

**Параллельная реализация задачи
моделирования нелинейной
многофазовой фильтрации
в деформируемых пористых средах
на кластере с ускорителями Xeon Phi**

**Киреев С.Е.
ИВМиМГ СО РАН**

Предыстория

- Ранее была разработана модель, которая позволяет описывать нелинейную неизотермическую фильтрацию воды/нефти/водонефтяной смеси сквозь упруго-деформируемую пористую среду.
- Созданный на её основе программный комплекс *Porodynamics* предназначен для исполнения на кластере с распределенной памятью и графическими ускорителями Nvidia Tesla.

Государственный контракт № 07.514.11.4156 «Поисковые проблемно-ориентированные исследования в области математического моделирования задач многофазной фильтрации в деформируемых пористых средах на вычислительных системах сверхвысокой производительности».

E.I. Romenskiy, Y.V. Perepechko, G.V. Reshetova Modeling of Multiphase Flow in Elastic Porous Media Based on Thermodynamically Compatible Systems Theory // Poster presentation at ECMOR XIV - 14th European conference on the mathematics of oil recovery

Постановка задачи

Цель данной работы:

- оптимизация программного комплекса *Porodynamics* для использования кластера с ускорителями Intel Xeon Phi
- Оценка эффективности использования ускорителей Intel Xeon Phi для решения этой задачи

Модель

Система уравнений

$$\frac{\partial u_i^1}{\partial t} + \frac{K_1}{\rho_1^0} \frac{\partial \rho_1}{\partial x_i} + \frac{\pi_1}{\rho_1^0} \frac{\partial s}{\partial x_i} - \alpha_1^0 \frac{2\mu}{\rho^0} \frac{\partial \varepsilon_{ik}}{\partial x_k} + \alpha_1^0 \frac{2\mu}{3\rho^0} \frac{\partial(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33})}{\partial x_i} = -c_2^0 \chi (u_i^1 - u_i^2)$$

$$\frac{\partial u_i^2}{\partial t} + \frac{K_2}{\rho_2^0} \frac{\partial \rho_2}{\partial x_i} + \frac{\pi_2}{\rho_2^0} \frac{\partial s}{\partial x_i} - \alpha_1^0 \frac{2\mu}{\rho^0} \frac{\partial \varepsilon_{ik}}{\partial x_k} + \alpha_1^0 \frac{2\mu}{3\rho^0} \frac{\partial(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33})}{\partial x_i} = c_1^0 \chi (u_i^1 - u_i^2)$$

$$\frac{\partial \rho_1}{\partial t} + \rho_1^0 \frac{\partial u_k^1}{\partial x_k} = \lambda \frac{\rho_1^0}{\alpha_1^0 \rho^0} (p_2 - p_1)$$

$$\frac{\partial \rho_2}{\partial t} + \rho_2^0 \frac{\partial u_k^2}{\partial x_k} = -\lambda \frac{\rho_2^0}{\alpha_1^0 \rho^0} (p_2 - p_1)$$

$$\frac{\partial \varepsilon_{ij}}{\partial t} - \frac{c_1^0}{2} \left(\frac{\partial u_i^1}{\partial x_j} + \frac{\partial u_j^1}{\partial x_i} \right) - \frac{c_2^0}{2} \left(\frac{\partial u_i^2}{\partial x_j} + \frac{\partial u_j^2}{\partial x_i} \right) = 0$$

$$\frac{\partial \alpha_1}{\partial t} = \lambda \frac{p_1 - p_2}{\rho^0}$$

$$\frac{\partial s}{\partial t} = 0$$

Модель двумерная

- u_i^1, u_i^2 ($i = 1, 2, 3$) – скорости твёрдой и жидкой фаз
 ρ_1, ρ_2 – плотности твёрдой и жидкой фаз
 ε_{ij} – тензор деформаций элемента среды
 s – энтропия смеси
 α_1^0 – фоновое значение объёмной концентрации твёрдой фазы
 ρ_1^0, ρ_2^0 – фоновые значения плотностей твёрдой и жидкой фаз
 $c_1^0 = \frac{\alpha_1^0 \rho_1^0}{\rho}, c_2^0 = \frac{\alpha_2^0 \rho_2^0}{\rho}, \rho^0 = \alpha_1^0 \rho_1^0 + \alpha_2^0 \rho_2^0$
 K_1, K_2 – отношения модулей объёмного сжатия к плотностям твёрдой и жидкой фаз
 π_1, π_2 – константы фаз, связанные с тепловым расширением
 μ – модуль сдвига упругой среды
 χ – коэффициент межфазного трения
 λ – коэффициент, характеризующий скорость релаксации давлений фаз к равновесному состоянию

$$p_1 = K_1 \rho_1, p_2 = K_2 \rho_2$$

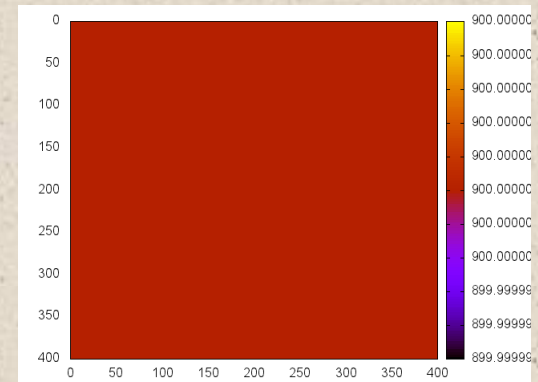
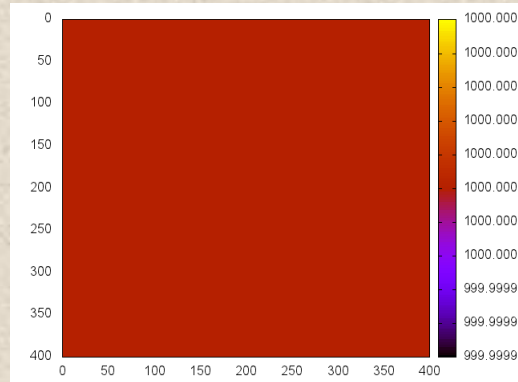
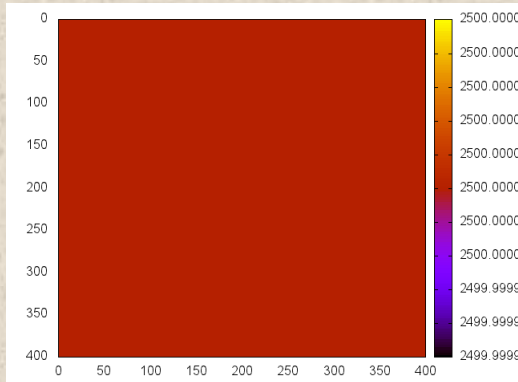
Модель

Система уравнений переписывается в векторной форме:

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} + \frac{\partial G(U)}{\partial y} = S(U)$$

где U – вектор консервативных переменных, зависящих от примитивных.

Новая система уравнений решается методом WENO-Рунге-Кутты 5-го порядка точности.



Алгоритм

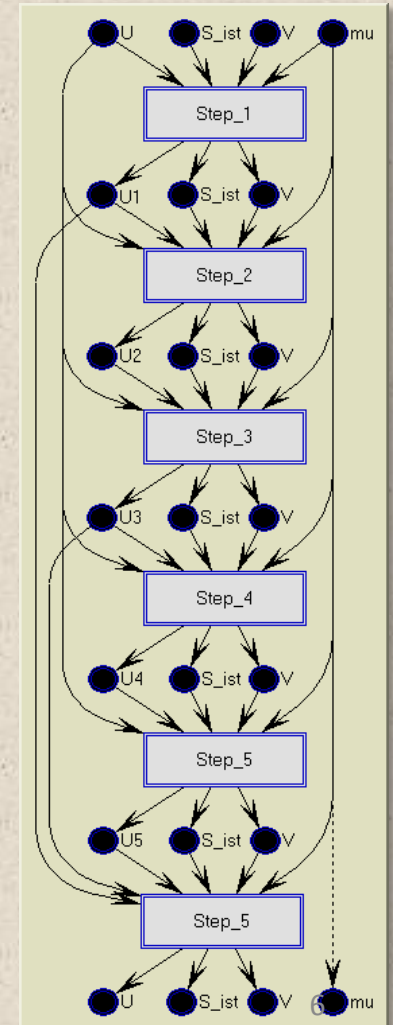
Схема используемого метода Рунге-Кутты ($U^n \rightarrow U^{n+1}$):

- $U_1 = U^n + \frac{1}{2} \tau L(U^n)$
- $U_2 = U_1 + \frac{1}{2} \tau L(U_1)$
- $U_3 = U_2 + \frac{1}{2} \tau L(U_2)$
- $U_4 = U_3 + \frac{1}{2} \tau L(U_3)$
- $U_5 = U_4 + \frac{1}{2} \tau L(U_4)$
- $U^{n+1} = \frac{1}{9} \cdot U + \frac{2}{5} \cdot U_1 + \frac{4}{9} \cdot U_3 + \frac{2}{45} \cdot U_5 + \frac{1}{45} \cdot \tau \cdot L(U_5)$

Вычисление оператора $L(U)$:

- $L(U) = S(U) - \delta F(U)/\delta x - \delta G(U)/\delta y$


- Используются только явные схемы

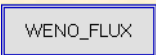


Алгоритм

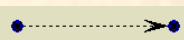
Шаг метода Рунге-Кутты –
представление в виде графа

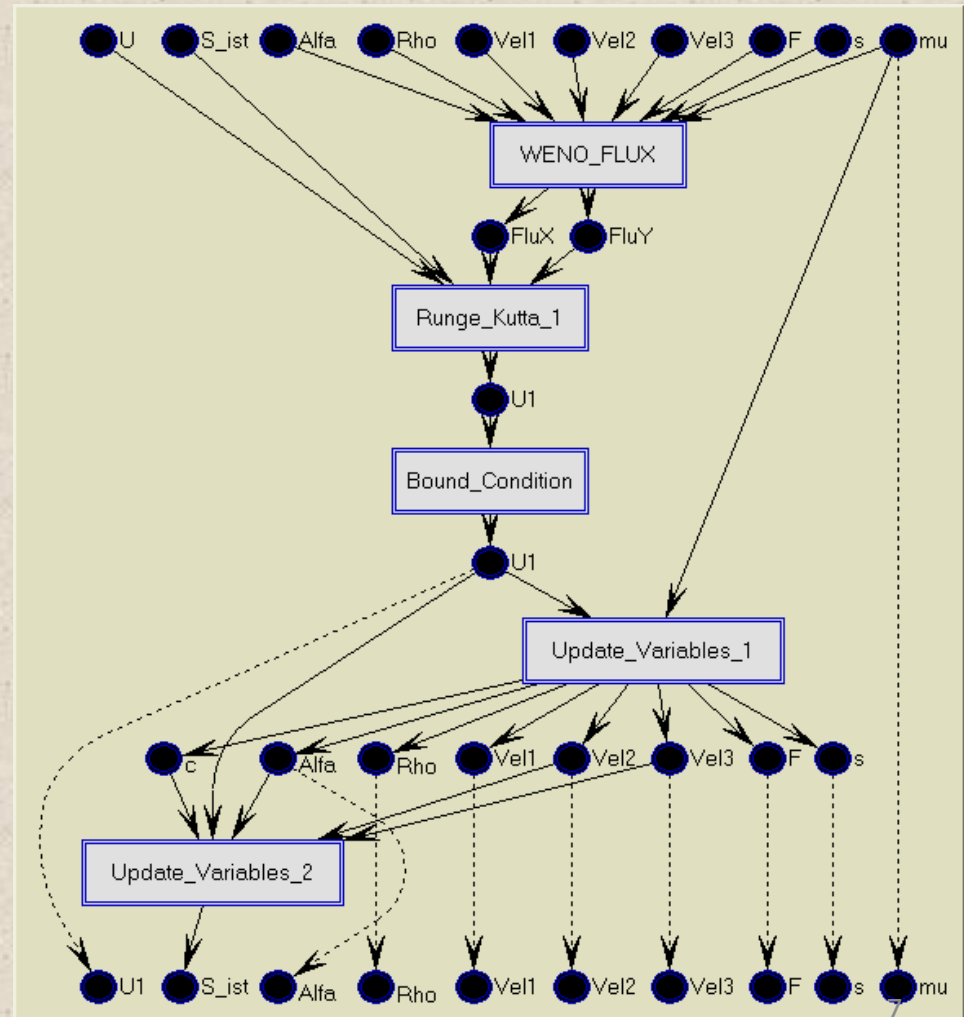
Обозначения:

 – переменные
единственного
присваивания (2D массивы)

 – операции однократного
срабатывания (гнезда
циклов – обход 2D массива)
`for (iy=ny1; iy<=ny2; iy++)
for (ix=nx1; ix<=nx2; ix++) { ... }`

 – зависимость по данным

 – «идентичность»
переменных



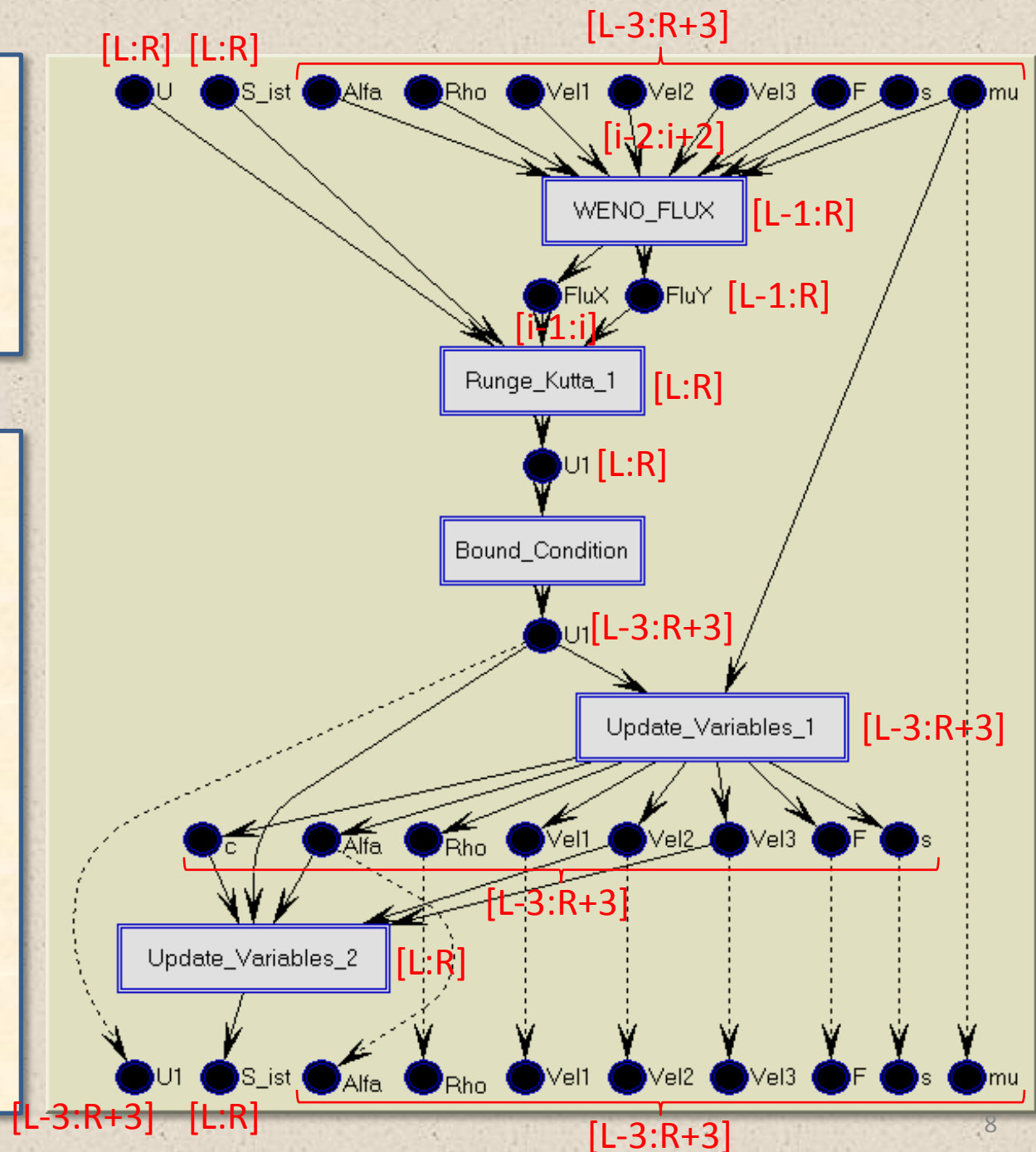
Аннотированный граф

- Границы массивов,
- Границы циклов
- Шаблоны обращений к элементам массива

Зачем он нужен?

Для анализа алгоритма:

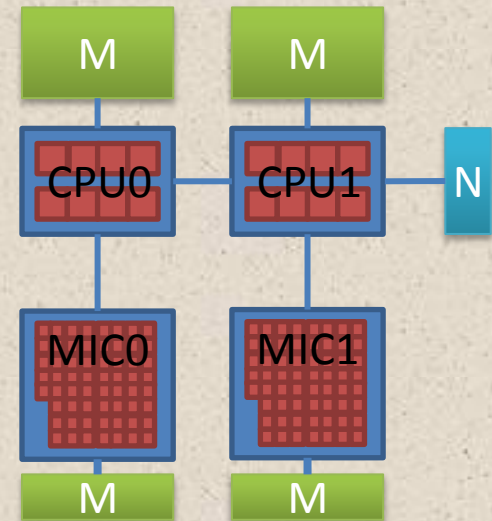
- Проверка корректности алгоритма
- Поиск независимых операций для распараллеливания
- Поиск места для обмена границами при декомпозиции области
- Поиск места для создания checkpoint



Вычислительная система

Кластер МВС-10П (МСЦ СО РАН)

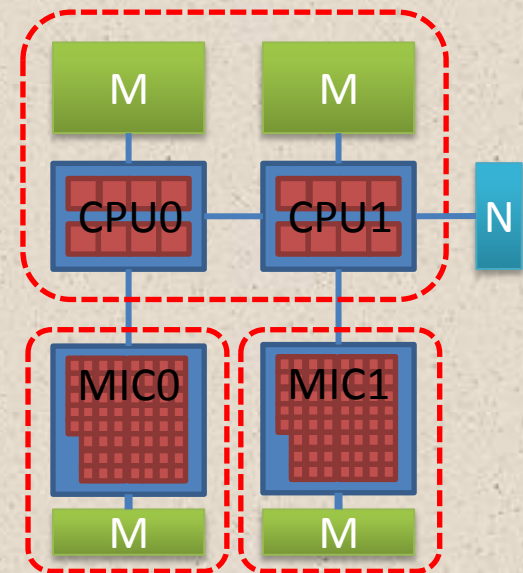
- Узел кластера:
 - хост-система
 - 2 × Xeon E5-2690, 2.9 ГГц
8 ядер × 2 потока – 32 потока
 - 64 ГБ памяти
 - **185 GFLOPS** пиковая произв-ть
 - два ускорителя Intel MIC
 - Xeon Phi 7110X, 1094 ГГц
61 ядро × 4 потока – 244 потока
 - 8 Гб памяти
 - **1094.8 GFLOPS** пиковая произв-ть



Вычислительная система

Кластер МВС-10П (МСЦ СО РАН)

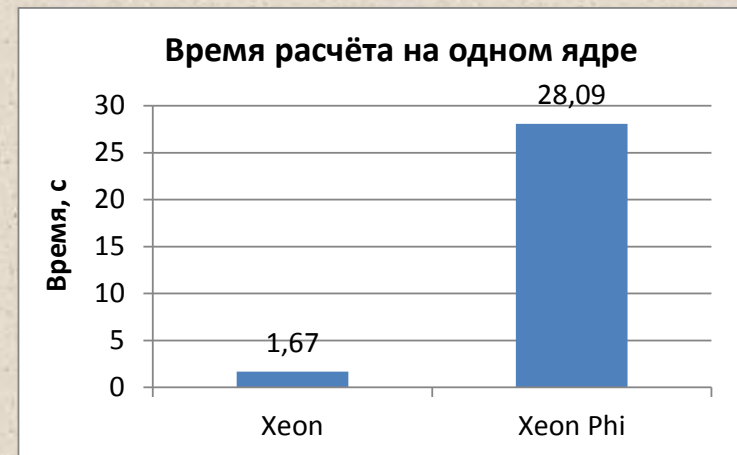
- Узел кластера:
 - хост-система
 - 2 × Xeon E5-2690, 2.9 ГГц
8 ядер × 2 потока – 32 потока
 - 64 ГБ памяти
 - **185 GFLOPS** пиковая произв-ть
 - два ускорителя Intel MIC
 - Xeon Phi 7110X, 1094 ГГц
61 ядро × 4 потока – 244 потока
 - 8 Гб памяти
 - **1094.8 GFLOPS** пиковая произв-ть



Вычислительная система

Кластер МВС-10П (МСЦ СО РАН)

- Узел кластера:
 - хост-система
 - 2 × Xeon E5-2690, 2.9 ГГц
8 ядер × 2 потока – 32 потока
 - 64 ГБ памяти
 - **185 GFLOPS** пиковая произв-ть
 - два ускорителя Intel MIC
 - Xeon Phi 7110X, 1094 ГГц
61 ядро × 4 потока – 244 потока
 - 8 Гб памяти
 - **1094.8 GFLOPS** пиковая произв-ть



исходная программа
Сетка: 100x100, 1 шаг по времени

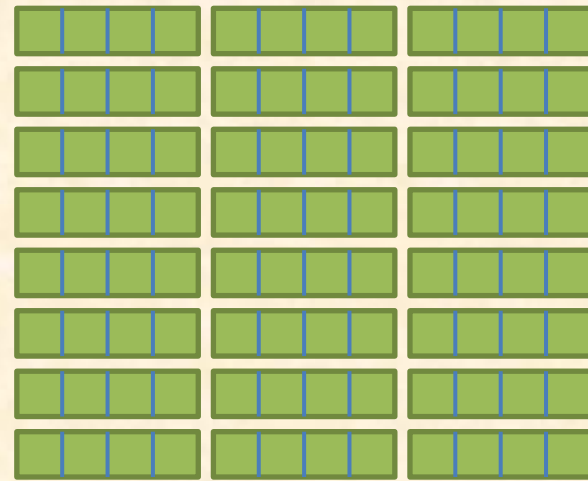
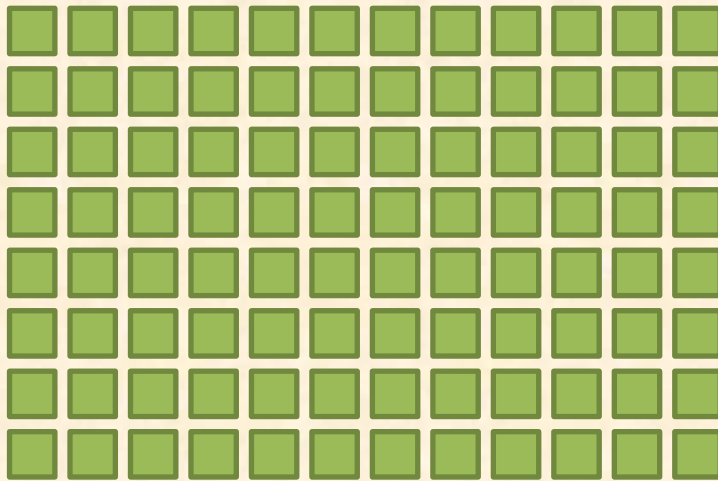
Оптимизация

Способы оптимизации для Xeon Phi

- Векторизация вычислений
 - Выровненный доступ в память
 - Поточковая запись в память
- Распараллеливание по ядрам/потокам
- Совместное использование всех ресурсов узла
 - Минимизация обменов данными
 - Обмены данными на фоне счёта

Оптимизация

Векторизация вычислений



Предпосылки к ускорению

■ ■ ■ ■ - медленнее,

■ ■ ■ ■ - быстрее

Предпосылки к замедлению

преобразование



требует времени

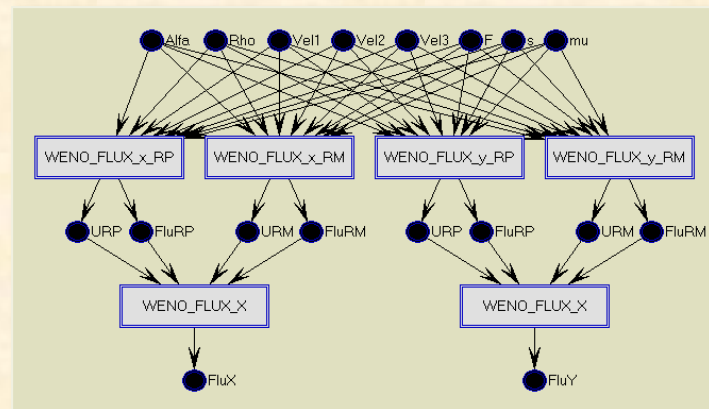
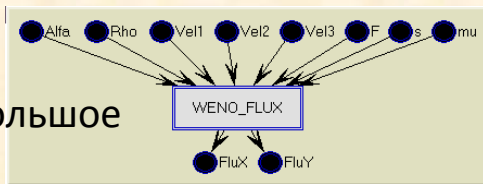
Оптимизация

Векторизация вычислений

- Добавление директив компилятора

```
#pragma ivdep
for (ix=nx1; ix<=nx2; ix++) ... // внутренний цикл по пространству
```
- При необходимости – упрощение кода
 - Подпрограмма Weno_flux
 - Разбиение на несколько гнезд циклов

Слишком большое
тело цикла



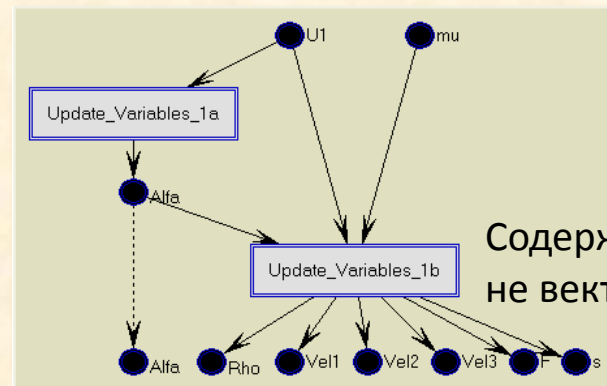
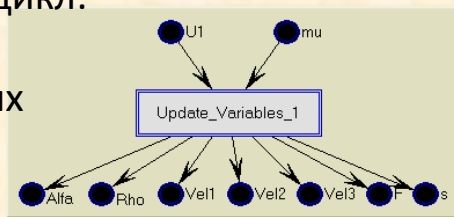
Оптимизация

Векторизация вычислений

- Добавление директив компилятора

```
#pragma ivdep
for (ix=nx1; ix<=nx2; ix++) ... // внутренний цикл по пространству
```
- При необходимости – упрощение кода
 - Подпрограмма Weno_flux
 - Разбиение на несколько гнезд циклов
 - Подпрограмма Update_Variables_1
 - Разбиение на несколько гнезд циклов

Содержит цикл:
решение
нелинейных
уравнений
методом
Ньютона



Содержит цикл –
не векторизуется

Оптимизация

Векторизация вычислений

- Добавление директив компилятора

```
#pragma ivdep
```

```
for (ix=nx1; ix<=nx2; ix++) ... // внутренний цикл по пространству
```

- При необходимости – упрощение кода

- Подпрограмма `Weno_flux`

- Разбиение на несколько гнезд циклов

- Подпрограмма `Update_Variables_1`

- Разбиение на несколько гнезд циклов

- Раскрытие гарантированного числа итераций метода Ньютона в отдельном векторизуемом цикле

```
for (ix=nx1; ix<=nx2; ix++) {  
    ...  
    while(...) { do_step; }  
    ...  
}
```



```
for (ix=nx1; ix<=nx2; ix++) {  
    ...  
    do_step; do_step;  
}  
for (ix=nx1; ix<=nx2; ix++) {  
    while(...) { do_step; }  
    ...  
}
```


Оптимизация

Векторизация вычислений

- Добавление директив компилятора

```
#pragma ivdep
```

```
for (ix=nx1; ix<=nx2; ix++) ... // внутренний цикл по пространству
```

- При необходимости – упрощение кода

- Подпрограмма Weno_flux

- Разбиение на несколько гнезд циклов

- Подпрограмма Update_Variables_1

- Разбиение на несколько гнезд циклов

- Раскрытие гарантированного числа итераций метода Ньютона в отдельном векторизуемом цикле

- Проверка отчёта компилятора обо всех интересующих циклах

- Ускорение: Xeon – на 10%, Xeon Phi – в 2 раза

Оптимизация

Выравнивание обращений к памяти

- Выровненное выделение памяти

```
data = _mm_malloc(totalsize,64);
```

- Выравнивание начала строк массивов

```
nx = align_up_to(nx, 64);
```

- Указание компилятору, что данные выровнены:

```
for (ix=nx1; ix<=nx2; ix++) // внутренний цикл по пространству  
{ __assume_aligned(Rho1,64);  
  ...  
}
```

- Проверка отчёта компилятора обо всех интересующих циклах и массивах

Оптимизация

Выравнивание обращений к памяти

- Выровненное выделение памяти

```
data = _mm_malloc(totalsize,64);
```

- Выравнивание начала строк массивов

```
nx = align_up_to(nx, 64);
```

- Указание компилятору, что данные выровнены:

```
for (ix=nx1; ix<=nx2; ix++) // внутренний цикл по пространству  
{ __assume_aligned(Rho1,64);  
  ...  
}
```

- Проверка отчёта компилятора обо всех интересующих циклах и массивах
- Ускорение: Xeon – на 0.2%, Xeon Phi – на 1%

Оптимизация

Потоковая запись в память

```
#pragma vector nontemporal  
for (ix=nx1; ix<=nx2; ix++) ... // внутренний цикл по пространству  
...
```

- Результат
 - Xeon Phi – ускорение на 1%
 - Xeon – замедление на 30%

Решение: не делать потоковую запись

Оптимизация

- **Было: 62 операции (15 делений)**

- double tmp1 = Teta1/((1.0-xx-yy) * Rho_10), tmp1__2 = tmp1 * tmp1, tmp1__3 = tmp1__2 * tmp1;
- double tmp2 = Teta2/(xx * Rho_20), tmp2__2 = tmp2 * tmp2, tmp2__3 = tmp2__2 * tmp2;
- double tmp3 = Teta3/(yy * Rho_30), tmp3__2 = tmp3 * tmp3, tmp3__3 = tmp3__2 * tmp3;
- double t1 = (P_10 + pi1 * delta_s + Kbig1 * (Teta1 / (1.0-xx-yy) - Rho_10) / Rho_10) * tmp1__2;
- double t2 = (P_20 + pi2 * delta_s + Kbig2 * (Teta2 / xx - Rho_20) / Rho_20) * tmp2__2;
- double t3 = (P_30 + pi3 * delta_s + Kbig3 * (Teta3 / yy - Rho_30) / Rho_30) * tmp3__2;
- f = t3 - t2;
- g = t3 - t1;
- fx = 2.0 * t2 / xx + tmp2__3 * Kbig2 / xx;
- fy = - 2.0 * t3 / yy - tmp3__3 * Kbig3 / yy;
- gx = - 2.0 * t1 / (1.0-xx-yy) - tmp1__3 * Kbig1 / (1.0-xx-yy);
- gy = fy + gx;

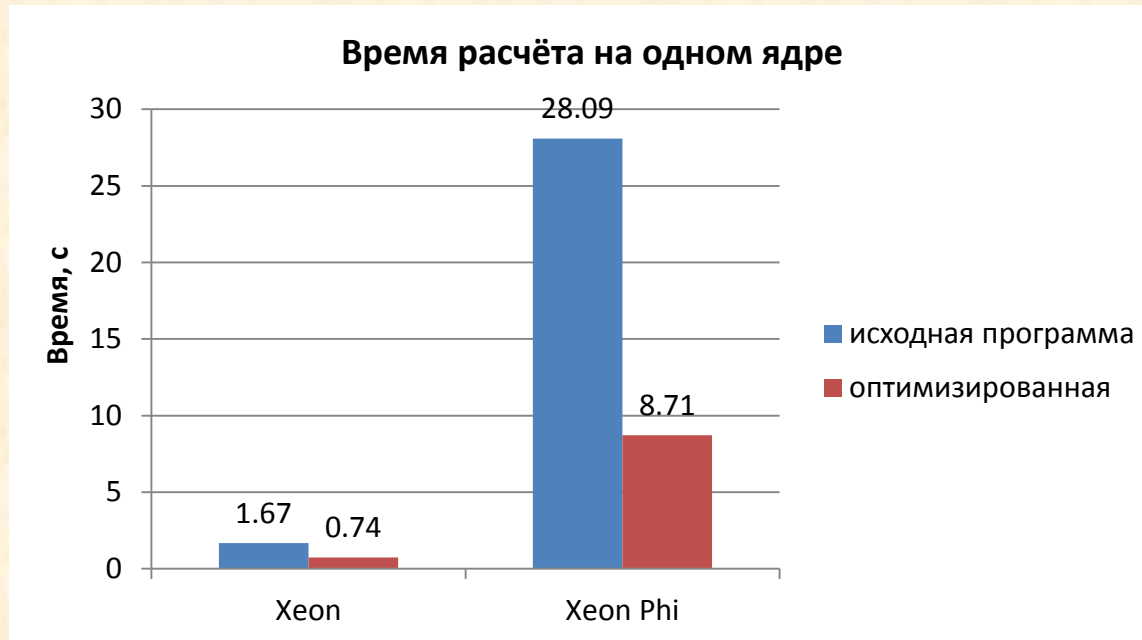
- **Стало: 47 операций (3 деления)**

- double rxx = 1.0 / xx;
- double ryy = 1.0 / yy;
- double rxyy = 1.0 / (1.0 - xx - yy);
- double tmp1 = Teta1_Rho_10 * rxyy, tmp1__2 = tmp1 * tmp1, tmp1__3 = tmp1__2 * tmp1;
- double tmp2 = Teta2_Rho_20 * rxx, tmp2__2 = tmp2 * tmp2, tmp2__3 = tmp2__2 * tmp2;
- double tmp3 = Teta3_Rho_30 * ryy, tmp3__2 = tmp3 * tmp3, tmp3__3 = tmp3__2 * tmp3;
- double t1 = (P_10 + pi1 * delta_s + Kbig1 * (tmp1 - 1.0)) * tmp1__2;
- double t2 = (P_20 + pi2 * delta_s + Kbig2 * (tmp2 - 1.0)) * tmp2__2;
- double t3 = (P_30 + pi3 * delta_s + Kbig3 * (tmp3 - 1.0)) * tmp3__2;
- f = t3 - t2;
- g = t3 - t1;
- fx = (2.0 * t2 + tmp2__3 * Kbig2) * rxx;
- fy = (- 2.0 * t3 - tmp3__3 * Kbig3) * ryy;
- gx = (- 2.0 * t1 - tmp1__3 * Kbig1) * rxyy;
- gy = fy + gx;

- **Ускорение на Хеон и Хеон Phi – более 2 раз**

Оптимизация

Результат оптимизации последовательной программы:

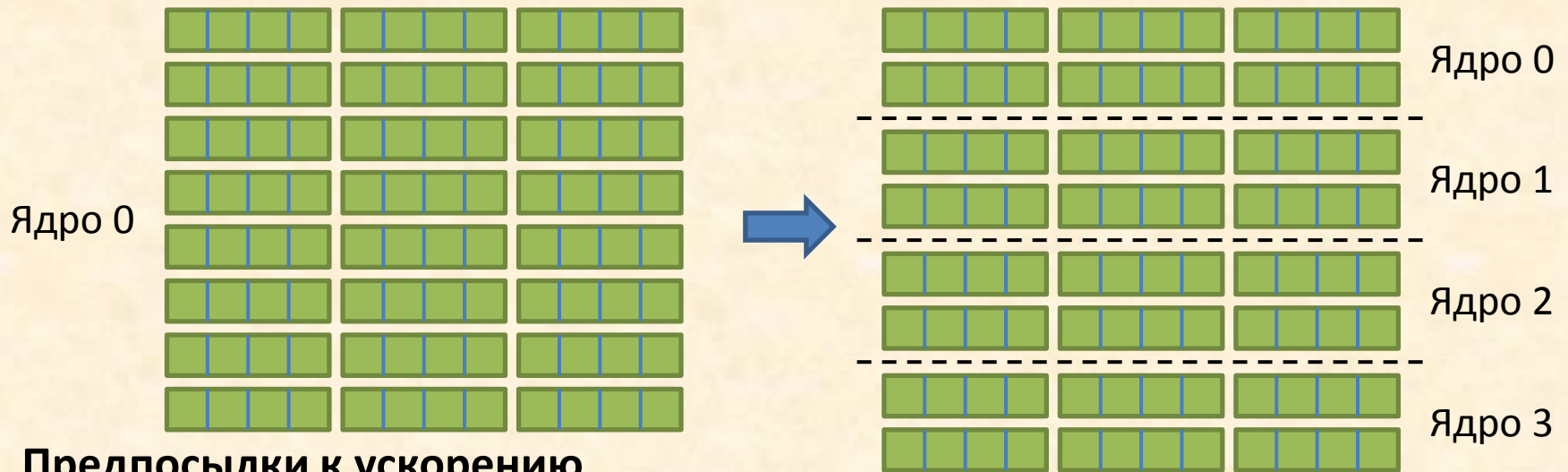


Xeon: ускорение в 2.3 раза

Xeon Phi: ускорение в 3.2 раза

Распараллеливание

Распараллеливание по ядрам



Предпосылки к ускорению

- Несколько ядер выполнят работу быстрее

Предпосылки к замедлению

- При использовании OpenMP компилятор иногда не может применить некоторые оптимизации к многопоточному коду (выровненный доступ к памяти)
- Накладные расходы на организацию многопоточности

Распараллеливание

Распараллеливание по ядрам

- Использование OpenMP

```
#pragma omp parallel for
```

```
for (iy=ny1; iy<=ny2; iy++) ... // внешний цикл по пространству
```

или

```
#pragma omp parallel for collapse(2)
```

```
for (equ=0; equ<16; equ++) // цикл по уравнениям
```

```
for (iy=ny1; iy<=ny2; iy++) ... // внешний цикл по пространству
```

- Результат:

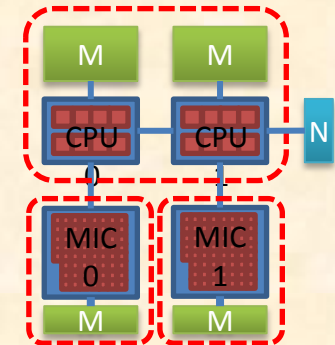


Распараллеливание

Использование всех ресурсов узла

Режимы использования:

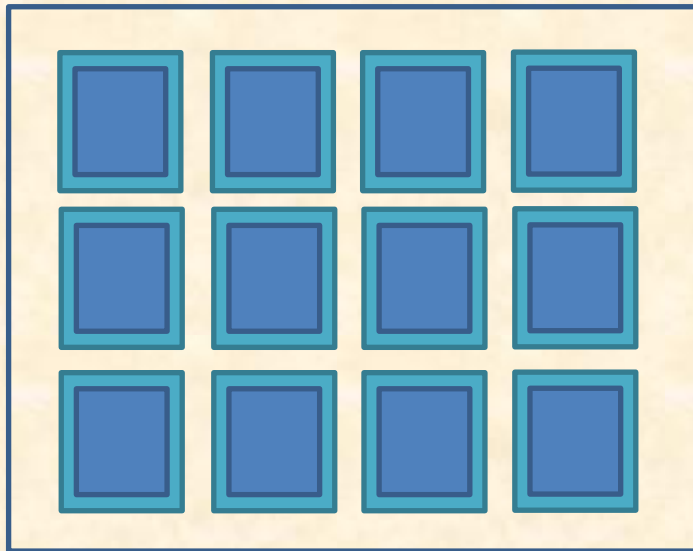
- *Симметричный* – размещение процессов одной MPI-программы на ядрах хост-процессоров и сопроцессоров
 - MPI + OpenMP
- *Offload* – выполнение на сопроцессорах узла помеченных частей основной программы, работающей на хост-процессорах
 - Offload + OpenMP



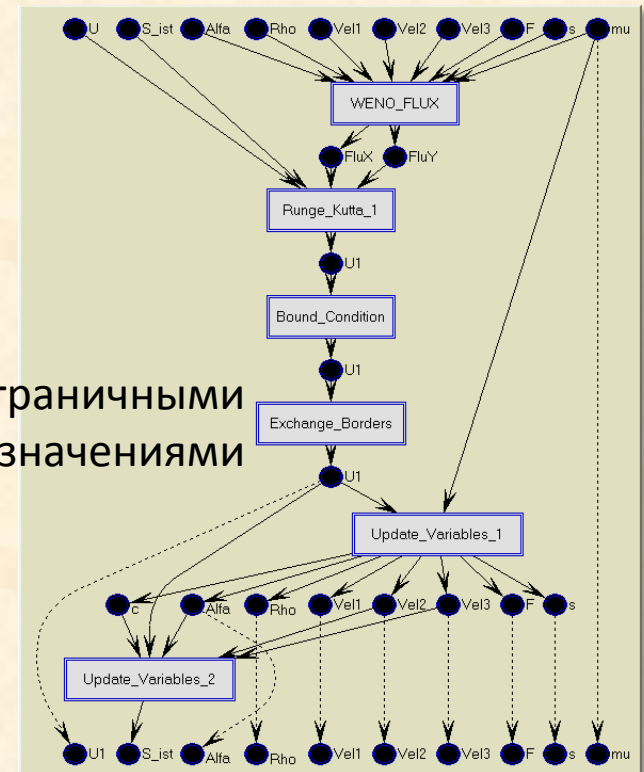
Распараллеливание

Распараллеливание в MPI

- 2D декомпозиция области моделирования на равные подобласти



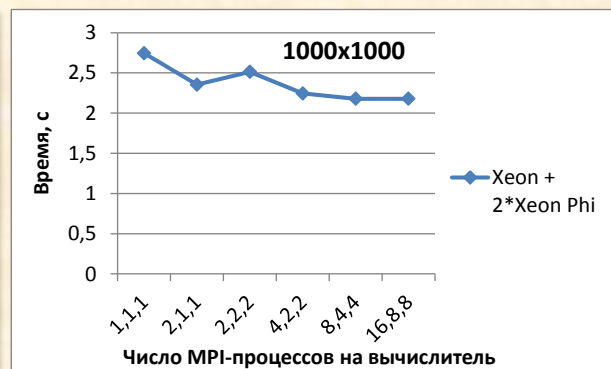
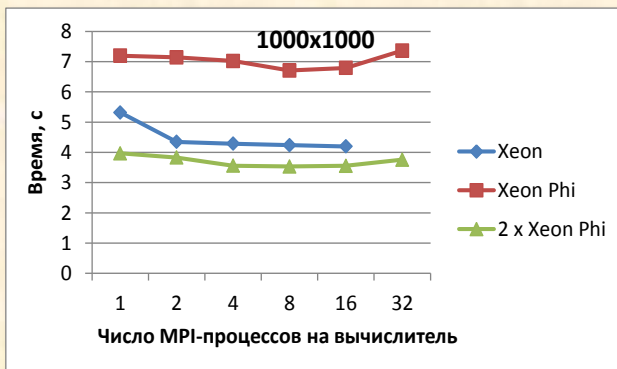
Обмен граничными значениями



Распараллеливание

Использование всех ресурсов узла

- Симметричный режим (MPI + OpenMP)

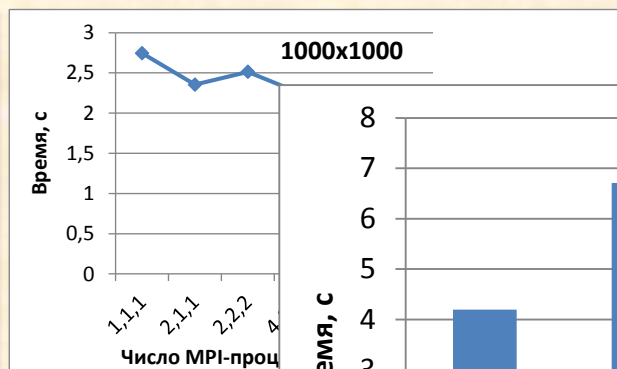
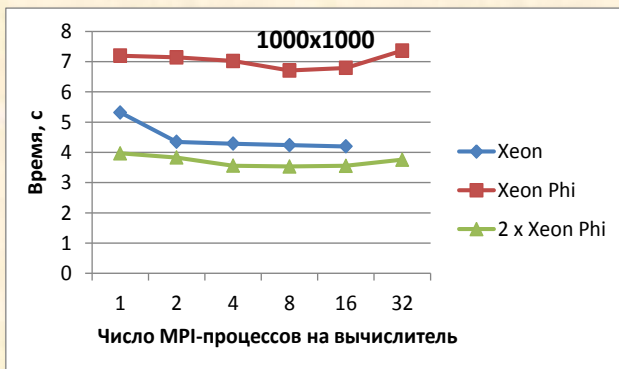


Определение оптимального соотношения
процессов/потоков

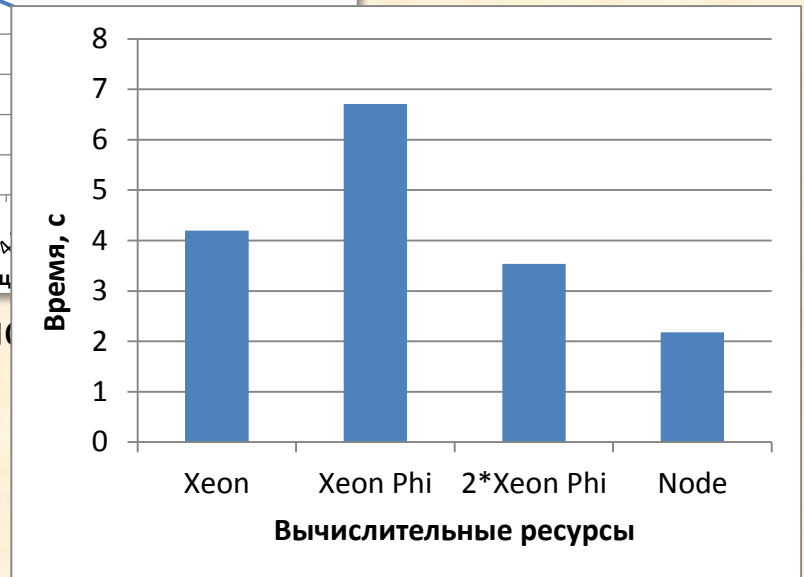
Распараллеливание

Использование всех ресурсов узла

- Симметричный режим (MPI + OpenMP)



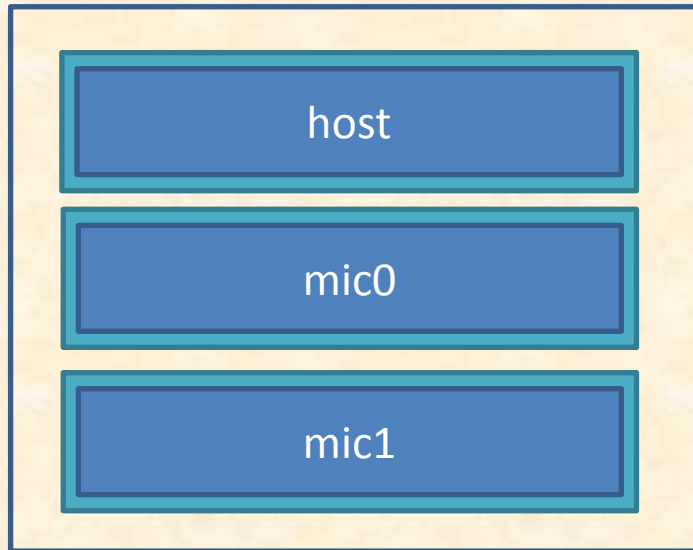
Определение оптимального соотношения
процессов/потоков



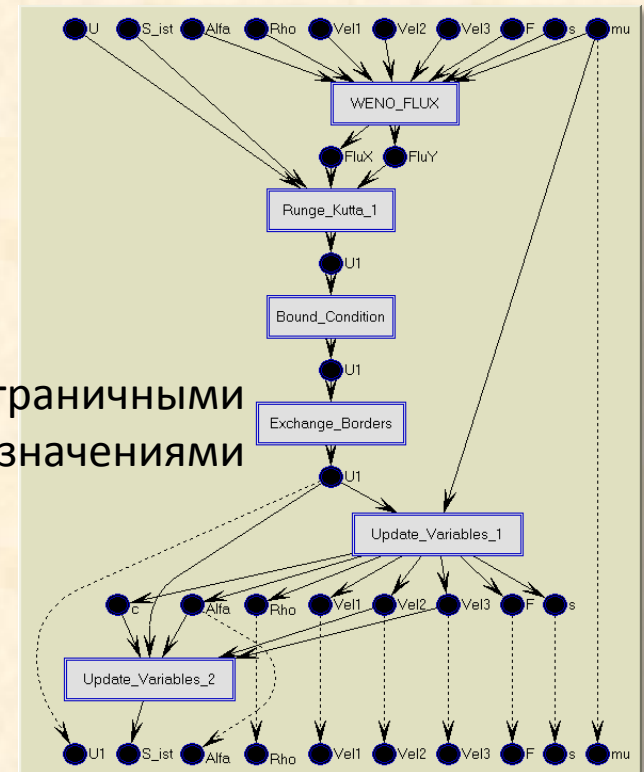
Распараллеливание

Распараллеливание в узле в режиме Offload

- Декомпозиция области моделирования на подобласти между хостом и сопроцессорами



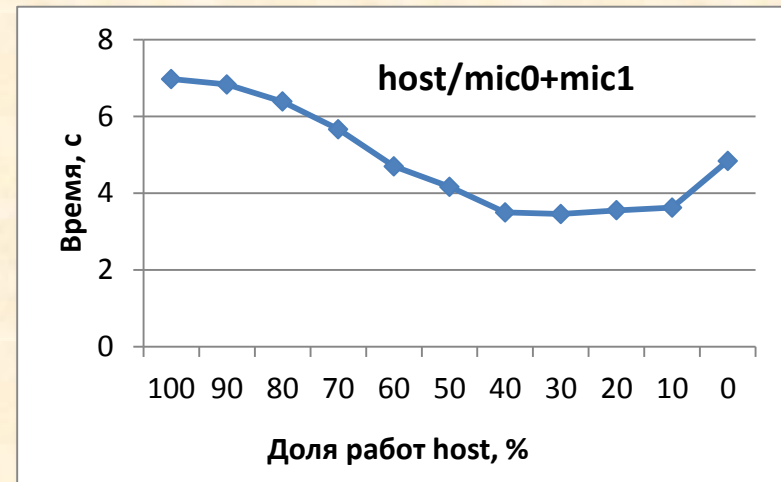
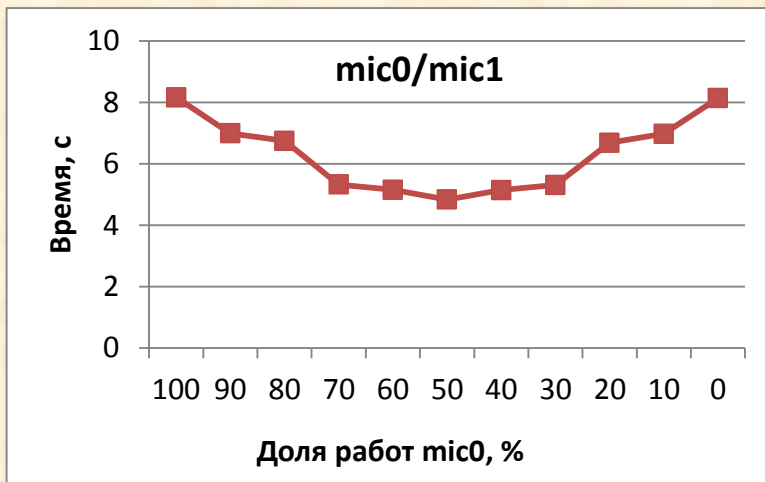
Обмен граничными значениями



Распараллеливание

Использование всех ресурсов узла

- Режим offload (+OpenMP)

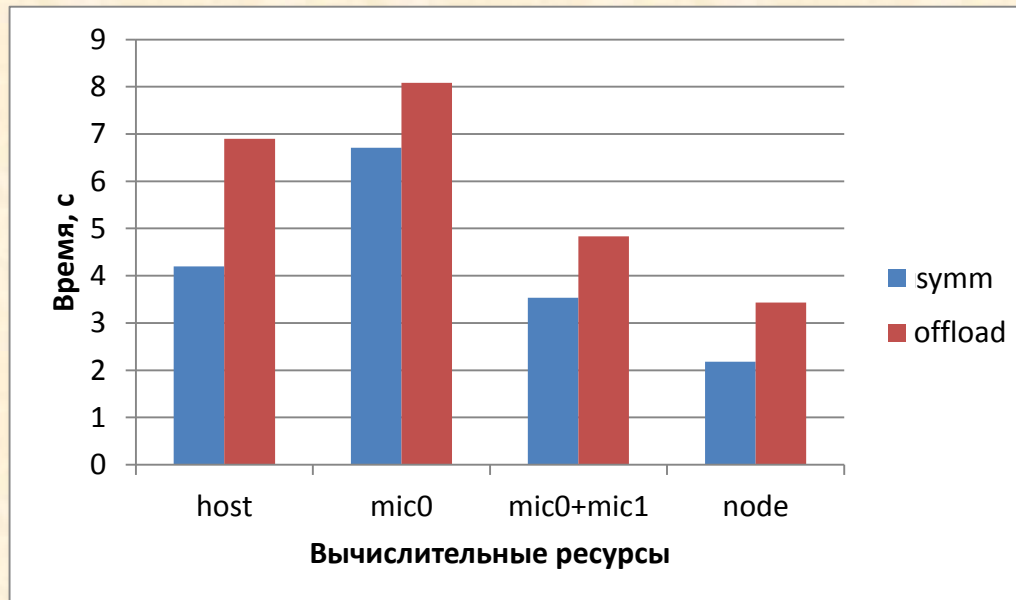


Определение оптимального соотношения работ
(1000x1000)

Распараллеливание

Использование всех ресурсов узла

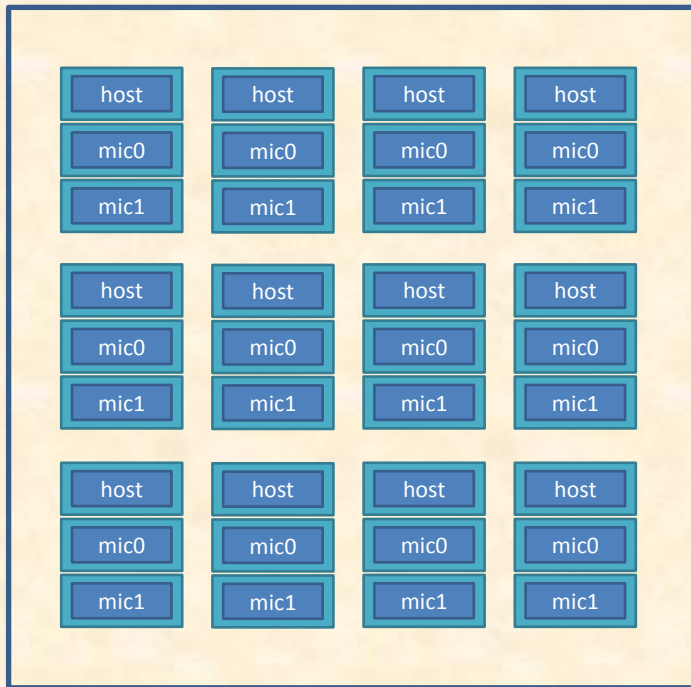
- Сравнение режимов



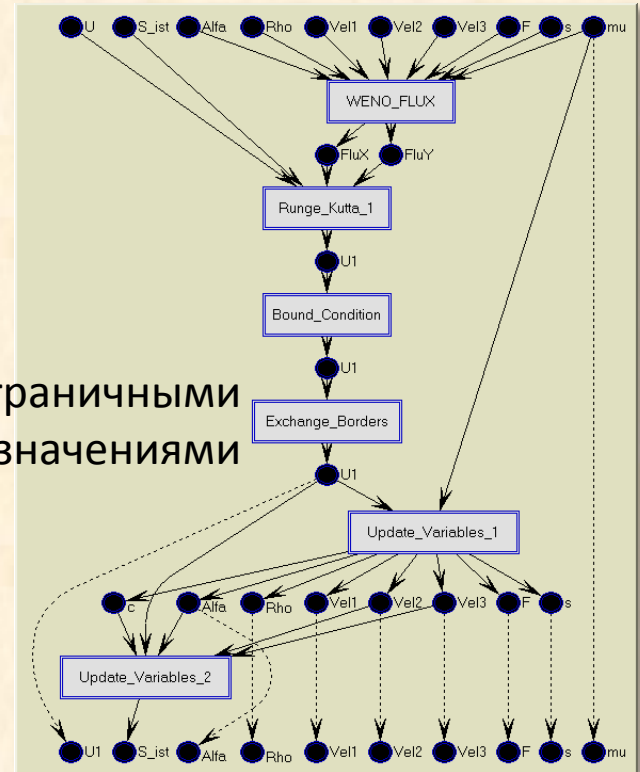
Распараллеливание

Распараллеливание в MPI + offload

- 2D декомпозиция области между узлами кластера и между вычислителями узла

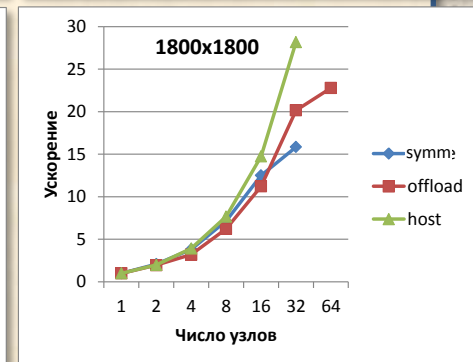
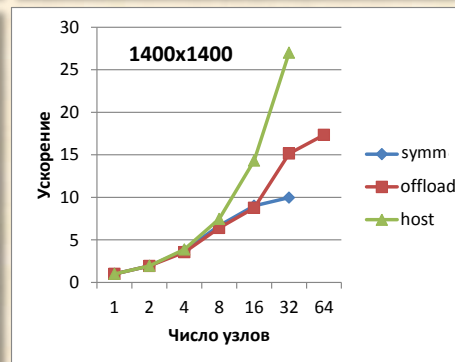
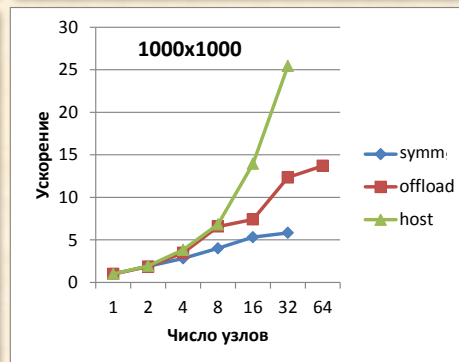
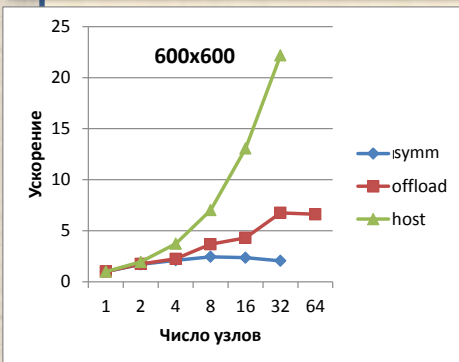
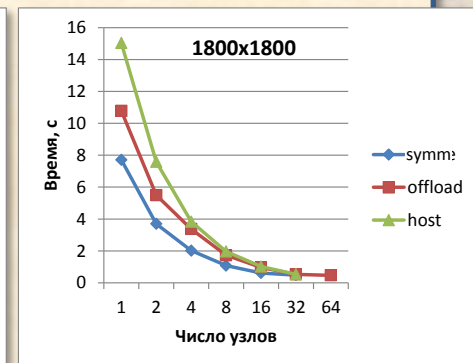
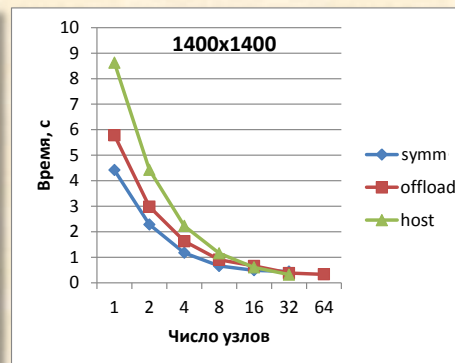
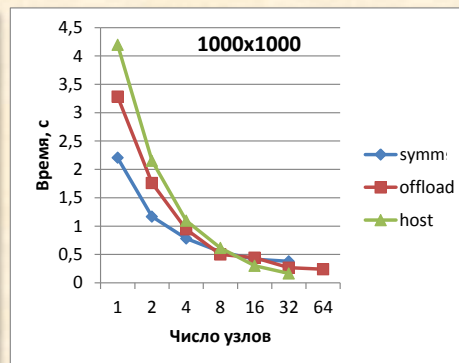
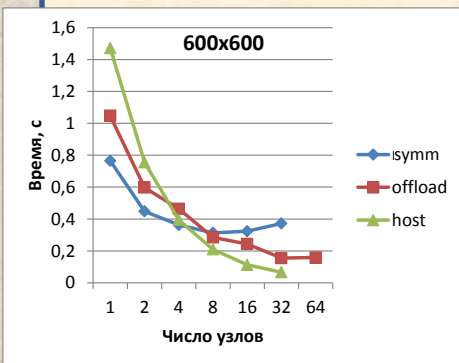


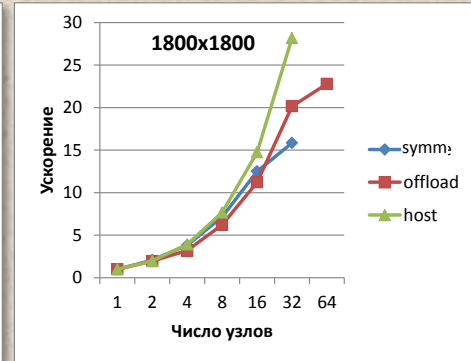
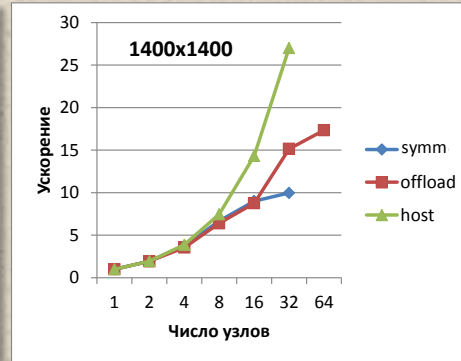
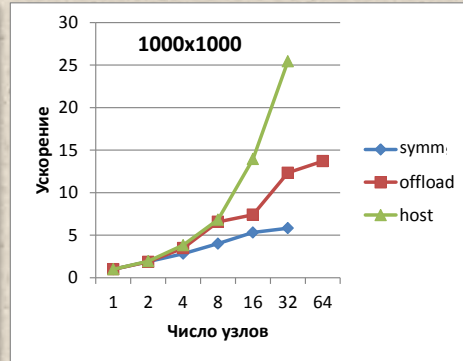
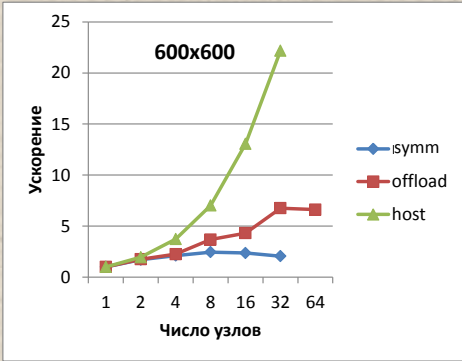
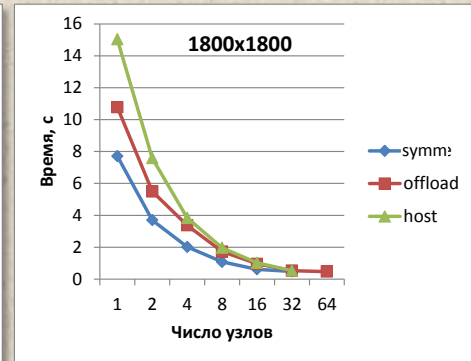
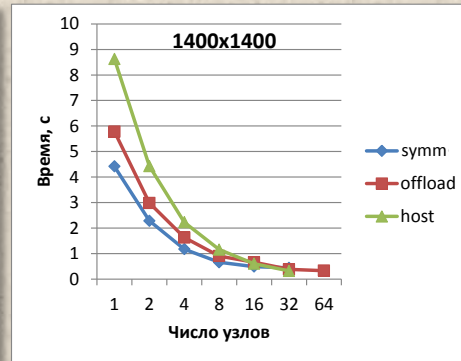
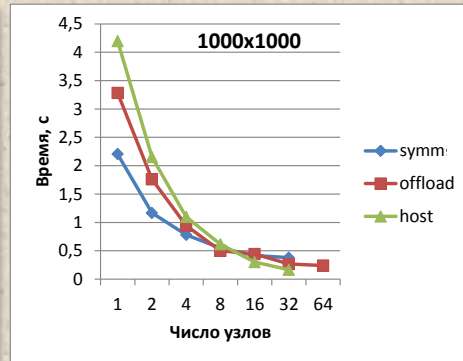
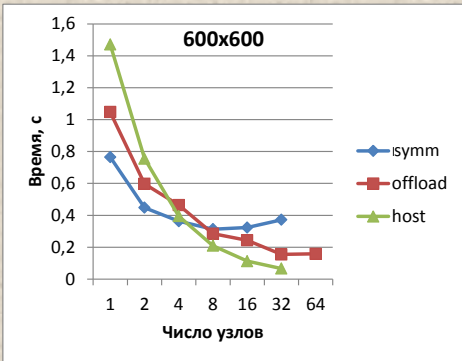
Обмен граничными значениями



Распараллеливание

Использование нескольких узлов кластера





Какие ограничения встретились

- Максимальный размер задачи, поместившейся в один ускоритель Xeon Phi (8 Гб памяти), составил 1800×1800
- В симметричном режиме программа работает не более чем на ~32 узлах кластера

Какой режим лучше использовать для данной задачи

- При малом числе узлов кластера – *симметричный*
- При большом числе узлов кластера – *только host-процессоры*

Выводы

- На данной задаче сопроцессор Xeon Phi сравним по производительности с 8-ядерным процессором Xeon E-2690 и при совместном использовании позволяет ускорить счёт до двух раз.
- В зависимости от числа узлов кластера и размера задачи быстрее работает симметричный режим или только host-процессоры. Режим offload почти нигде не выигрывает.
- Существующие средства программирования в режиме offload имеют некоторые недостатки, снижающие удобство использования этого режима для сложных задач.
- Компилятор не смог реализовать некоторые заявленные возможности по оптимизации кода (выровненный доступ к данным, ключ -ipo).

Спасибо за внимание!