Оптимизация отображения неоднородно взаимодействующих MPI процессов на вычислительную архитектуру

В.В. Гетманский[1], В.С. Чалышев[1], Д.И. Крыжановский[1], Е.И.Лексиков[2]

Singularis-Lab Ltd.[1], Intel[2]

Разработан метод отображения на кластерную архитектуру неоднородно взаимодействующих параллельных процессов в вычислительном приложении, использующем MPI. Метод предназначен для сокращения задержек при синхронизации за счет назначения наиболее интенсивно взаимодействующих процессов, на вычислительные ядра с наиболее быстрым интерконнектом. Метод использует представление вычислительной задачи и архитектуры кластера в виде взвешенного графа. Разработан эвристический алгоритм, дающий за приемлемое время результат отображения номеров процессов на номера вычислительных ядер кластера. На примере хорошо масштабируемого вычислительного пакета получено ускорение вычислений на 16-20% в результате оптимизации отображения для тестов от 300 до 4800 процессов.

1. Постановка задачи

Проблема отображения параллельной программы на архитектуру вычислительной системы с целью уменьшения времени обмена данными рассмотрена в ряде работ отечественных [1, 2] и зарубежных [3, 4, 5] авторов. В настоящей работе рассмотрена разработка и тестирование метода отображения графа задачи на граф вычислительной системы. Вершины графа задачи соответствуют параллельным процессам, а ребра графа задачи взвешены объемами пересылаемых между процессами данных. Вершины соединены ребрами только если соответствующие им процессы обмениваются данными. В научных вычислительных пакетах граф задачи имеет произвольную структуру. Пример такого графа для вычислительного пакета ZeusMP, запущенного на 2 узлах кластера показан на рис. 1. Ширина линий, обозначающих ребра для наглядности пропорциональна весу ребра. Данное обозначение будет использовано на остальных рисунках.

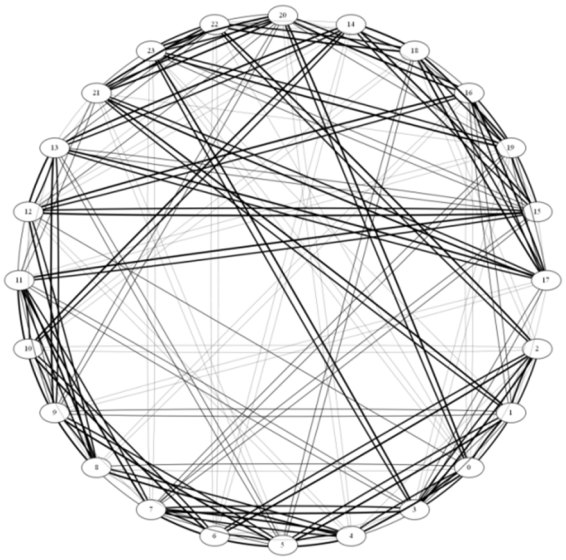


Рис. 1. Граф задачи ZeusMP

Граф системы представляет собой топологию кластера с учетом различной производительности каналов связи (интерконнекта). Ребра графа системы взвешены постоянными коэффициентами, представляющими соотношение времени передачи данных внутри узла кластера и времени передачи данных между узлами.

Современные кластеры состоят из многосокетных узлов, как правило, содержащих сопроцессоры GPGPU, FPGA или MIC. Каналы связи также работают с различной скоростью (Общая память, InfiniBand, Ethernet). Таким образом, граф кластера тоже может иметь неоднородную структуру из-за сложной конфигурации узлов кластера (рис. 2).

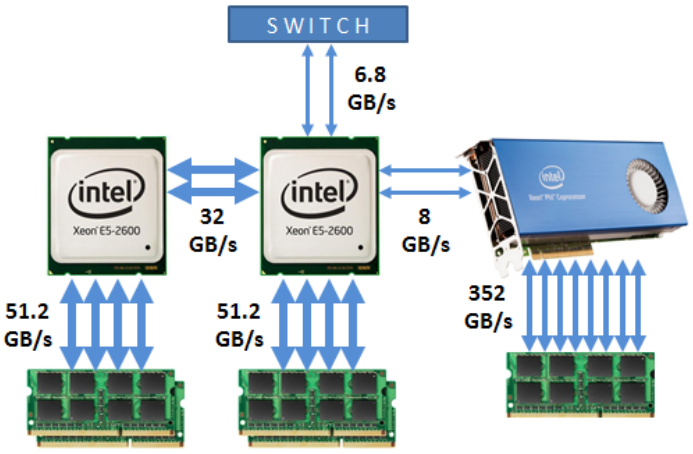


Рис. 2. Конфигурация узлов гетерогенного кластера

Основное проверяемое предположение в настоящей работе в том, что обмен данными можно ускорить, если распределить процессы на кластере так, что наиболее интенсивный обмен (ребра графа задачи с максимальным весом) будет происходить по наиболее быстрому каналу связи (ребра графа системы с максимальным весом). На основе предположения возникает следующая постановка задачи.

Обозначим граф задачи как

$$G_1(P, L), L_i = (n_i, d_i), i = 1, \dots K,$$

где P – набор вершин графа (процессы), L – набор связей (взаимодействие процессов), n_i–частота обмена данными, d_i–объем обмениваемых данных, K–число связей.

Обозначим граф системы как

$$G_2(V, D), D_j = (l_j, b_j), j = 1, \dots, S,$$

где V–множество вершин (соответствующих вычислительным ядрам центральных процессоров или сопроцессоров), D–множество ребер (каналы передачи данных), l_j–задержка, b_i–пропускная способность, S–общее число каналов передачи данных. В первом приближении в графе системы передача данных возможна между любой парой вершин, то есть G_2– полносвязный граф и $K < S$.

Искомое отображение можно записать как

$$G_1 \to G_2\ (P \to V): L_i \to D_j, F(D, L) \to min.$$

Минимизируемая целевая функция задает временную задержку на пересылки и имеет вид

$$F(D, L) = \sum_{k=1}^{N} T_k,\ \ T_k(D_i, L_j) = l_i \cdot n_j + \frac{d_j}{b_i}, F \to min.$$

Для разработанного прототипа используется упрощенная формулировка. Все ребра графа задачи взвешены объемом обмениваемых данных между процессами, которым соответствуют вершины ребра. Граф системы взвешен коэффициентом задержки канала связи r_i. Для задания того, что сетевое взаимодействие заведомо медленнее взаимодействия через общую память r_i определено как

$$r_i = \begin{cases} 1, \text{если обмен через общую память} \\ 2, \text{если обмен через } InfiniBand \end{cases}$$

Целевая функция вычисляется как сумма произведений весов ребер:

$$\tilde{F}(\tilde{D}, L) = \sum_{k=1}^{N} \tilde{T}_k,\ \ \tilde{T}_k(\tilde{D}_i, L_j) = r_i \cdot d_j,\ \tilde{D}_i = (r_i), \tilde{F} \to min.$$

Таким образом, для решения задачи отображения необходимо построить граф задачи и граф системы, разработать алгоритма поиска отображения графа задачи на граф системы с целью минимизации суммы произведений весов ребер.

2. Алгоритм отображения

Разработаны два алгоритма отображения: первый использует переборную стратегию, второй «жадную» стратегию. Для графов, содержащих более 10 вершин полный перебор выполняется неоправданно долго, поэтому практически применим только второй алгоритм, который дает либо оптимальное решение, либо близкое к оптимальному. Экспериментально установлено, что значение целевой функции становится не хуже исходного после работы алгоритма для проведенных тестов.

2.1. Алгоритм полного перебора

Алгоритм ищет решение с помощью полного перебора с возвратом. Вычислительная сложность алгоритма $O\left(\binom{N}{M} \cdot M!\right)$, где N – число вершин в графе системы, M – число вершин в графе задачи. Таким образом, на практике перебор можно использовать только для графов системы, содержащих до 15 вершин и для графов задачи, содержащих до 10 вершин.

2.2. Алгоритм с «жадной» стратегией

Реализация основана на похожем подходе, описанном в работе [5]. Основное отличие предлагаемой реализации в том, что отсутствует жесткое требование равенства вершин в графе задачи и графе системы.

Общий алгоритм:

Поиск первого приближения. В разработанном прототипе используется отображение i-й вершины P_i графа кластера на j-ю вершину V_j графа системы.

Итеративная процедура улучшения решения:

2.1. Перестановка отображения двух вершин. Смена отображения $P_i \to V_j, P_k \to V_l$, на отображение $P_i \to V_l, P_k \to V_j$.

2.2. Отображение на новую вершину. Смена отображения $P_i \to P_j$ на $P_i \to P_k$.

Вычислительная сложность алгоритма $O(IE(M+N))$, где I – число итераций, E – число ребер в графе задачи M – число вершин в графе задачи, N – число вершин в графе системы.

В общем случае оценить сходимость алгоритма к точному решению сложно. Предположим, что все ребра в графе задачи 2 типов и имеют кратные веса a и $a \cdot k$, все ребра в графе системы тоже 2 типов с весами b и $b \cdot r$. В этом случае верхняя граница сходимости $Mkr/(r-1)$, то есть сходимость алгоритма для большинства случаев быстрее этого значения.

3. Описание синтетического теста

Синтетический тест моделирует обмен данными различного объема и с различной интенсивностью в итеративной процедуре, выполняющейся в MPI программе. Тест представляет собой наиболее оптимизируемый частный случай входных данных для алгоритма оптимизации. В тесте используется обмен данными двух типов: обмен внутри группы тесно взаимодействующих процессов с интенсивным взаимодействием и обмен между группами процессов.

Параметры синтетического теста: N_{bl} – число групп процессов, N_{perbl} – число процессов в группе, D_{bl} – объем данных, пересылаемых между группами, D – объем данных, пересылаемых внутри группы, N_{skip} – число пропускаемых итераций.

В каждой группе каждый процесс взаимодействует с каждым процессом группы (полно связанный подграф). Взаимодействие между группами включает только обмен данными между последним процессом группы i и первым процессом группы с номером $i+1$.Например, для значений параметров $N_{bl} = 2$, $N_{perbl} = 6$ взаимодействие между 11-м процессом, входящим в первую группу и первым процессом, входящим во вторую группу, показано тонкими ребрами на графе задачи (рис. 3).

Взаимодействие между группами происходит не на каждой итерации, поэтому задается пропуск итераций. Номера процессов перемешаны случайным образом (с постоянным параметром генератора случайных чисел для воспроизводимости коммуникационного взаимодействия).

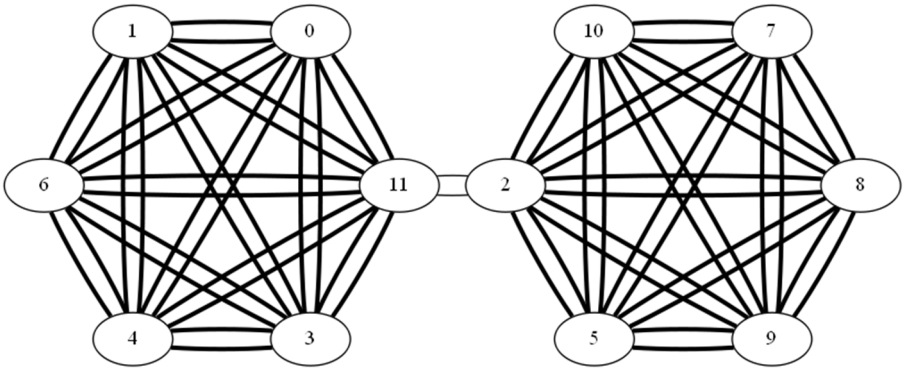


Рис. 3. Пример графа для 2 блоков с 6 процессами в каждом

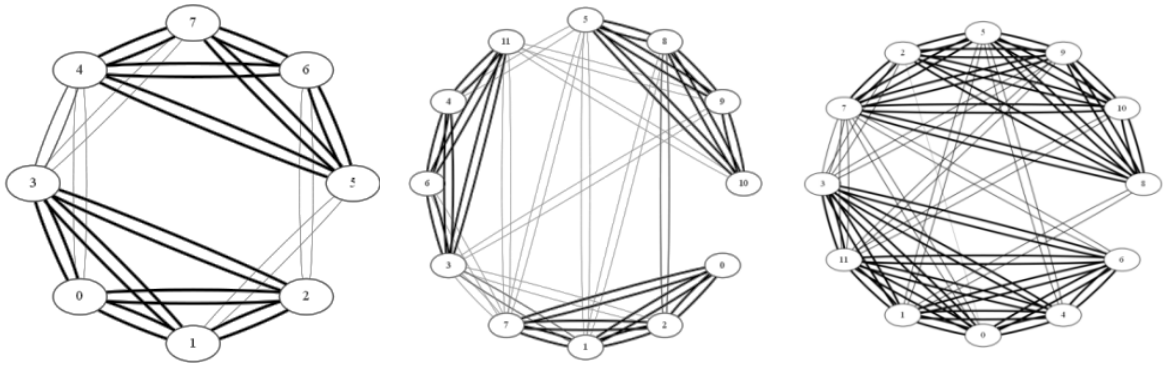
Смещения и длины в линейном массиве с данными строятся на каждой итерации для операций MPI_Alltoallv коллективного обмена MPI-библиотеки. Эти массивы дополняются на каждой $1 + N_{skip}$ итерации данными для обмена между группами. Например, в случае $N_{skip} = 2$ массивы для графа задачи, показанном на рис. 3. имеют вид, представленный в таблице 1.

Таблица 1. Смещения и длины для блоков обмениваемых данных MPI-процессов

Процесс назначения		0	1	2	3	4	5	6	7	8	9	10	11
итер. 3k, 1+3k, процесс 2	смещение	0	0	0	0	0	0	D	D	2D	3D	4D	5D
	длина	0	0	0	0	0	D	0	D	D	D	D	0
итер. 3k, 1+3k, процесс 11	смещение	0	D	2D	2D	3D	4D	4D	5D	5D	5D	5D	5D
	длина	D	D	0	D	D	0	D	0	0	0	0	0
итер. 2+3k, процесс 2	смещение	0	0	0	0	0	0	D	D	2D	3D	4D	5D
	длина	0	0	0	0	0	D	0	D	D	D	D	D_{bl}
итер. 2+3k, процесс 11	смещение	0	D	2D	2D+ D_{bl}	3D+ D_{bl}	4D+ D_{bl}	4D+ D_{bl}	5D+ D_{bl}	5D+ D_{bl}	5D+ D_{bl}	5D+ D_{bl}	5D+ D_{bl}
	длина	D	D	D_{bl}	D	D	0	D	0	0	0	0	0

В таблице показано, что массив смещений и длин обмениваемых блоков для процессов, участвующих в обмене между группами (процессы 2 и 11), каждую итерацию с номером 0+3k и 1+3k содержит 5 обмениваемых блоков длиной D, а каждую итерацию с номером 2+3k содержит дополнительный блок обмена между группами длиной D_{bl}. У остальных процессов массив длин и смещений не меняется.

Пример графов задач, полученных с помощью запусков синтетического теста с различными параметрами показан на рис. 4. Тонкими линиями показаны системные взаимодействия процессов при синхронизации, полученные при сборе статистики.



а б в

Рис. 4. Пример графов задачи, построенных по синтетическим тестам: а) 2 блока с 4 процессами на блок, б) 3 блока с 4 процессами на блок, в) 2 блока с 6 процессами на блок

Таким образом, с помощью синтетического теста моделируется неоднородное взаимодействие заданного количества групп параллельных процессов с возможностью менять объем данных для обмена и интенсивность обмена данными между группами. При задании числа процессов, соответствующего числу ядер на один узел кластера и перенумерации процессов случайным образом, собирается статистика обмена данными, которая подается на вход алгоритму оптимизации.

4. Результаты тестирования

4.1. Запуск тестов

Запуск тестов проводился с использованием программы запуска Intel® MPI Library [6]. Передача набора параметров, необходимых для назначения процессов на вычислительные ядра, реализована с помощью конфигурационного файла, передаваемого по ключу командной строки --configfile. Строка запуска имеет вид

mpirun --configfile config.txt.

Разработан инструментарий для генерации файла конфигурации. В качестве входного файла используется список хостов кластера. Первый шаг – запуск непродолжительного теста для сбора статистики коммуникаций MPI-процессов. Входные данные для генерации конфигурационного файла (config.txt) – это список хостов с именами узлов машин на кластере и числом ядер на каждом узле (hostlist.txt).

Например, для кластера из 4 узлов с четырехъядерными процессорами содержимое файла hostlist.txt имеет вид в листинге на рис. 5:

```
line 01: node1 4
line 02: node2 4
line 03: node3 4
line 04: node4 4
```

Рис. 5. Пример файла hostlist.txt для четырех узлов

Здесь и в остальных листингах начало строки обозначено как line xx, xx –номер строки. Генерируемый файл конфигурации config.txt для сбора статистики имеет вид, приведенный в листинге на рис. 6:

```
line 01: -env I_MPI_STATS 4 -env I_MPI_STATS_FILE
/path/to/stat.txt -n 4 -host node1 /path/to/application
line 02: -env I_MPI_STATS 4 -env I_MPI_STATS_FILE
/path/to/stat.txt -n 4 -host node2 /path/to/application
line 03: -env I_MPI_STATS 4 -env I_MPI_STATS_FILE
/path/to/stat.txt -n 4 -host node3 /path/to/application
line 04: -env I_MPI_STATS 4 -env I_MPI_STATS_FILE
/path/to/stat.txt -n 4 -host node 4 /path/to/application
```

Рис. 6. Пример файла config.txt для сбора статистики на четырех узлах

Запуск MPI программы с использованием файла конфигурации config.txt указанного формата приводит к формированию средствами библиотеки Intel® MPI Library файла статистики обменов stats.txt. Программе оптимизации подаются на вход stats.txt и hostlist.txt, на выходе генерируется config.txt с оптимизированным отображением процессов на узлы кластера. Для рассматриваемого примера содержимое config.txt приведено в листинге на рис. 7:

```
line 01:-n 2 -host node1 /path/to/application
line 02:-n 2 -host node4 /path/to/application
line 03:-n 1 -host node2 /path/to/application
line 04:-n 2 -host node1 /path/to/application
line 05:-n 3 -host node2 /path/to/application
line 06:-n 4 -host node3 /path/to/application
line 07:-n 2 -host node4 /path/to/application
```

Рис. 7. Пример файла config.txt для четырех узлов после оптимизации

Оптимизированный файл конфигурации используется для запуска длительного теста. Методология тестирования состоит в замере времени выполнения длительного теста до и после оптимизации размещения процессов на кластере. Показатель эффективности – ускорение, вычисляемое как $S = T/T_{opt}$, где T – время работы теста до оптимизации, T_{opt} – после оптимизации.

Для запуска тестов использован суперкомпьютер «Торнадо ЮУрГУ» [7]. Конфигурация выделенных узлов: двухсокетные узлы с процессорами Intel® Xeon® X5680 (2 процессора по 6 ядер), Intel® Xeon Phi™ SE10X (61 ядро по 1.1 GHz), 2GB RAM, InfiniBand QDR, топология сети – толстое дерево. При тестировании были поставлены задачи исследования синтетического теста до и после оптимизации отображения процессов на ядра двух сопроцессоров и на центральных процессорах кластера при различной размерности блоков. На центральных процессорах кластера также было необходимо провести запуск масштабируемого на большое число узлов вычислительного приложения с неоднородным взаимодействием процессов, в качестве которого был выбран CORAL QBox[8].

4.2. Синтетический тест для двух узлов кластера с сопроцессорами

Конфигурация для тестирования состоит из двух узлов с установленными сопроцессорами Intel® Xeon Phi™, содержащими по 60 ядер, доступных для вычислений. Такая конфигурация была выбрана для оценки влияния работы MPI тестов, характеризующихся слабой связанностью по данным групп параллельных процессов с интенсивным обменом, что имеет место, например, при моделировании связанных физических задач. Данный тест был необходим для оценки влияния сетевого взаимодействия в случае интенсивного обмена групп с большим числом параллельных процессов, поэтому в графе системы было только 2 типа весов ребер: связи между ядрами и сетевой интерконнект, взаимодействие ядер сопроцессора принималось приближенно равнозначным. Результаты тестирования приведены в таблице 2. Параметры теста: D = 60Мб, D_{bl} = 60Кб, $N_{skip} = 0$.

Таблица 2. Результаты синтетического теста для 2 узлов с сопроцессорами

N_{bl}	N_{perbl}	T, сек.	T_{opt}, сек.	S
2	60	660,847	84,125	7,855
3	40	440,512	194,514	2,264
4	30	320,938	153,668	2,088
5	24	261,291	86,774	3,011
6	20	221,371	35,210	6,287
8	15	165,581	34,700	4,771
15	8	93,015	39,100	2,378
20	6	68,347	30,007	2,277

Каждый тестовый запуск проводился в Native режиме (код выполняется только на сопроцессорах) и использовал все вычислительные ядра сопроцессора: $N_{bl} \cdot N_{perbl} = 120$. Содержимое сгенерированного конфигурационного файла для сбора статистики приведено в листинге на рис. 8.

```
line 01:-env I_MPI_STATS 4 -env I_MPI_STATS_FILE
/path/to/stat.txt -n 60 -host mic0 /path/to/application
line 02:-env I_MPI_STATS 4 -env I_MPI_STATS_FILE
/path/to/stat.txt -n 60 -host mic1 /path/to/application
```

Рис. 8. Пример файла config.txt для сбора статистики на 2 сопроцессорах

Фрагмент файла конфигурации config.txt после оптимизации на основе собранной статистики приведен в листинге на рис. 9.

```
line 01:-n 5 -host mic0 /home/test/MPICommunication 20 6
line 02:-n 1 -host mic1 /home/test/MPICommunication 20 6
line 03:-n 4 -host mic0 /home/test/MPICommunication 20 6
line 04:-n 2 -host mic1 /home/test/MPICommunication 20 6
line 05:-n 1 -host mic0 /home/test/MPICommunication 20 6
line 06:-n 3 -host mic1 /home/test/MPICommunication 20 6
. . .
```

Рис. 9. Фрагмент файла config.txt для 2 сопроцессоров после оптимизации

Файл для сбора статистики состоит из 2 строк (по числу узлов), а сгенерированный после оптимизации конфигурационный файл для запуска содержит 55 строк, так как номера процессов изначально перемешаны случайным образом.

В результате отображения параллельных процессов на ядра получено ускорение выполнения тестов в несколько раз, что свидетельствует о существенной зависимости расположения параллельных процессов на вычислительных ядрах и целесообразности использования алгоритма отображения.

4.3. Синтетический тест для 10 узлов с InfiniBand интерконнектом

В качестве тестового стенда использовался кластер с 10 двухсокетными узлами и 6-ядерными процессорами Intel® Xeon®. Тестирование проводилось для фиксированного объема данных, но с разным числом итераций (N_{iter}) в соответствии с параметрами, приведенными в таблице 3. Для обмена внутри групп общий объем передаваемых данных между парой процессов 800 Мб, для обмена между группами – 800 Кб. В тесте сгенерированы 10 групп по 12 процессов в каждой (по числу ядер каждого узла кластера). Фрагмент графа задачи приведен на рис. 10.

Таблица 3. Результаты синтетического теста для 10 узлов кластера

N_{iter}	D_{bl}	D	$T, sec.$	$T_{opt}, sec.$	S
100	8 Kb	8 Mb	1755	880	1,99
1000	0,8 Kb	0,8 Mb	23,21	9,56	2,42
10000	0,08 Kb	0,08 Mb	5,96	1,01	5,9

Результаты свидетельствуют о том, что более частые обмены небольших порций данных с помощью наиболее эффективных в данном случае операций коллективного обмена библиотеки MPI оптимизируются лучше, чем менее интенсивные обмены больших порций данных. Частые обмены небольших порций характерны для вычислительных приложений, использующих, например, декомпозицию расчетной области или параллельную реализацию явных методов интегрирования систем дифференциальных уравнений. Для некоторых задач при неполной загрузке узлов (тест с менее чем 12 процессами на узел) выполнение параллельной программы может оказаться быстрее при использовании InfiniBand, чем при использовании общей памяти, например, для описанного в источнике [9] случая. Для рассматриваемой задачи эксперименты показали проявление такого же эффекта: в случае неполной загрузки узлов (по 4-8 ядер из 12) тест срабатывал медленнее при переназначении наиболее интенсивного обмена на общую память, чем при переназначении интенсивного обмена на InfiniBand. Разница времени выполнении составляла не более 5%.

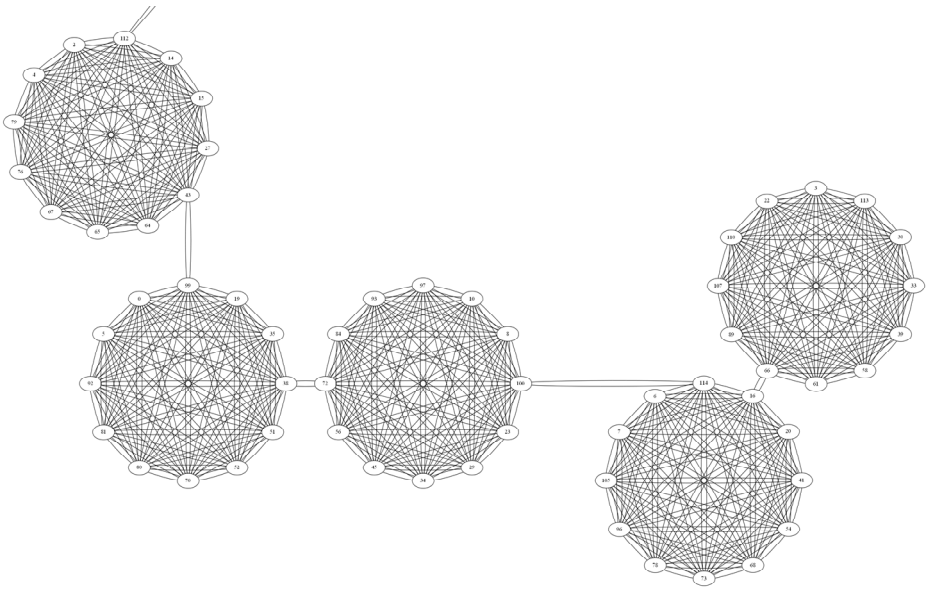


Рис. 10. Фрагмент графа задачи для синтетического теста (5 групп из 10 по 12 процессов)

4.4. Результаты тестирования на удаленном кластере открытого пакета для моделирования молекулярной динамики CORAL QBox

CORAL QBox – это пакет моделирования молекулярной динамики для определения свойств материала, использующий теорию функционала плотности. Метод имеет вычислительную сложность $O(N^3)$, где N – общее число валентных электронов в системе.

Тесты собраны и запущены для различного числа узлов удаленного кластера вплоть до 400. В тестах задействовано разное число узлов (N_{nodes}), при максимальной загрузке узлов число параллельных процессов составляет $12N_{nodes}$.

Основной параметр, влияющий на производительность QBOX – это параметр максимального числа строк в 2D сетке параллельных процессов (nrowmax). Количество строк и столбцов в этой сетке подбирается так, что nrowmax уменьшается, пока число параллельных процессов (N) не будет делиться на nrowmax, после чего число столбцов принимается равным N/nrowmax. Эксперименты показали, что максимальная эффективность имеет место при nrowmax = N_{nodes}, соответственно число столбцов составляет при этом 12. Число атомов при генерации входных данных было выбрано как максимальное число атомов, при котором задача помещается в оперативную память, значение которой ограничено двумя гигабайтами на узел.

Граф задачи для пакета QBox имеет общий вид из-за неоднородного обмена данными. По статистике, собранной при запуске на двух узлах с nrowmax = 2, построен граф задачи, на кото-

ром можно выделить 2 группы по 12 процессов с тесным взаимодействием (рис.11), причем порядок соответствующих вершин графа соответствует номерам ядер процессоров. Каждой подгруппе соответствует полносвязный подграф графа задачи, в то время, как между подграфами набор связей неполный и их суммарный вес, соответствующий объему переданных данных сопоставим по величине с суммарным весом ребер, инцидентных ребрам одной вершины в подгруппе (для каждой вершины средний объем переданных данных 185Мб, а общий объем переданных данных между подграфами 421 Мб).

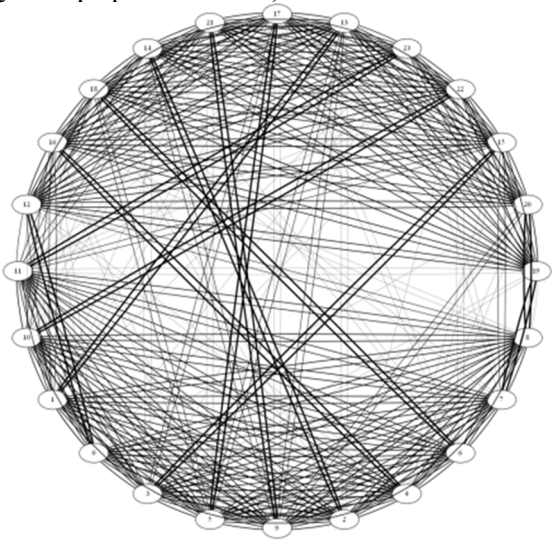


Рис. 11. Две группы из 12 процессов в графе задачи для пакета QBox

Тесты в пакет QBox были подобраны для наилучшего времени выполнения при размерностях сеток 25x12, 50x12, 100x12, 150x12, 200x12, 400x12 для 25-400 узлов соответственно (число процессов и ядер 300-4800). Коэффициент для взвешивания ребер графа системы составил 2:1 (общая память по отношению к InfiniBand). Результаты тестирования показаны в таблице 4.

Таблица 4. Результаты тестирования QBox

Число узлов	Размерность сетки	T, сек.	T_{opt}, сек.	S
25	25x12	773,5	660,5	1,171083
50	50x12	398,3	336,4	1,184007
100	100x12	216,8	180,2	1,203108
150	150x12	180,5	155,1	1,163765
200	200x12	172,4	148,3	1,162508
400	400x12	170	145,2	1,170799

На основании собранных данных видно, что оптимизация обменов данными дает практически постоянное улучшение производительности на 17–20%. Максимум приходится на 100 узлов (1200 параллельных процессов). С этого же числа узлов падает масштабируемость задачи, так как размерность задачи ограничена оперативной памятью и задачу большей размерности для сохранения масштабируемости рассчитать при такой конфигурации кластера нельзя. Время выполнения рассматриваемой задачи в пакете QBox после 100 узлов сходится к постоянному значению (рис. 12). Тесты на большем числе процессов (до 4800 процессов на 400 узлах) показывают примерно одинаковый результат как по времени выполнения, так и по ускорению за счет оптимизации отображения процессов. В рассматриваемом случае сильного масштабирования время пересылок на несколько порядков меньше, по сравнению со временем расчета одной итерации, поэтому, экстраполируя данные на большее число процессов, можно сделать предположение о том, что время выполнения MPI-программы будет сохраняться постоянным, так же, как и эффект от оптимизации, для достаточно большого числа процессов.

При существенном увеличении числа процессов время расчета одной итерации будет приближаться ко временем выполнения пересылки данных, что приведет к увеличению влияния пересылок. При этом время выполнения параллельной программы будет расти, а эффект от оптимизации, предположительно, будет усиливаться из-за более существенного влияния времени пересылок на общее время выполнения. Последний сценарий не используется на практике, так как предполагает неэффективное использование ресурсов.

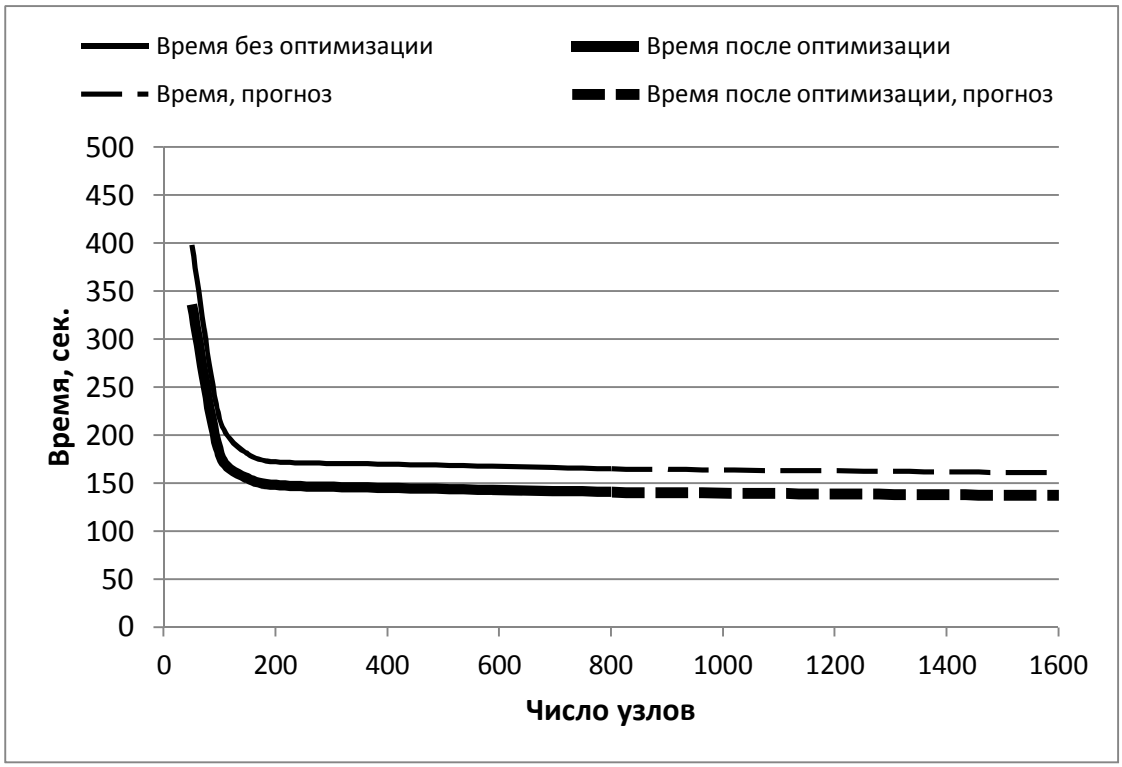


Рис. 12. Время выполнения QBox до и после оптимизации

Рассмотренный вычислительный пакет CORAL QBox хорошо масштабируется до 100 узлов. На большем числе узлов эффективность параллельного расчета снижается, так как размерность задачи недостаточно большая и время обмена данными сопоставимо с расчетным временем. Для задач с большей размерностью пакет масштабируется лучше (анализ проведен в работе [8]) возможно получение более хороших результатов по ускорению за счет оптимизации обмена данными.

5. Заключение

Предложенный подход обеспечивает улучшение производительности параллельного расчета в случае, если граф задачи MPI-приложения имеет неоднородное распределение весов ребер и выраженные подграфы, вершины в которых сильно связаны между собой и количество этих вершин близко к количеству ядер на одном узле кластера.

Положительный эффект ускорения в 2-7 раз получен для синтетического теста при тестировании на узлах с сопроцессорами Intel® Xeon Phi™ и процессорами Intel® Xeon®. Основные факторы, влияющие на эффективность метода – это неоднородность взаимодействия процессов, частота обмена данными и объемы пересылаемых данных.

Положительный эффект ускорения открытого пакета CORAL QBox на 17–20% получен предложенным методом для числа процессов до 4800.

Предложенный метод не дает ощутимого эффекта в случае плохо масштабируемого приложения, задач с равномерным обменом между параллельными процессами и задач с малой интенсивностью обменов.

Экспериментально установлено, что использование подхода при неполной загрузке узлов кластера дает отрицательный эффект из-за сопоставимой в этом случае скорости передачи данных через интерконнект между узлами кластера и через общую память на каждом узле.

Метод отображения задачи можно усовершенствовать, используя физические параметры интерконнекта (пропускную способность и задержки канала связи), а также учитывая полную топологию кластера.

Список литературы

1. Копысов С.П., Новиков А.К., Тонков Л.Е., Береснев Д.В. Методы привязки параллельных процессов и потоков к многоядерным узлам вычислительных систем // Вестн. Удмуртск. ун-та. Матем. Мех. Компьют. науки, 2010, вып. 1, С. 123–132

2. Курносов М.Г. Назначение ветвей параллельной программы на процессорные ядра распределенной вычислительной системы // Материалы Межд. научно-технической конференции “Многопроцессорные вычислительные и управляющие системы”, пос. Дивноморское, Геленджик, Россия, 2007, С. 227–231.

3. Karlsson, C., Davies, T., Chen, Z. Optimizing Process-to-Core Mappings for Application Level Multi-dimensional MPI Communications// Cluster Computing (CLUSTER), 2012 IEEE International Conf. Proceedings, 24-28 Sept. 2012, P. 486-494.

4. Zhang J., Zhai J., Chen W., Zheng W. Process Mapping for MPI Collective Communications //Lecture Notes in Computer Science Volume 5704, 2009, pp 81-92

5. Chen H., Chen W., Huang J., Robert B., Kuhn H. MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters// ICS '06 Proceedings of the 20th annual international conference on Supercomputing, June 2006, P. 353 - 360

6. Intel® MPI Library Reference Manual // http://software.intel.com/sites/products/documentation/hpc/ics/impi/41/lin/Reference_Manual/index.htm

7. Larsson P. Shared Memory Communication vs Infiniband // http://www.nsc.liu.se/~pla/blog/2013/09/12/smp-vs-infiniband

8. Gygi F., Yates R.K., Lorenz J., Draeger E.W., Franchetti F., Ueberhuber C., Supinski B., Gunnels S., and Sexton J. Large-Scale First-Principles Molecular Dynamics simulations on the BlueGene/L Platform using the Qbox code// Proceedings of the ACM/IEEE SC 2005 Conference, 12-18 Nov. 2005, P. 24.

9. Суперкомпьютер «Торнадо ЮУрГУ» // http://supercomputer.susu.ac.ru/computers/tornado/