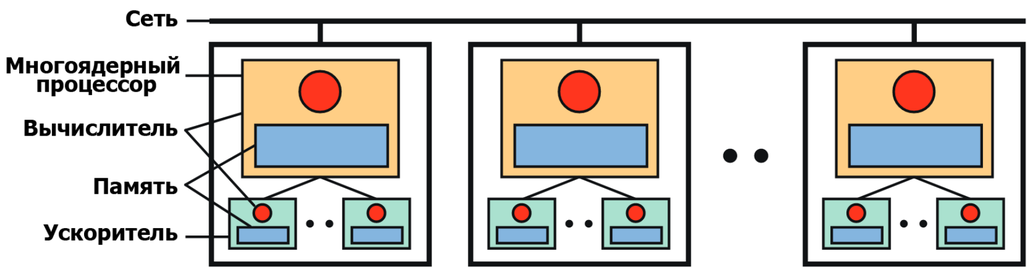
Инкрементальное распараллеливание программ с использованием DVM-системы

В.А. Бахтин[1,2], Д.А. Захаров[1], В.А. Крюков[1,2], Н.В. Поддерюгина[1], М.Н. Притула[1]
ИПМ им. М.В. Келдыша РАН[1], МГУ им. М.В. Ломоносова[2]

DVM-система, созданная в ИПМ им. М. В. Келдыша РАН при активном участии аспирантов и студентов факультета ВМК МГУ им. М. В. Ломоносова, предназначена для разработки параллельных программ научно-технических расчетов на языках C-DVMH и Fortran-DVMH. Эти языки используют единую модель параллельного программирования (DVMH-модель) и являются расширением стандартных языков Си и Фортран спецификациями параллелизма, оформленными в виде директив компилятору.

DVMH-модель позволяет создавать эффективные параллельные программы для гетерогенных вычислительных кластеров, в узлах которых в качестве вычислительных устройств (вычислителей) наряду с универсальными многоядерными процессорами могут использоваться ускорители различной архитектуры (графические процессоры, сопроцессоры Intel Xeon Phi, ...).

При этом отображенные на узел **вычисления автоматически распределяются между вычислителями узла с учетом их производительности**.



DVM-система состоит из следующих компонент:

- **Компиляторы с языков Fortran-DVMH и C-DVMH**

Выполняют преобразование входной программы в параллельную программу, использующую стандартные технологии программирования MPI, OpenMP и CUDA.

- **Библиотека поддержки выполнения DVMH-программ (Lib-DVMH)**

Реализует модель выполнения параллельной программы. Осуществляет динамическую настройку DVMH-программ на выделенные для их выполнения ресурсы: количество узлов кластера, ядер, ускорителей и их производительность.

- **DVMH-отладчик**

Обеспечивает автоматизированную отладку параллельных программ с использованием динамического контроля корректности и сравнительной отладки.

- **Анализатор производительности DVMH-программ**

Предоставляет информацию об основных характеристиках эффективности выполнения параллельной программы и ее частей на вычислительной системе.

Распараллеливание программы в DVMH-модели можно разделить на следующие этапы:

1) Распределение данных (массивов) и вычислений (параллельных циклов).
2) Определение и спецификация удаленных данных (данные, которые вычисляются на одном процессоре, а используются на других).
3) Определение вычислительных регионов (последовательности операторов, циклов) для выполнения на ускорителях.
4) Управление перемещением данных между памятью ЦПУ и памятью ускорителей.

Основная сложность разработки параллельной программы для кластера - **необходимость принятия глобальных решений по распределению данных и вычислений с учетом свойств всей программы**, а затем выполнения кропотливой работы по модификации программы и ее отладке. Большой объем программного кода, многомодульность, многофункциональность затрудняет принятие решений по согласованному распределению данных и вычислений.

Для решения данной проблемы может использоваться **метод инкрементального**, или частичного распараллеливания. Идея этого метода заключается в том, что распараллеливанию подвергается не вся программа целиком, а ее части (области распараллеливания) - в них заводятся дополнительные экземпляры требуемых данных, производится распределение этих данных и соответствующих вычислений. Данные области могут быть построены на основе времен, полученных с помощью анализатора производительности, который входит в состав DVM-системы.

Для взаимодействия с теми частями программы, которые не подвергались распараллеливанию, используются операции копирования исходных (нераспределенных) данных в дополнительные (распределенные) данные и обратно. Конечно, операции копирования могут снизить или вообще ликвидировать эффект от распараллеливания. Кроме того, до полного распараллеливания (пока не все массивы программы будут распределены) программе будет требоваться память и для распределенных, и для нераспределенных массивов, что может ограничивать размер решаемой задачи.

К **достоинствам инкрементального распараллеливания** можно отнести:

1) Возможность распараллелить не всю программу, а ее времяемкие фрагменты, упрощает работу программиста, так как существенно сокращается объем кода программы для анализа и распараллеливания.
2) Отказ от распараллеливания сложных фрагментов программы позволяет с большей вероятностью найти хорошие решения для выделенных областей распараллеливания.
3) Найденные решения могут быть использованы в качестве подсказки при распараллеливании других частей программы на следующих этапах.
4) Использование DVMH-модели позволяет быстро проверить эффективность различных вариантов распределения данных, т.к. переход от одного варианта распределения данных на другой, как правило, осуществляется за счет изменения нескольких DVM-указаний и не требует сложной модификации кода программы.

Рассмотрим процесс инкрементального распараллеливания программы на языке Fortran для решения уравнения теплопроводности методом Якоби с использованием DVM-системы.

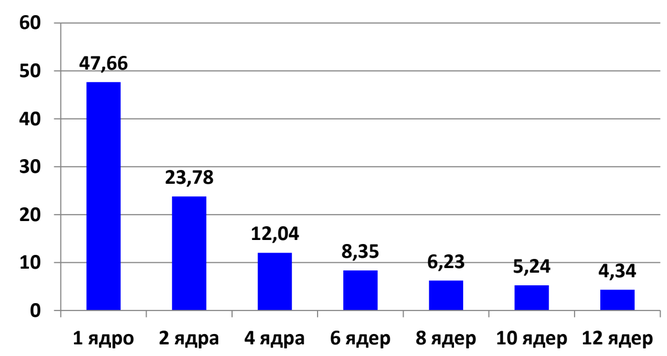
Этап 1. Анализ последовательной программы. Определение времяемких фрагментов программы

```
program jacobi
    double precision, allocatable, dimension(:,:,:) :: f, newf, r
    …
    allocate(f(mx, my, mz))
    allocate(newf(mx, my, mz))
    allocate(r(mx, my, mz))
    curf = 0
    do n = 1, NITER
      if (curf .eq. 0) then
        eps = dostep(f, newf, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
      else
        eps = dostep(newf, f, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
      endif
      print *, 'Iteration=' , n, 'eps=', eps
      curf = 1 - curf
    enddo
end
```

```
double precision function dostep(f, newf, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
    integer :: mx, my, mz, i, j, k
    double precision, dimension(mx,my,mz) :: f, newf, r
    double precision :: rdx2, rdy2, rdz2, beta, eps
    eps = 0.
    do k = 2, mz - 1
      do j = 2, my – 1
        do i = 2, mx - 1
          newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2 +
&                 (f(i,j-1,k)+f(i,j+1,k))*rdy2 + (f(i,j,k-1)+f(i,j,k+1))*rdz2
&                 -r(i,j,k)) * beta
          eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
        enddo
      enddo
    enddo
    dostep = eps
end function
```

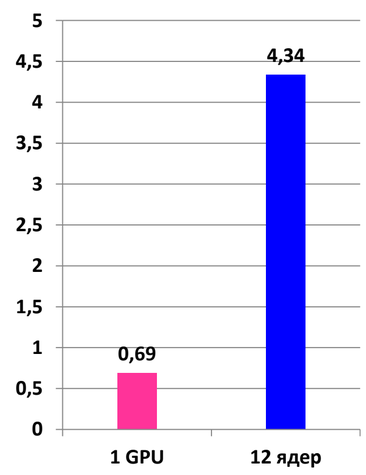
Этап 2. Распределение вычислений. Получение программы для мультипроцессора

```
double precision function dostep(f, newf, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
    integer :: mx, my, mz, i, j, k
    double precision, dimension(mx,my,mz) :: f, newf, r
    double precision :: rdx2, rdy2, rdz2, beta, eps
    eps = 0.
!DVM$ PARALLEL (k,j,i), REDUCTION(max(eps))
    do k = 2, mz - 1
      do j = 2, my – 1
        do i = 2, mx - 1
          newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2 +
&                 (f(i,j-1,k)+f(i,j+1,k))*rdy2 + (f(i,j,k-1)+f(i,j,k+1))*rdz2
&                 -r(i,j,k)) * beta
          eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
        enddo
      enddo
    enddo
    dostep = eps
end function
```



Этап 3. Определение вычислительных регионов. Получение программы для ускорителя

```
double precision function dostep(f, newf, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
    integer :: mx, my, mz, i, j, k
    double precision, dimension(mx,my,mz) :: f, newf, r
    double precision :: rdx2, rdy2, rdz2, beta, eps
    eps = 0.
!DVM$ ACTUAL(eps)
!DVM$ REGION INOUT(f,newf, eps), IN(r,rdx2,rdy2,rdz2,beta)
!DVM$ PARALLEL (k,j,i), REDUCTION(max(eps))
    do k = 2, mz - 1
      do j = 2, my – 1
        do i = 2, mx - 1
          newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2 +
&                 (f(i,j-1,k)+f(i,j+1,k))*rdy2 + (f(i,j,k-1)+f(i,j,k+1))*rdz2
&                 -r(i,j,k)) * beta
          eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
        enddo
      enddo
    enddo
!DVM$ ENDREGION
!DVM$ GET_ACTUAL(eps)
    dostep = eps
end function
```

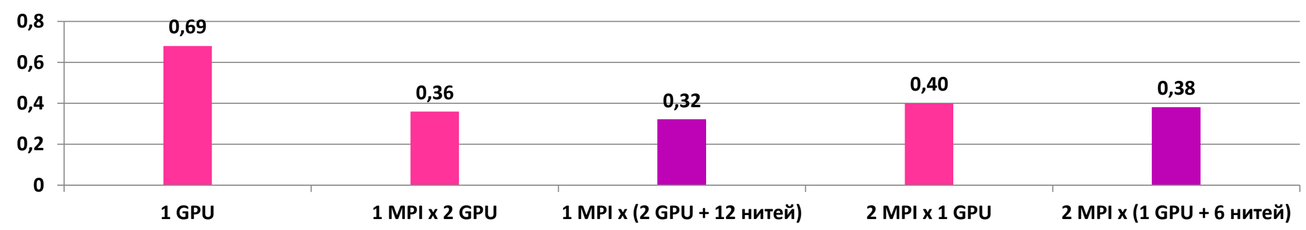


Этап 4. Распределение данных, определение удаленных данных. Получение программы для кластера с ускорителями

```
program jacobi
    double precision, allocatable, dimension(:,:,:) :: f, newf, r
!DVM$ DISTRIBUTE (BLOCK, BLOCK,BLOCK) :: f
!DVM$ ALIGN newf(i,j,k) WITH f(i,j,k)
!DVM$ ALIGN r(i,j,k) WITH f(i,j,k)
    …
    allocate(f(mx, my, mz))
    allocate(newf(mx, my, mz))
    allocate(r(mx, my, mz))
    curf = 0
    do n = 1, NITER
      if (curf .eq. 0) then
        eps = dostep(f, newf, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
      else
        eps = dostep(newf, f, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
      endif
      print *, 'Iteration=' , n, 'eps=', eps
      curf = 1 - curf
    enddo
end
```

```
double precision function dostep(f, newf, r, rdx2, rdy2, rdz2, beta, mx, my, mz)
    integer :: mx, my, mz, i, j, k
    double precision, dimension(mx,my,mz) :: f, newf, r
 !DVM$ INHERIT f,newf,r
    double precision :: rdx2, rdy2, rdz2, beta, eps
    eps = 0.
!DVM$ ACTUAL(eps)
!DVM$ REGION
!DVM$ PARALLEL(k,j,i) ON newf(i,j,k),REDUCTION(max(eps)),
!DVM$*  SHADOW_RENEW(f)
    do k = 2, mz - 1
      do j = 2, my – 1
        do i = 2, mx - 1
          newf(i, j, k) = ((f(i-1,j,k)+f(i+1,j,k))*rdx2 +
&                 (f(i,j-1,k)+f(i,j+1,k))*rdy2 + (f(i,j,k-1)+f(i,j,k+1))*rdz2
&                 -r(i,j,k)) * beta
          eps = max(eps,abs(newf(i,j,k)-f(i,j,k)))
        enddo
      enddo
    enddo
!DVM$ ENDREGION
!DVM$ GET_ACTUAL(eps)
    dostep = eps
end function
```

Этап 5. Запуск полученной программы в различных режимах на кластере с ускорителями



На рисунках показаны времена выполнения 10 итераций алгоритма Якоби (в секундах) для массивов размером 512x512x512 элементов на гибридном вычислительном комплексе К-10 (ИПМ РАН) с процессорами Intel Xeon E5-2660 и графическими ускорителями nVidia Fermi M2090. На ядрах центрального процессора, на графическом ускорителе, при одновременном использовании ядер центрального процессора и нескольких графических ускорителей.

С использованием инкрементального подхода в DVMH-модели разработано множество параллельных программ:

- «Перколяция» (моделирование динамических процессов фильтрации в перколяционных решетках);
- «Композит» (моделирование многокомпонентной многофазной изотермической фильтрации при разработке месторождений нефти и газа);
- пакет прикладных программ «РЕАКТОР» (нейтронно-физический расчет ядерных реакторов и гибридных ядерных установок);
- Elasticity3D (численное моделирование 3D сейсмических полей в упругих средах для областей со сложной геометрией свободной поверхности);
- «Каверна» (моделирование циркуляционного течения в плоской квадратной каверне);
- и другие.

Детали распараллеливания этих и многих других задач можно найти на сайте проекта: http://dvm-system.org