Филиал МГУ имени М.В. Ломоносова в городе Сарове

Направление подготовки

«Фундаментальная информатика и информационные технологии»

Чернышов Михаил Михайлович

Параллельная программа (MPI & OpenMP) для уравнения колебаний струны с использованием явной разностной схемы «крест»

ОТЧЕТ

Саров, 2023

Содержание

1 Постановка задачи

Реализовать численное решение двумерного нестационарного уравнения колебания струны в прямоугольной области. Следует использовать явную разностную схему. Обратить внимание на необходимость использования равномерного распределения по процессам расчетной области.

Требования к программе:

1. Параллельная программа должна быть гибридной: одновременно использовать технологию MPI, для обеспечения взаимодействия вычислительных узлов, и одну из двух технологий Posix threads или OpenMP, для взаимодействия процессов, запущенных на ядрах процессоров. (Гибридная программа по желанию)
2. Программа должна демонстрировать эффективность не менее 50% на числе вычислительных ядер, не менее 108
3. Программа должна поддерживать блочное разбиение области на n частей по одной стороне прямоугольник и на m частей по другой стороне прямоугольника.

2 Математическая постановка задачи

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} = a^2 \Delta u + f(x,y,t), \;\; x \in (0, L_x), \;\; y \in \left(0, L_y\right), \;\; t \in (0, T] \\ u(x,y,0) = \varphi(x,y) \\ \dfrac{\partial u}{\partial t}(x,y) = \psi(x,y) \\ u(0,y,t) = \mu_1(y,t) \\ u(x,0,t) = \mu_2(x,t) \\ u(L_x,y,t) = \mu_3(y,t) \\ u\left(x, L_y, t\right) = \mu_4(x,t) \end{cases}$$

Со следующими параметрами:

$$L_x = 6$$
$$L_y = 4$$
$$T = 5$$
$$a = 0.7$$
$$f = -9\cos(3t)\sin(4x)\cos(5y) - 36\sin(6t)\cos(7x)\sin(8y)$$
$$+a^2(41\cos(3t)\sin(4x)\cos(5y) + 113\sin(6t)\cos(7x)\sin(8y))$$
$$\varphi(x,y) = \sin(4x)\cos(5y)$$
$$\psi(x,y) = 6\cos(7x)\sin(8y)$$
$$\mu_1(y,t) = \sin(6t)\sin(8y)$$
$$\mu_2(x,t) = \cos(3t)\sin(4x)$$
$$\mu_3(y,t) = \cos(3t)\sin(4L_x)\cos(5y) + \sin(6t)\cos(7L_x)\sin(8y)$$
$$\mu_4(x,t) = \cos(3t)\sin(4x)\cos\left(5L_y\right) + \sin(6t)\cos(7x)\sin\left(8L_y\right)$$

3 Аналитическое решение

$$u = \cos(3t)\sin(4x)\cos(5y) + \sin(6t)\cos(7x)\sin(8y)$$

4 Метод численного решения

Вводим равномерные сетки:

$$t_n = n\tau; \quad n = 0,1,\dots,N; \quad N\tau = T$$
$$x_i = ih_x; \quad i = 0,1,\dots,M_x; \quad M_x h_x = L_x$$
$$y_j = jh_y; \quad j = 0,1,\dots,M_y; \quad M_y h_y = L_y$$

Заменяя производные в уравнении конечными разностями, приходим к следующему разностному уравнению:

$$\frac{u_{i,j}^{n+1} - 2u_{i,j}^{n} + u_{i,j}^{n-1}}{\tau^2} = a^2\left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{{h_x}^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{{h_y}^2}\right) + f$$

Начальное условие задачи аппроксимируем со вторым порядком погрешности по τ:

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\tau} = \psi_{i,j} + \frac{\tau}{2}\left(a^2\left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{{h_x}^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{{h_y}^2}\right) + f\right)$$

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \tau\left(\psi_{i,j} + \frac{\tau}{2}\left(a^2\left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{{h_x}^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{{h_y}^2}\right) + f\right)\right)$$

Таким образом, нам известны значения на нулевом и первом временных слоях. Решения на следующих временных слоях могут быть найдены используя граничные условия и разностное уравнение, записанное в следующем виде:

$$u_{i,j}^{n+1} = 2u_{i,j}^{n} - u_{i,j}^{n-1} + \tau^2 \left(a^2 \left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{{h_x}^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{{h_y}^2} \right) + f \right)$$

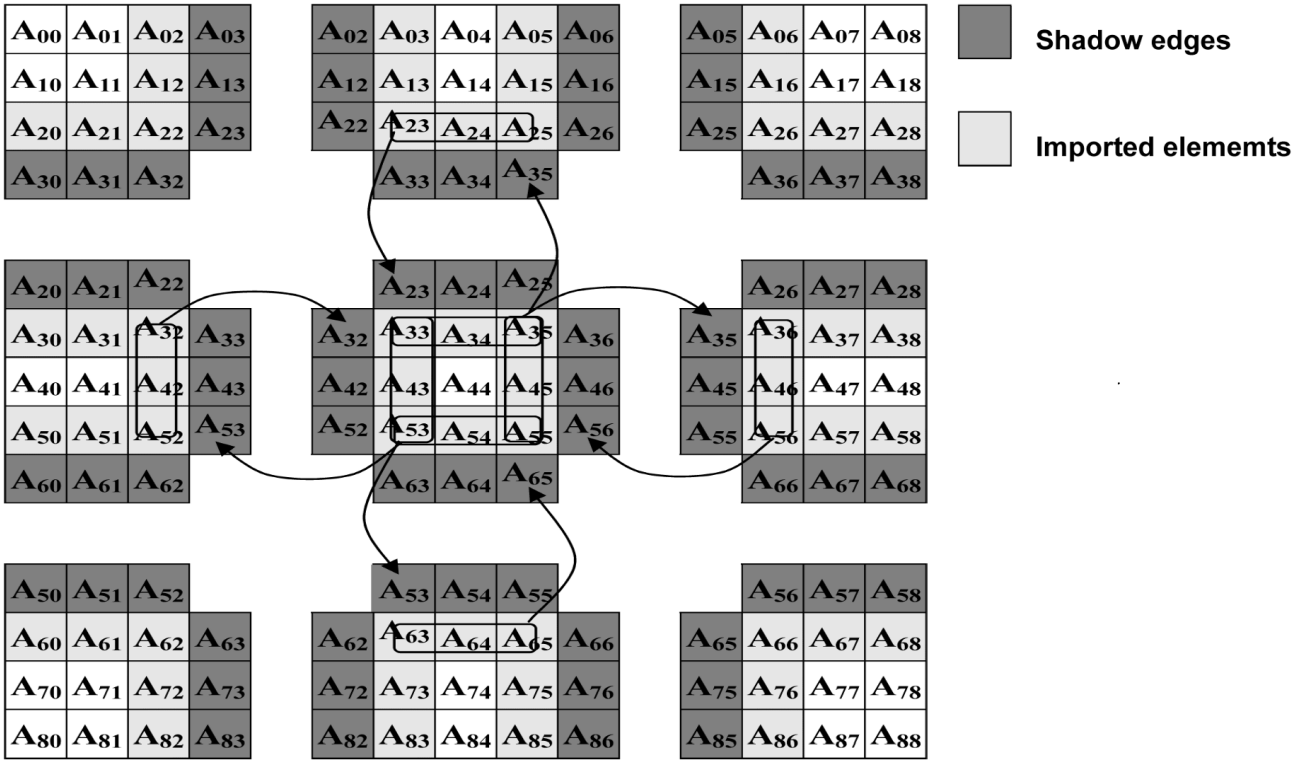
Погрешность рассматриваемой схемы – $O(\tau^2 + h_x^2 + h_y^2)$.

Условие устойчивости рассматриваемой схемы:

$$a\tau \sqrt{\frac{1}{h_x^2} + \frac{1}{h_y^2}} \leq 1$$

5 Способ декомпозиции области

Реализовано блочное разбиение.



6 Вычислительная система

Вычислительная система ЦХАБД состоит из 7 узлов. На каждом узле по 2 процессора Intel Xeon Gold 6140.

Характеристики процессора Intel Xeon Gold 6140

Год выхода	2017
Socket	LGA3647
Шина	10.4 GT/s UPI
Количество ядер	18
Количество потоков	36
Базовая частота	2300 MHz
Turbo Boost	3700 MHz
Архитектура (ядро)	Skylake-SP
Техпроцесс	14 nm
TDP	140 W
Макс. температура	91° C
Кэш L1, КБ	18x32 + 18x32
Кэш L2, КБ	18x1024
Кэш L3, КБ	25344
Контроллер оперативной памяти	6-канальный (DDR4-2666) поддерживается ECC память
Контроллер PCIe	PCI Express 3.0 (48 линий)

7 Аналитическое время решения

Количество узлов по времени:

$$N + 1 \approx N$$

Количество узлов по пространству:

$$(M_x + 1)(M_y + 1) \approx M_x \times M_y$$

При большом количестве временных шагов можно пренебречь уникальностью первых двух шагов. При большом количестве пространственных узлов можно пренебречь вычислением граничных условий и считать их как обычные вычисления внутренней области. Количество узлов, для которых необходимо произвести вычисления:

$$N \times M_x \times M_y$$

Для получения времени необходимо умножить на количество тактов на узел C, разделить на тактовую частоту $clock\_rate$ и разделить на количество вычислительных ядер p:

$$N \times M_x \times M_y \times C \div clock\_rate \div p$$

В этой оценке не хватает временных затрат на пересылки данных. Пренебрегаем уникальностью процессов, у которых нет соседей со всех сторон, тогда в рамках обмена по горизонтали каждый процесс должен:

- получить левую теневую грань;
- отправить правую грань;
- получить правую теневую грань;
- отправить левую грань.

Аналогично для обмена по вертикали. Такие обмены происходят на каждом временном шаге. Таким образом, получаем оценку затрат на коммуникацию:

$$N \times 4 \times \frac{sizeof(element) \times (\frac{M_y}{n_y} + \frac{M_x}{n_x})}{bandwidth},$$

где

- $sizeof(element)$ – размер одного элемента массива в байтах;
- n_y – количество блоков по вертикали;
- n_x – количество блоков по горизонтали;
- $\frac{M_y}{n_y}$ – размер блока по вертикали, пренебрегаем остатком от деления;
- $\frac{M_x}{n_x}$ – размер блока по горизонтали, пренебрегаем остатком от деления;
- $bandwidth$ – пропускная способность сети (B/s).

Таким образом, получаем итоговую оценку:

$$N \times M_x \times M_y \times C \div clock\_rate \div p + N \times 4 \times \frac{sizeof(element) \times (\frac{M_y}{n_y} + \frac{M_x}{n_x})}{bandwidth}$$

Определим значения параметров:

- $clock\_rate = 2.3 \times 10^9$;
- $sizeof(element)$ – 8 байт, так как я использую double;
- $C = 21 + 12 \times 30 = 381$, так как каждая тригонометрическая операция на современных процессорах занимает около 30 тактов;
- $bandwidth = 1.23 \times 10^9$, получено экспериментально.

Для измерения пропускной способности "точка-точка" используется следующая методика. Процесс с номером 0 посылает процессу с номером 1 сообщение длины L байт. Процесс 1, приняв сообщение от процесса 0, посылает ему ответное сообщение той же длины. Используются блокирующие (blocking) вызовы MPI (MPI_Send, MPI_Recv). Эти действия повторяются K раз с целью минимизировать погрешность за счет усреднения. Процесс 0 измеряет время t, затраченное на все эти обмены. Пропускная способность определяется по формуле:

$$bandwidth = 2 \times K \times L \div t$$

Таким образом, аналитическая оценка времени решения:

$$N \times M_x \times M_y \times 381 \div (2.3 \times 10^9) \div p + N \times 4 \times \frac{8 \times (\frac{M_y}{n_y} + \frac{M_x}{n_x})}{1.23 \times 10^9}$$

8 Отклонения от точного решения

Вычисления выполнялись с одинаковым шагом по пространству в каждом направлении $(h = h_x = h_y)$.

$$\left\|u(ih_x, jh_y, T) - u_{i,j}^N\right\|_2 = \sqrt{h_x h_y \sum_{i=0}^{M_x}\sum_{j=0}^{M_y}\left(u(ih_x, jh_y, T) - u_{i,j}^N\right)^2}$$

h \ τ	0.01	0.005	0.0025	0.00125
0.01	0.00233339	0.00335007	0.00362810	0.00369852
0.005	—	0.00058308	0.00083756	0.00090728
0.0025	—	—	0.00014575	0.00020939
0.00125	—	—	—	0.00003644

$$\left\|u(ih_x, jh_y, T) - u_{i,j}^N\right\|_C = \max_{\substack{i=0,\dots,M_x \\ j=0,\dots,M_y}} \left|u(ih_x, jh_y, T) - u_{i,j}^N\right|$$

h \ τ	0.01	0.005	0.0025	0.00125
0.01	0.00172093	0.00274429	0.00300307	0.00306776
0.005	—	0.00042999	0.00068593	0.00075082
0.0025	—	—	0.00010749	0.00017150
0.00125	—	—	—	0.00002687

9 Численные расчеты

Расчеты проводились 10 раз с последующим усреднением результатов.

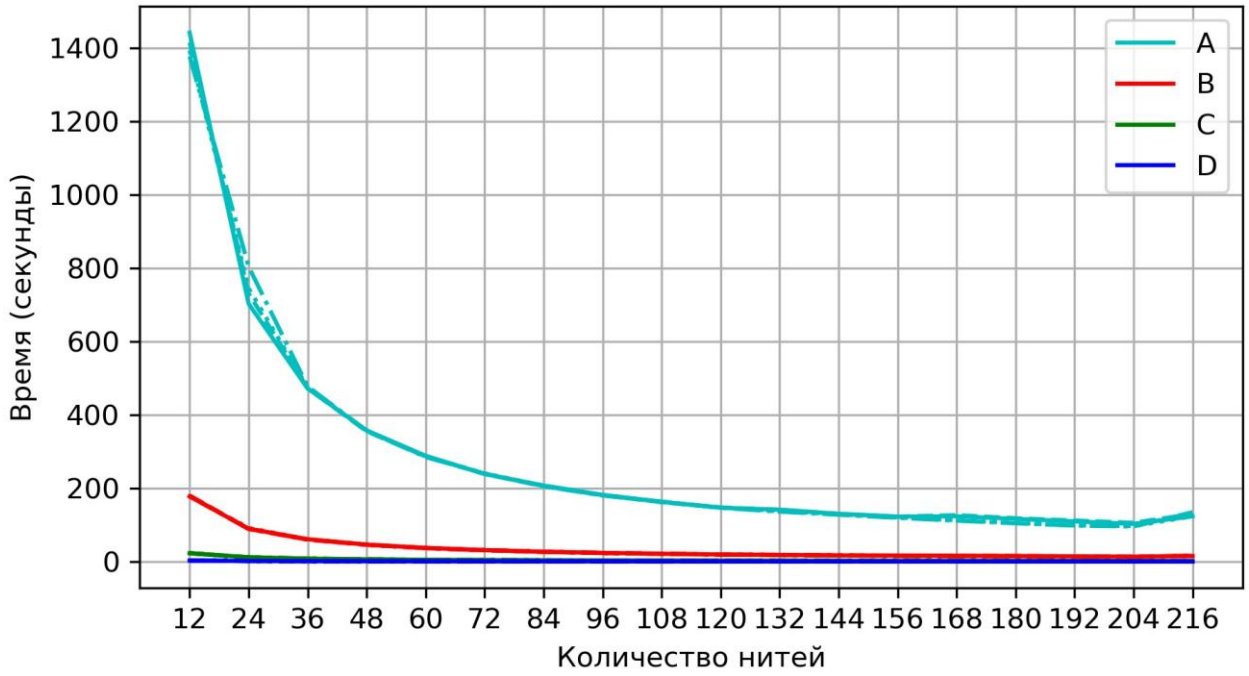
Было реализовано 2 программы: в 1-й использовались синхронные операции обмена, а во 2-й – асинхронные. Также тестировались следующие случаи: по 1 mpi процессу на каждый узел (всего 6 mpi процессов, разбиение 2×3) и по 1 mpi процессу на каждый numa/socket/L3cache (всего 12 mpi процессов, разбиение 3×4).

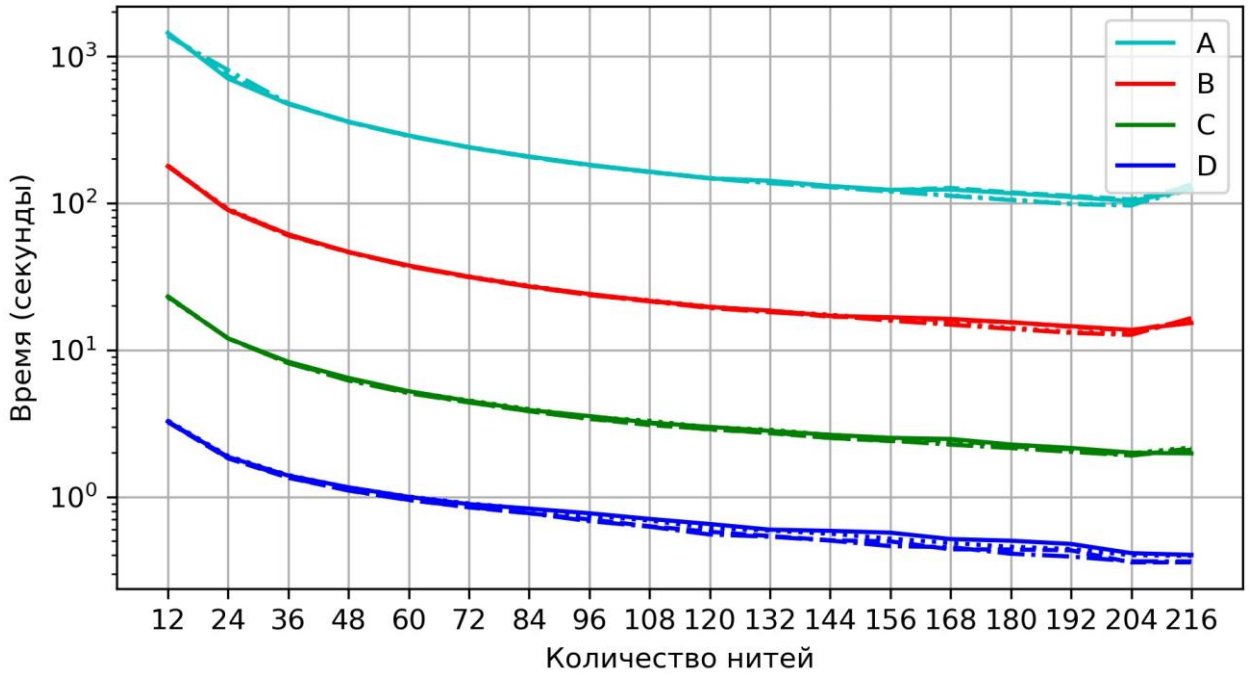
Наборы данных на которых проводились расчеты

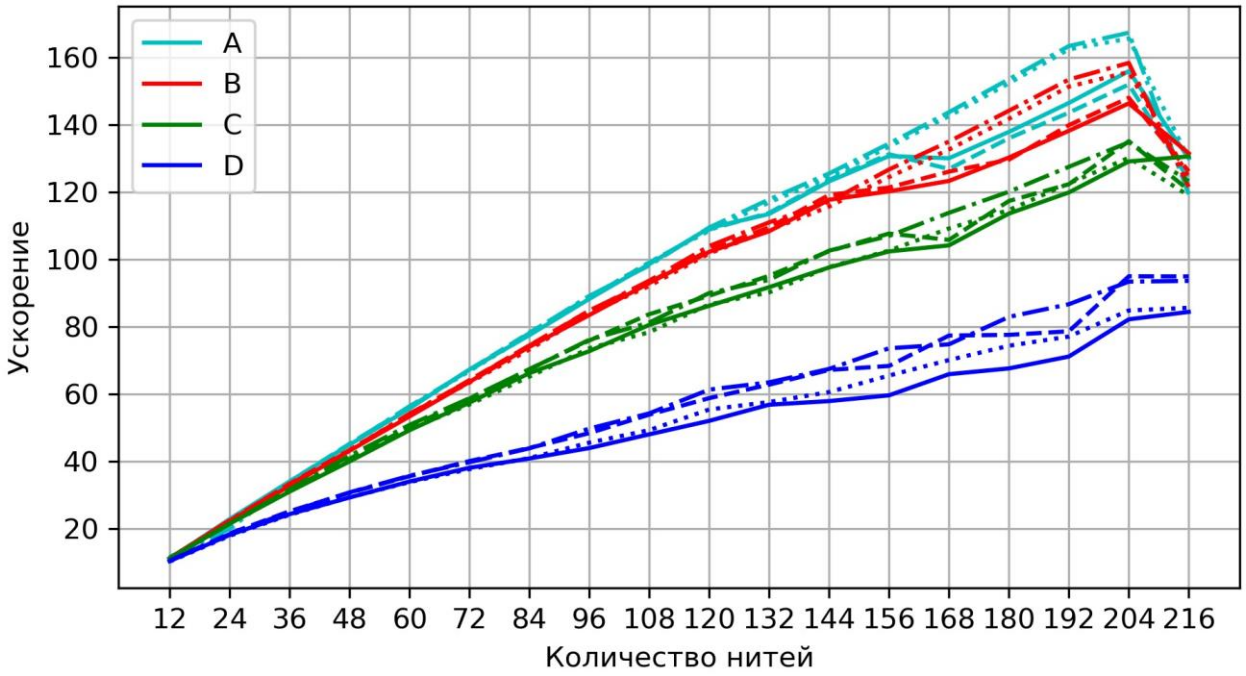
Набор данных	N	M_x	M_y
A	4000	4800	3200
B	2000	2400	1600
C	1000	1200	800
D	500	600	400

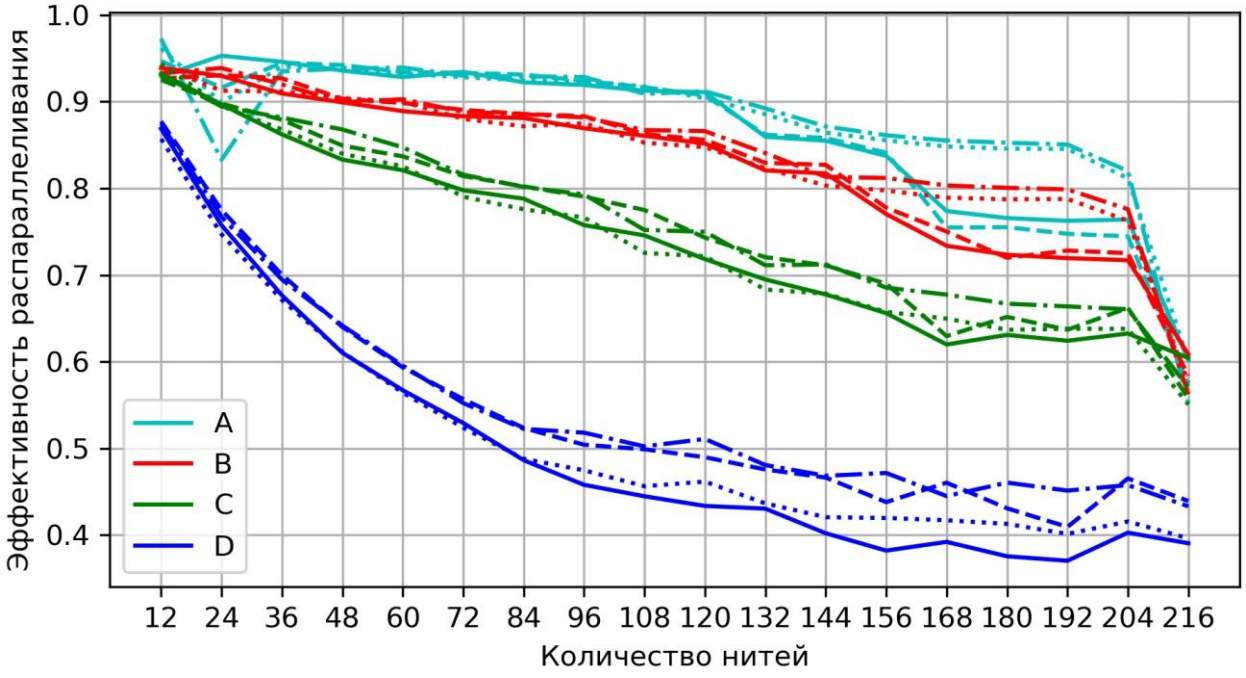
Далее представлены картинки, содержащие сведения о времени решения, ускорении и эффективности распараллеливания. Разные виды линий используются для разных программ и распределений mpi процессов:

- сплошная линия – синхронные обмены, распределение по узлам;
- штриховая линия – асинхронные обмены, распределение по узлам;
- линия из точек – синхронные обмены, распределение по numa;
- штрихпунктирная линия – асинхронные обмены, распределение по numa.


A
B
C
D
Время (секунды)
0
200
400
600
800
1000
1200
1400
12 24 36 48 60 72 84 96 108 120 132 144 156 168 180 192 204 216
Количество нитей


A
B
C
D
Время (секунды)
10^0
10^1
10^2
10^3
12 24 36 48 60 72 84 96 108 120 132 144 156 168 180 192 204 216
Количество нитей


A
B
C
D
Ускорение
160
140
120
100
80
60
40
20
12 24 36 48 60 72 84 96 108 120 132 144 156 168 180 192 204 216
Количество нитей


A
B
C
D
Эффективность распараллеливания
1.0
0.9
0.8
0.7
0.6
0.5
0.4
12 24 36 48 60 72 84 96 108 120 132 144 156 168 180 192 204 216
Количество нитей

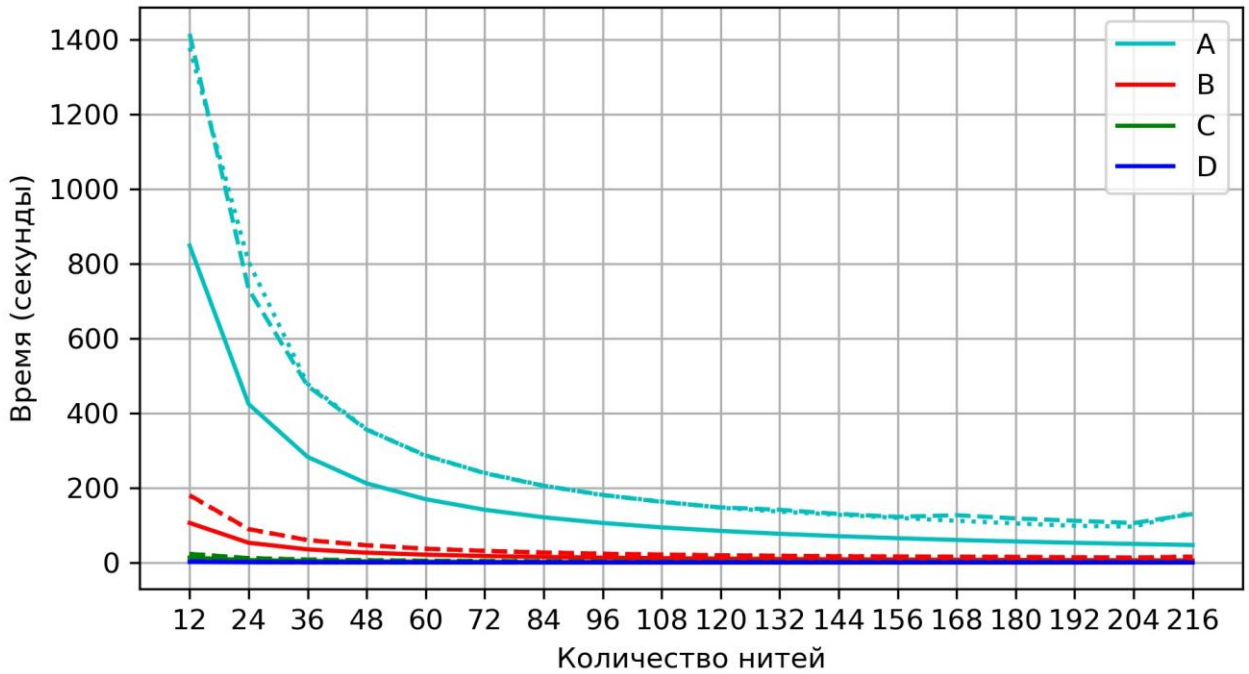
На вычислительно более сложном наборе данных распределение процессов по numa и использование асинхронных операций является лучшим решением: обеспечивается большая локальность данных, а также асинхронные обмены имеют нефиксированный порядок выполнения.

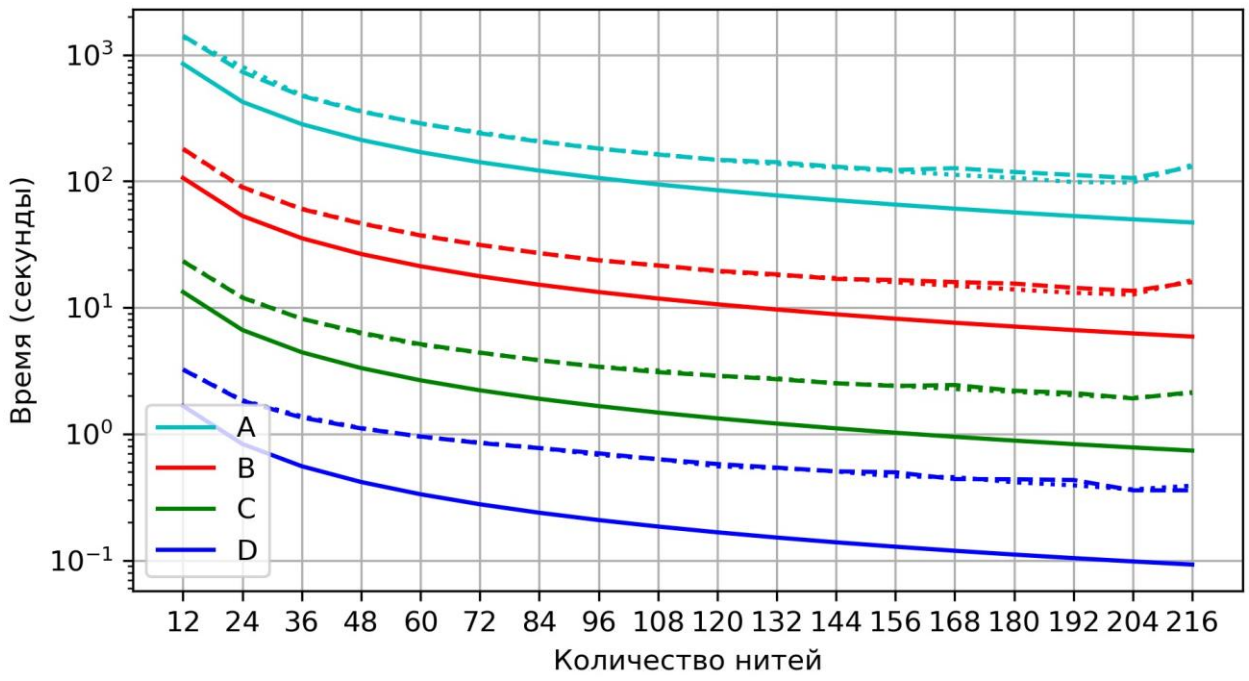
Сильное падение производительности на максимальном количестве вычислительных ядер связано с тем, что на узлах установлена система и ей нужны ресурсы на обеспечение функционирования.

9.1 Сравнение с аналитическим временем решения

Далее представлены картинки, содержащие сведения о времени решения. Разные виды линий используются для разных программ и распределений:

- сплошная линия – аналитическая оценка из пункта 7;
- штриховая линия – асинхронные обмены, распределение по узлам;
- линия из точек – асинхронные обмены, распределение по numa.





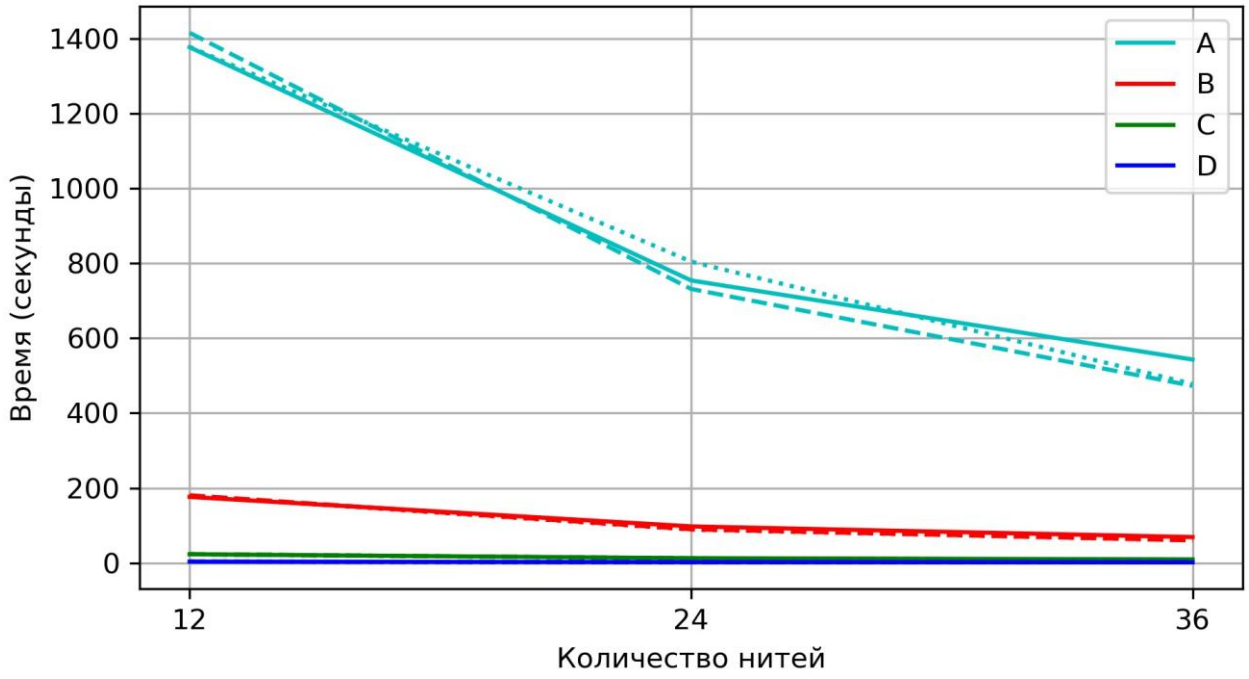
Реальное время решения отличается от аналитического, так как аналитическое не учитывает время запросов в память.

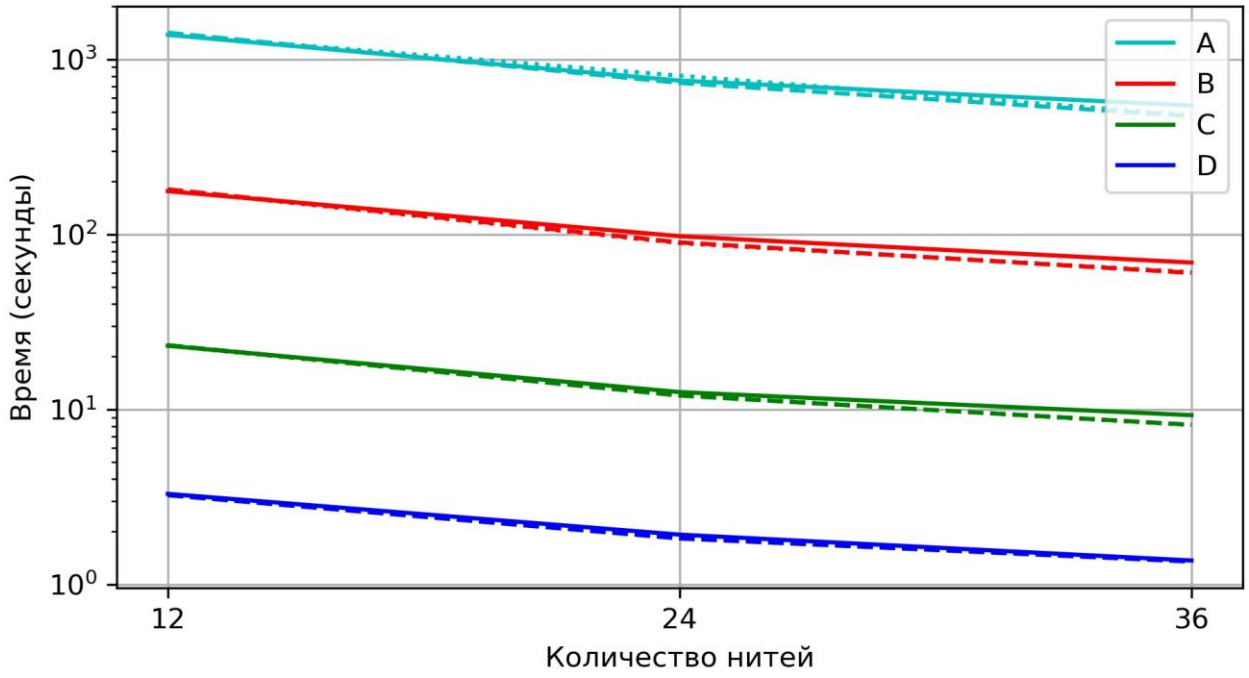
Построение графиков ускорения и эффективности не имеет смысла, так как последовательная программа не соответствует аналитической оценке.

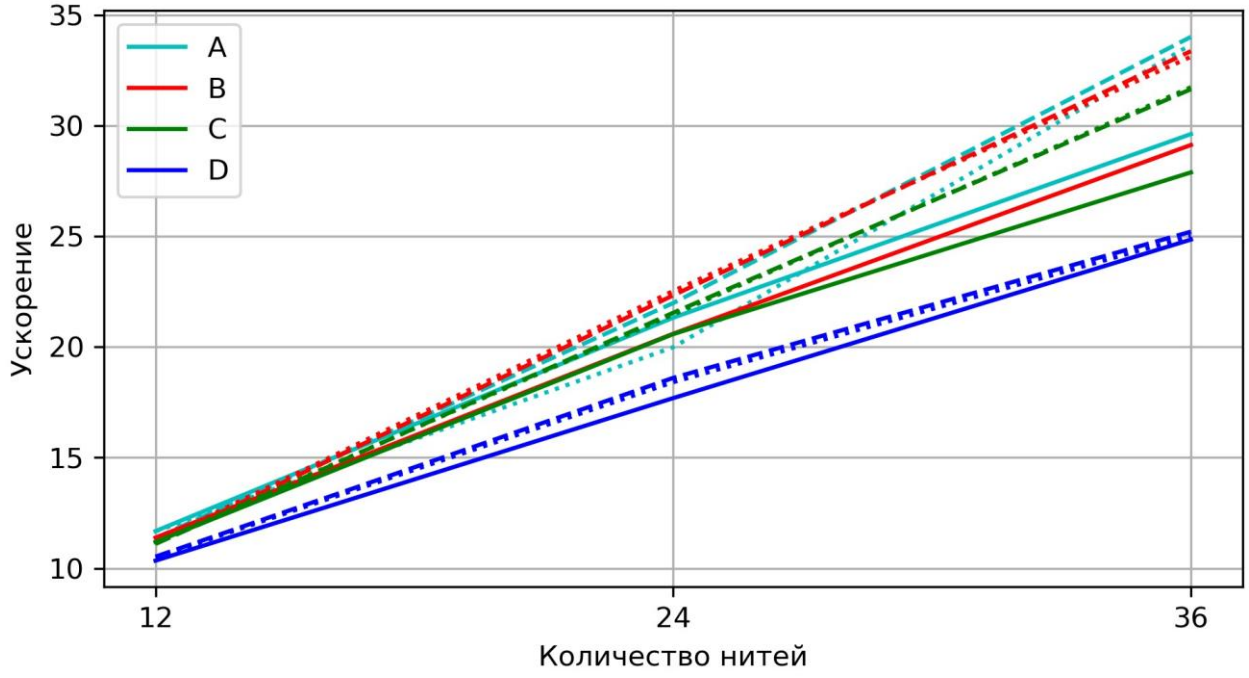
10 Сравнение с OpenMP

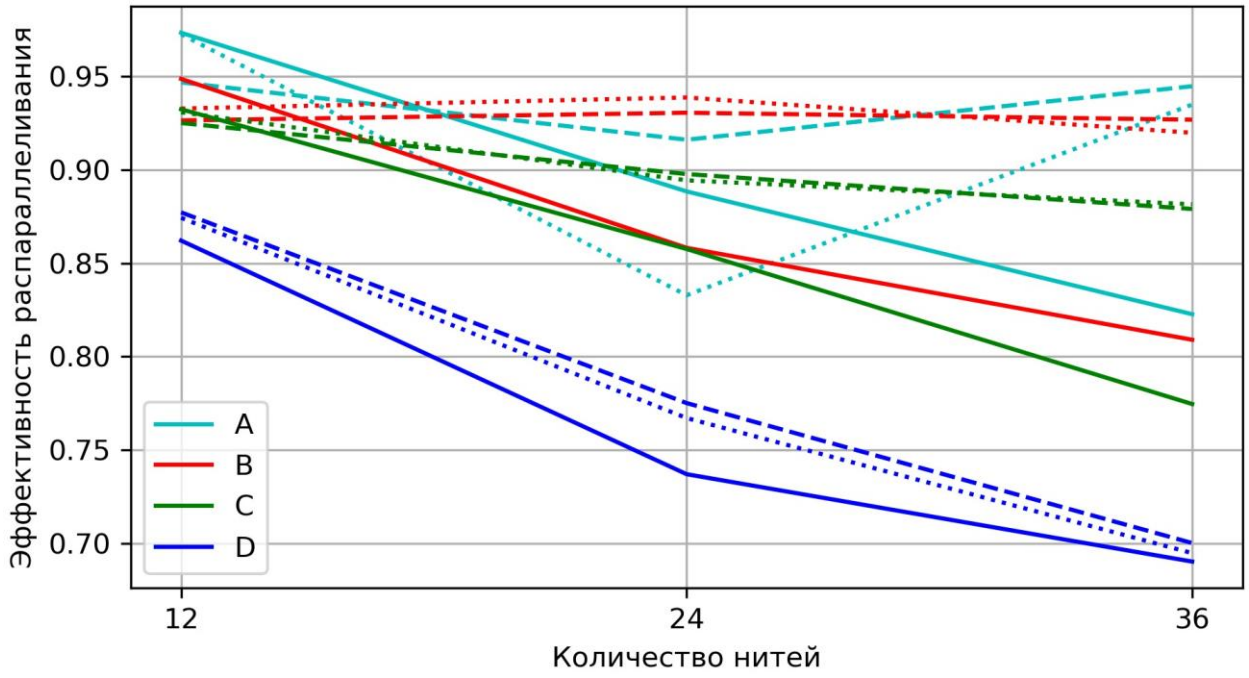
Далее представлены картинки, содержащие сведения о времени решения, ускорении и эффективности распараллеливания. Разные виды линий используются для разных программ и распределений mpi процессов:

- сплошная линия – OpenMP;
- штриховая линия – асинхронные обмены, распределение по узлам;
- линия из точек – асинхронные обмены, распределение по numa.




A
B
C
D
Время (секунды)
10^3
10^2
10^1
10^0
12
24
36
Количество нитей


A
B
C
D
Ускорение
35
30
25
20
15
10
12
24
36
Количество нитей



MPI программы показали себя лучше чем OpenMP, так как обладают большей локальностью данных.

11 Анализ полученных результатов

Было реализовано 2 программы: с синхронными обменами и с асинхронными. Рассматривался запуск программ с распределением процессов по узлам вычислительной системы и по numa/socket/L3cache.

Самой лучшей на вычислительно более сложном наборе данных оказалась связка: асинхронные обмены и распределение по numa. Это произошло, так как обеспечивается большая локальность данных, а также асинхронные обмены имеют нефиксированный порядок выполнения.

На лучшей комбинации: асинхронные обмены и распределение по numa – эффективность распараллеливания составила более 90% на 120 вычислительных ядрах.

При уменьшении шагов в два раза по пространственным и временной сеткам отклонение от точного решения уменьшается в 4 раза, что согласуется с оценкой погрешности схемы.

12 Сведения о том, как компилировалась и запускалась программа

Программа запускалась на ЦХАБД.

Компиляция программы:

```
mpicc wave_eq_2d_mpi_openmp.c -lm -fopenmp -o wave_eq_2d_mpi_openmp
mpicc wave_eq_2d_mpi_openmp_v2.c -lm -fopenmp -o wave_eq_2d_mpi_openmp_v2
```

Запуск производился с помощью sbatch:

```
sbatch run_mpi_openmp
```

run_mpi_openmp – специальный файл с параметрами запуска, составленный по аналогии с примерами, которые хранились на суперкомпьютере.

Следует заметить, что при запуске программы аналогично примерам каждый mpi процесс привязан к вычислительному ядру, поэтому реальные нити не запускаются.

Следующая опция убирает привязку к ядрам:

```
mpirun --bind-to none
```

Процессу становятся доступны все ядра узла.

Для распределения по numa:

```
mpirun --map-by numa
```

Процессы раскладываются на разные numa и каждому становятся доступны все ядра из текущего numa.

Для ЦХАБД numa эквивалентно socket и L3cache, и это все эквивалентно одному процессору на узле.

При таком запуске нити дейсвительно будут работать, но каждая нить будет прикреплена ко всем ядрам в пределах распределения, что может плохо сказываться на производительности. Для закрепления нитей на ядрах используются переменные окружения со следующими параметрами:

```
export OMP_PROC_BIND=close
export OMP_PLACES=cores
```

Оказалось, что к определенным вычислительным ядрам будут привязаны даже mpi процессы до входа в параллельную область OpenMP.

13 Первоначальные проблемы с запуском MPI & OpenMP

Когда я первый раз попытался запустить свою программу на вычислительной системе, то столкнулся с тем, что увеличение количества нитей не давало производительности, хотя были задействованы не все вычислительные ядра. Мне подсказали, что можно посмотреть маску процесса: к каким вычислительным ядрам он привязан.

Для ясного понимания ситуации я реализовал программу, которая выводит информацию на каких кластерах запущен mpi процесс и к каким ядрам он привязан, аналогичный вывод информации после входа в параллельную область OpenMP.

После нескольких экспериментов с параметрами mpirun стало понятно как правильно запускать и как привязать нити и процессы к ядрам.

Кстати, опция sbatch --ntasks-per-socket=n, по-моему мнению, должна работать как mpirun --map-by socket, но я не заметил, что она вообще что-то делает.