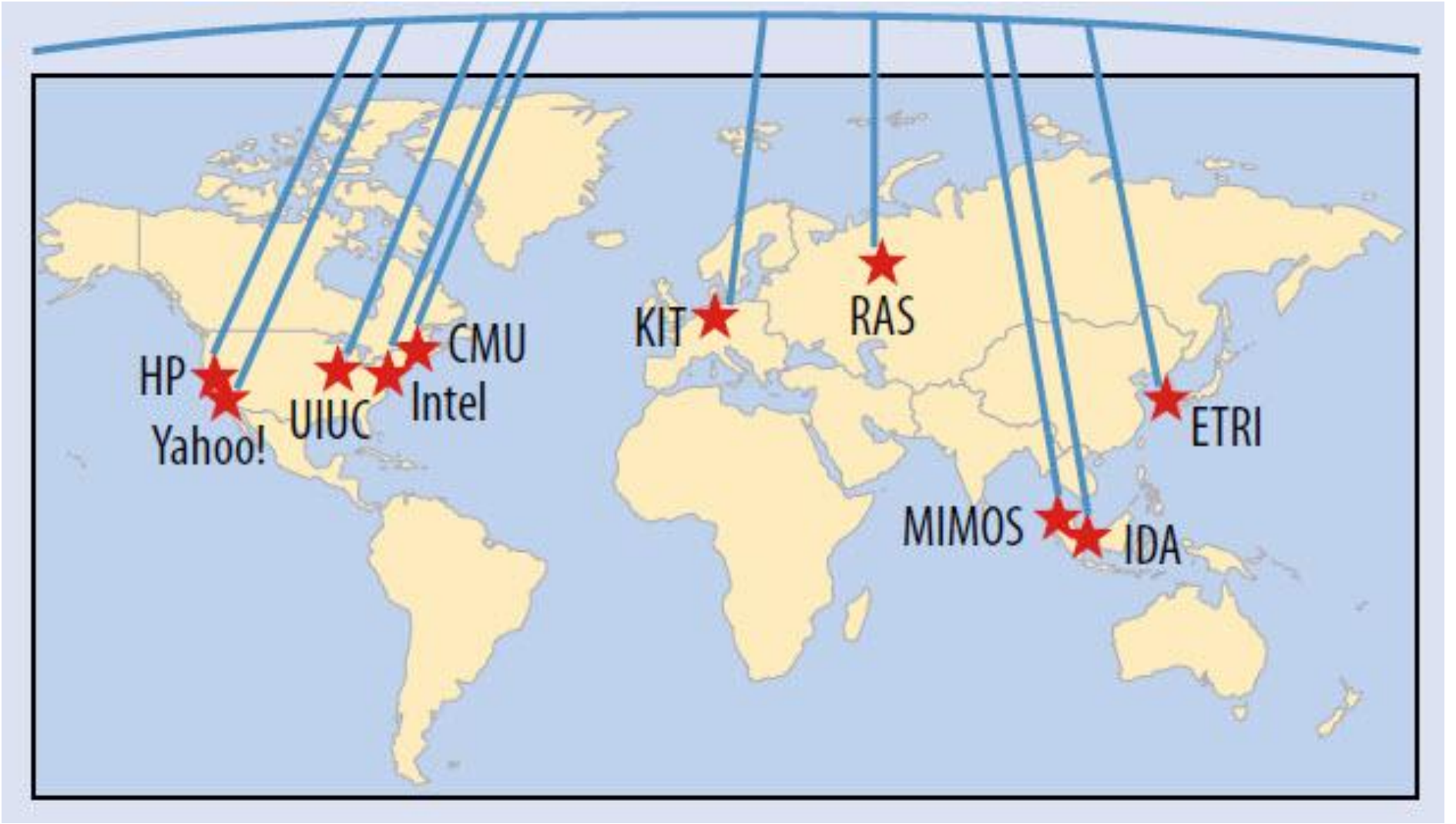
«Университетский кластер». Проект OpenCirrus

OpenCirrus был основан компаниями HP, Intel и Yahoo

Цель – создание открытого испытательного стенда на базе распределенных центров обработки данных, который призван поддержать разработчиков, как прикладных, так и системных программных средств в новой инновационной области «облачных вычислений»

Российская Академия наук, в составе ИСП РАН, МСЦ РАН и **РНЦ «Курчатовский институт»**, стала первой (июнь 2009) в Восточной Европе и седьмой в мире организацией, присоединившейся к программе OpenCirrus, став одним из семи «центров компетенции» (Center of Excellence, COE)

OpenCirrus – география проекта



«Университетский кластер». Текущее состояние (1)

Реализованы базовые сетевые службы VPN «Университетский кластер» (*служба доменных имен DNS, централизованная авторизация, каталог ресурсов*)

Разворачиваются базовые сервисы:

«рабочее место» разработчика, обеспечивающего, в том числе, совместную разработку распределенных коллективов

«виртуальная аудитория» - возможности по проведению лекций, лабораторных работ в режиме «он-лайн»

Сервисы стенда в рамках проекта OpenCirrus:
Physical Resource Set (*Tycoon*), Elastic Compute (*Tashi*),
Группа сервисов, обеспечивающих работу с большими распределенными массивами данных (*Hadoop*).

Более 20 заявок на использование этих сервисов

«Университетский кластер». Текущее состояние (2)

ИСП РАН совместно с **РНЦ «Курчатовский институт»** и компанией **HP**, на базе открытого пакета OpenFOAM реализует сервис:
CFD Compute – решение задач механики сплошной среды

Для обеспечения полного цикла решения задач в рамках сервиса ***CFD Compute***, а также других задач инженерного анализа реализуются сервисы:

Scientific Visualisation на базе пакета ***ParaView***;

CAD Compute (инструмент построения расчетных сеток) на базе открытого пакета ***SALOME***

Пакетизация идет с конца 80-х гг.

Универсальные пакеты – PHOENICS, ANSYS, Nastran, FIDAP, StarCD, FIRE, FLUENT, CFX и др.
Монополизация – слияние FLUENT и CFX на базе ANSYS

Ansys CFD : Полная стоимость одного профессионального рабочего места в год составляет 55236 евро с возможностью распараллеливания кода всего на 4 ядра, и далее:

3-32 ядра за каждые 4: 1600 евро; 33-128 ядра за каждые 4: 540 евро;

Стоимость распараллеливания кода на 128 ядер составляет около 100 000 евро.

Российские пакеты: FlowVision, SINF, GDT

FlowVision - Коммерческая лицензия в год
Рабочее место 297000 руб.
Параллельный счет: 128 ядер= 3.2 млн руб.
Академическая лицензния: только бессрочная
Стоимость рабочего места 124000 руб.
Стоимость распаралеливания кода на 128 ядер составляет 795 000 руб.

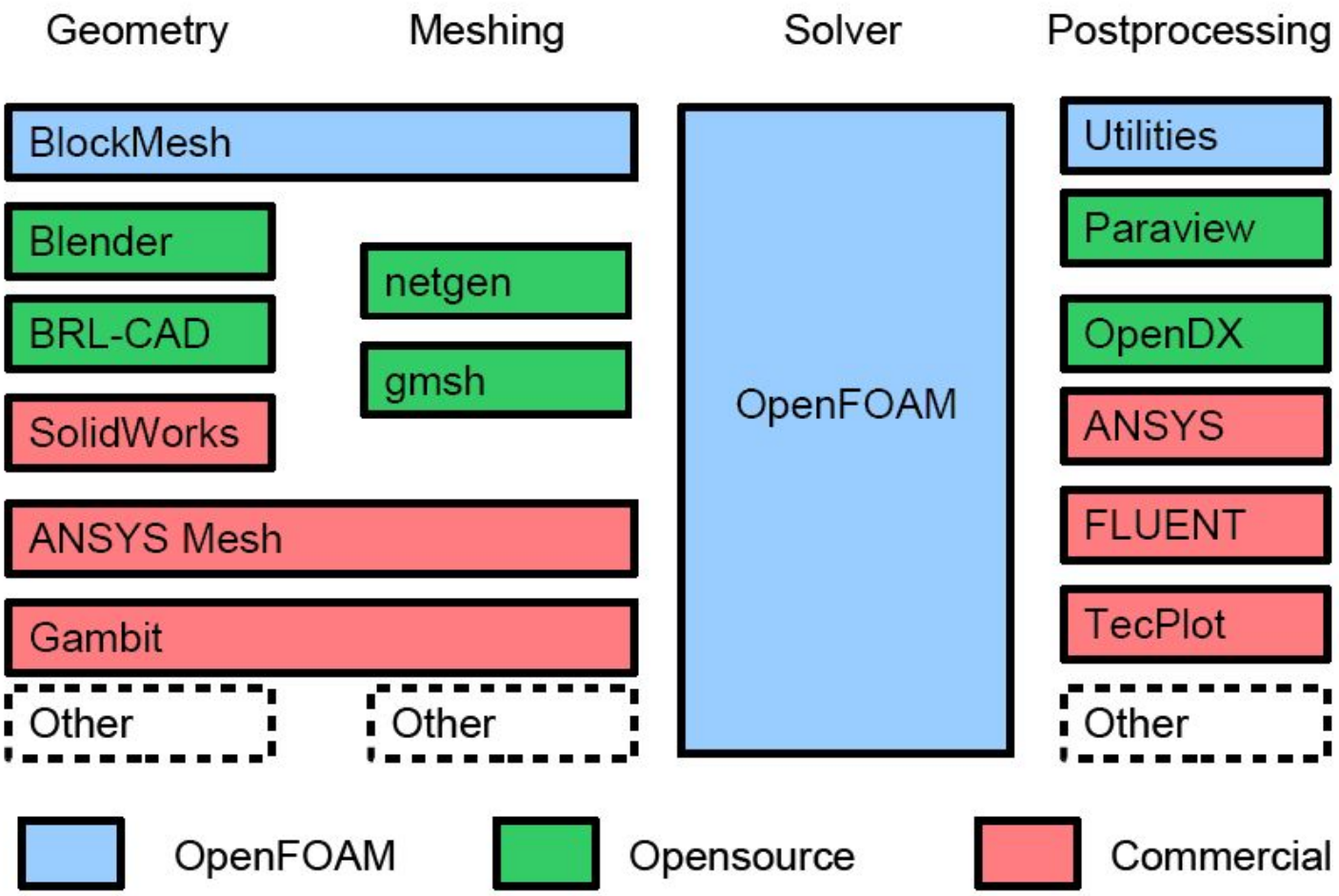
Кризис – Д. Б. Сполдинг (2007): «Коммерческий пакет – тормоз развития»!!!

Свободное Программное Обеспечение

open-source „process chain“

- geometry modelling
  - Blender
  - Salome
- geometry import and surface meshing
  - Gmsh
  - NETGEN
- isentropic volume meshing (tetras)
  - NETGEN
  - Tetgen → ATTENTION: not allowed for commercial applications, NOT OPEN-SOURCE
- an-isotropic boundary layer grids
  - ... → Engrid
- solver
  - OpenFOAM
  - Code Saturne
  - Elmer
- visualisation
  - ParaView
  - Open Data Explorer

Основные этапы и модули при решении задач МСС

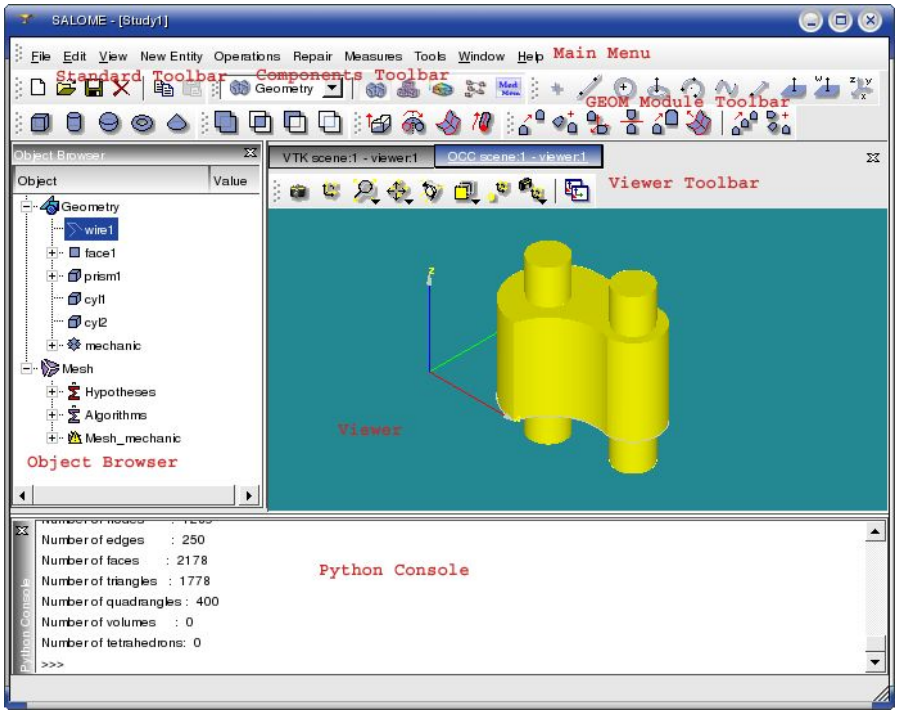


Salome (EDF, CEA, OpenCASCADE)

- **Salome** - является бесплатным программным обеспечением, которое предоставляет платформу для Пре и Пост-обработки числового моделирования.

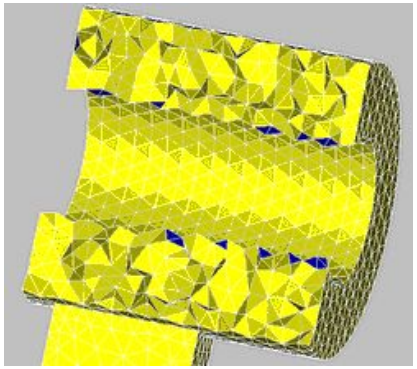
- Основано на открытой и гибкой архитектуре, сделанной из компонентов многократного использования.

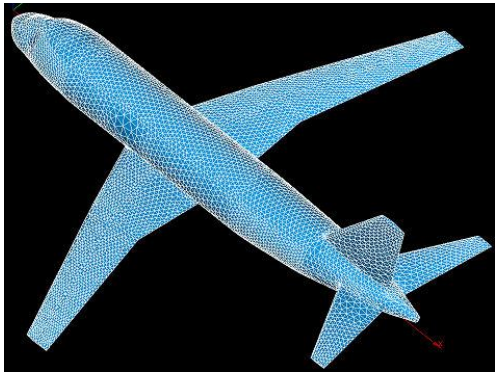
- **Salome** - CAD/CAE интегрированная платформа. С помощью программы возможно:
- Трехмерное моделирование;
- Визуализация;
- Управление вычислительными схемами;
- Постобработка



- **Salome** разработана для интеграции отдельных компонентов:

- Интерфейсы автоматизированного проектирования;
- Генераторы ячеек сетки модели;
- Средства для решения задач с использованием конечных элементов;
- Платформа Salome – это продукт с открытым кодом.





OpenFOAM

OpenFOAM — свободно распространяемое программное обеспечение для проведения численных расчетов.
OpenFOAM — объектно-ориентированная платформа, реализованная на языке программирования C++.

OpenFOAM – перспективное и динамично развивающиеся открытое программное обеспечение для моделирования задач механики сплошных сред. В его разработке и развитии принимают участие десятки организаций и сотни разработчиков по всему миру.

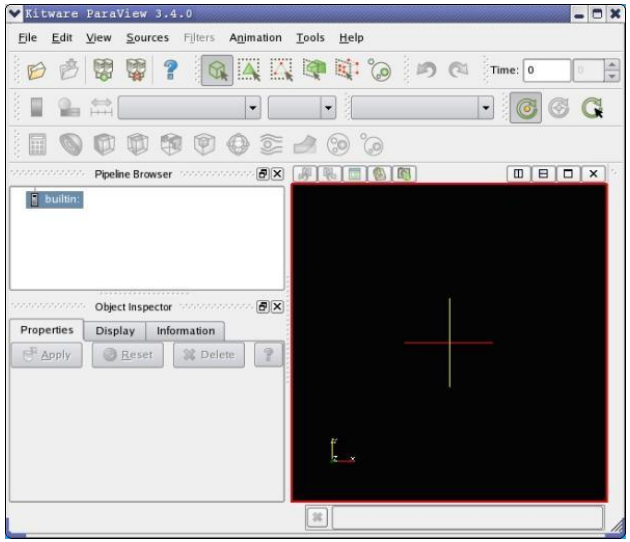
OpenFOAM – обладает большой функциональностью и удовлетворяет всем основным требованиями, предъявляемым к современному программному обеспечению для расчета промышленных задач

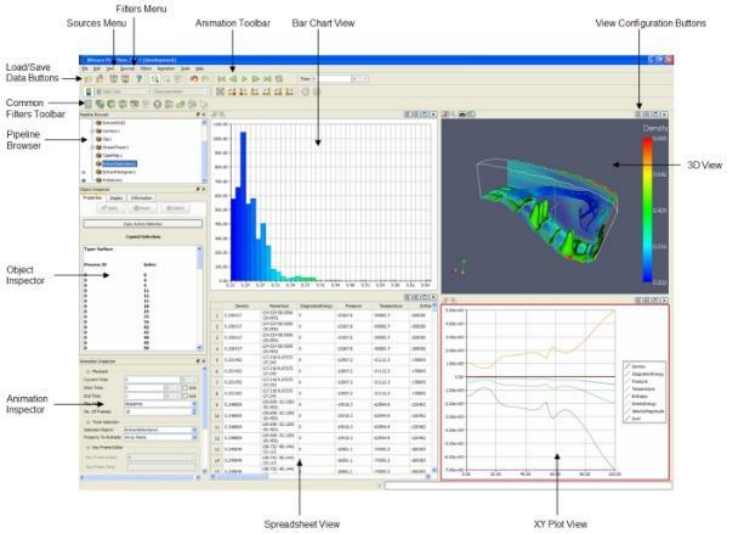
Разработан в Imperial College of Science. London. UK. 1991-2003

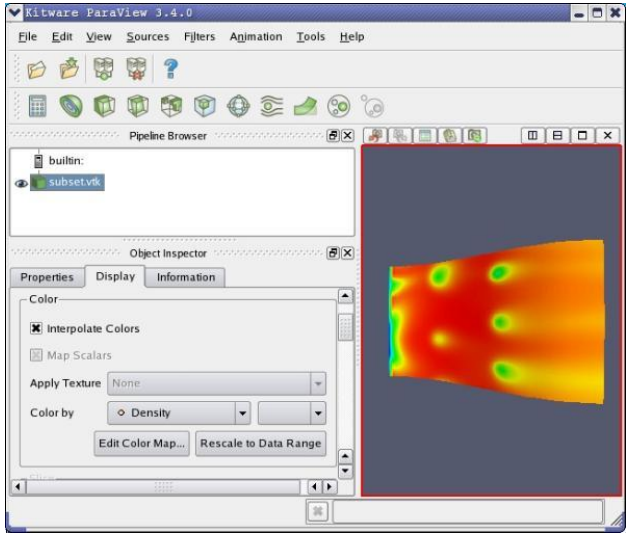
Открытие кода Open Foam в 2004 г. на условиях GPL

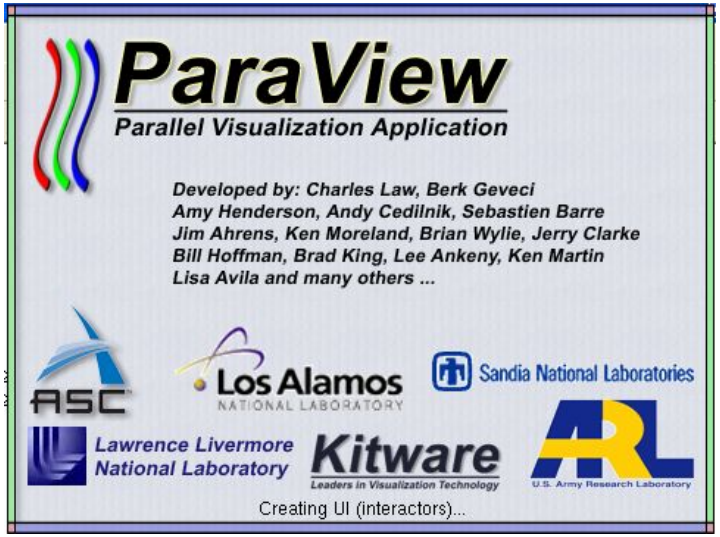
Разработчик OpenCFD. Ltd. UK.
Open Source Conferences - 2007,2008, 2009,2010
Около 3000 пользователей в мире.

Paraview (ARL,ASC, Los Alamos NL, Kitware, Sandia NL,Kitware)







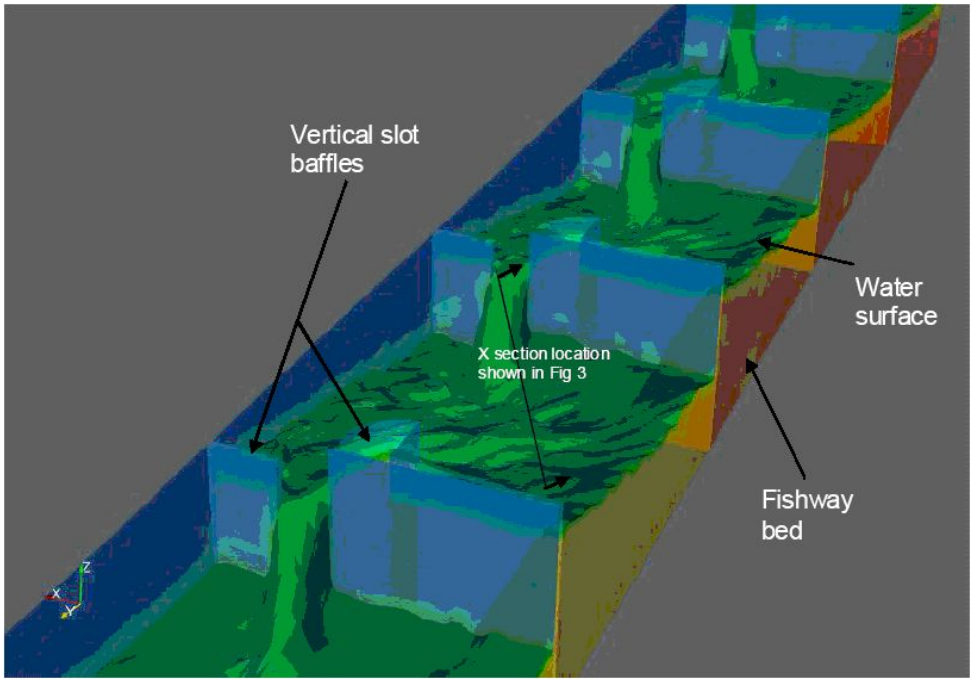


- Multi-view support
- Quantitative analysis
- Undo/redo Python scripting
- Time support
- Plot styles
- Plugins
- Model/View
- Representation/
- Display:
- Stream Lines/Vector Fields
- Contours/iso-surfaces
- XY Plots
- Animation
- Parallel processing
- Documentation

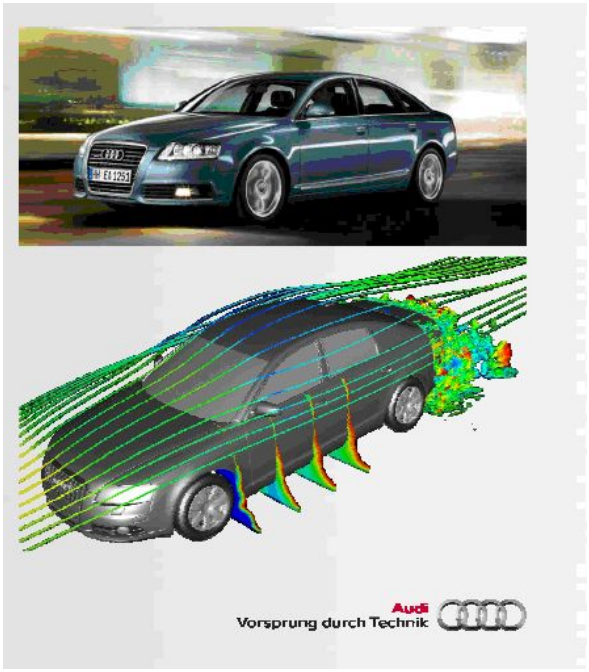
Salome, OpenFoam, Paraview в промышленности, крупных научных центрах и Университетах

- Audi, Volkswagen, Volvo, Seat, ABB Corporate Research, Airbus, BAE Systems, Caldrerys SA, Esteco, Mitsubishi, Shell Oil, Toyota,, Scania, IREQ Hydro Quebec, National Energy Technology Lab., US Dept. of Energy, NRC Canada, US Navy, Sweedish Energy Agency, CSC (Finland), Ohio Supercomputer Center, BEinGrid

- **Университеты:** MIT, Chalmers University, TU Munchen, Politechnico de Milano, University of A Coruna, FSB University Zagreb, University College Dublin, Universitat Rostock, PennState University. Всего около 200 в мире.
- **Россия:** МГТУ им. Н.Э.Баумана, ЮУрГУ, РНЦ КИ, ИБРАЭ РАН, ЭНИМЦ МС, ИАТЭ (Обнинск)



Дамба. H=80 м. 200 Австралия. 2009. Сетка 200 блоков.



Audi A6. 2009. Сетка 47 М. 128-192 ядер.