УДК 519.673
DOI: 10.15827/0236-235X.128.573-580
Дата подачи статьи: 21.08.19
2019. Т. 32. № 4. С. 573–580

Исследование оптимального количества процессорных ядер для алгоритма многократной маркировки перколяционных кластеров на суперкомпьютерных вычислительных системах

С.Ю. Лапшина [1], *старший научный сотрудник, lapshina@jscc.ru*
А.Н. Сотников [1], *д.ф.-м.н., профессор, asotnikov@jscc.ru*
В.Е. Логинова [1], *ведущий инженер-программист, vl@jscc.ru*
К.Ю. Юдинцев [1], *научный сотрудник, climenty@jscc.ru*

[1] *Межведомственный суперкомпьютерный центр РАН – филиал ФНЦ НИИСИ РАН, г. Москва, 119334, Россия*

Статья посвящена выбору оптимального количества запрашиваемых процессорных ядер для запуска алгоритма многократной маркировки перколяционных кластеров. Работа выполнена в ходе проведения имитационных экспериментов задачи мультиагентного моделирования процессов распространения массовых эпидемий на современных суперкомпьютерных системах, установленных в Межведомственном суперкомпьютерном центре РАН.

Алгоритм может быть использован в любой области в качестве инструмента дифференцирования кластеров решетки большого размера, так как ему на вход подаются данные в формате, не зависящем от приложения. В МСЦ РАН этот инструмент использовался для изучения задачи распространения эпидемий, для чего была разработана соответствующая мультиагентная модель.

В модели рассматривается абстрактное заболевание, передаваемое контактным путем. В ходе моделирования определяется пороговое значение вероятности инфицирования (то есть сама вероятность инфицирования является изменяемым параметром), при котором возникает эффект перколяции на решетке распространения заболевания. Если это значение близко к индексу контагиозности конкретного заболевания, то следует ожидать распространения эпидемии в планетарном масштабе.

В процессе имитационных экспериментов применялся усовершенствованный для многопроцессорной системы вариант алгоритма многократной маркировки перколяционных кластеров Хошена–Копельмана, связанный с механизмом линковки меток, который также может быть использован в любой области в качестве инструмента дифференцирования кластеров решетки большого размера.

В статье дана оценка времени выполнения алгоритма многократной маркировки перколяционных кластеров Хошена–Копельмана при различных значениях входных параметров на четырех основных высокопроизводительных вычислительных системах, установленных в Межведомственном суперкомпьютерном центре РАН: суперкомпьютерах МВС-10П МП2 KNL, МВС-10П ОП, МВС 10П Торнадо, МВС-100К.

Ключевые слова: *мультиагентное моделирование, перколяционный кластер, механизм линковки меток, высокопроизводительные вычислительные системы, процессорные ядра.*

При моделировании процессов распространения массовых эпидемий и пандемий часто незначительные изменения значений одного или нескольких параметров (например, вероятности инфицирования отдельных представителей) могут привести к скачкообразному изменению поведения всей популяции (болезнь из локальной и неопасной переходит в стадию широкомасштабной пандемии) [1]. Одним из способов исследования подобных эффектов является изучение формирования и роста *перколяционных кластеров.*

Исследования свойств перколяционных кластеров целесообразно сопровождать активным использованием разнообразных технологий компьютерного моделирования, в том числе и технологии мультиагентной имитации.

Принципиальной особенностью мультиагентной имитации по сравнению с другими технологиями (агрегатной, дискретно-событийной и т.д.) является возможность выявления и регистрации свойств поведения сложной системы под воздействием сугубо индивидуалистического характера поведения ее представителей [2]. Уточнение полученных результатов с целью совершенствования исходной модели порой требует проведения большого количества ресурсоемких имитационных экс-

периментов (главным образом на многопроцессорных вычислительных архитектурах), использования специализированных алгоритмов распараллеливания и подбора оптимальных значений входных параметров [3–5].

В мультиагентной модели распространения эпидемий, разработанной в МСЦ РАН как часть программно-аппаратного комплекса BIOCLUST [6], применяется многопроцессорный алгоритм *многократной маркировки перколяционных кластеров* (ММПК), основанный на предложенном в 1976 г. Хошеном и Копельманом [7, 8] алгоритме маркировки связных подграфов (кластеров) некоторого графа.

Алгоритм ММПК Хошена–Копельмана

Алгоритм ММПК Хошена–Копельмана – однопроходный алгоритм, позволяющий выделять связные подграфы (кластеры) некоторого случайного графа. Идея его в том, что всем занятым узлам решетки приписываются различные кластерные метки, принадлежность узла к тому или иному кластеру является глобальным свойством и может быть определена только после просмотра всей решетки.

Узлы решетки нумеруются. Номер узла – начальное значение метки этого узла. При последовательном обходе решетки для каждого узла рассматриваются связанные с ним соседние узлы. Каждой группе (текущий узел, соседние с ним узлы) ставится минимальная метка этой группы. На каждом шаге все производимые замены меток должны отражаться на всех узлах решетки, то есть, если на некотором шаге метка одного узла была заменена, то нужно заменить и все остальные метки узлов, равные данной. Далее приведен псевдокод алгоритма Хошена–Копельмана:

```
макс_метка = 0;
for x from 0 to n_колонок {
 for y from 0 to n_рядов {
 if A[x,y] != 0 then
 слева = A[x-1,y];
 сверху = A[x,y-1];
 if (слева == 0) and (сверху == 0) then {
  макс_метка = макс_метка + 1;
  A_с_метками[x,y] = макс_метка;
 }
 else {
 if (слева != 0) {
  if (справа != 0)
   {
    объединить (слева, сверху);
   A_с_метками [x, y] = найти(сверху);
   }
 } else
 A_с_метками [x, y] = найти(справа);
 }
 }
 },
```

где A[x, y] – исходный массив; A_с_метками [x, y] – конечный массив; объединить(x, y) – команда присвоения узлу y метки узла x; найти(x) – команда нахождения ближайшей ячейки того же кластера (ячейки с той же меткой), что и x.

В результате работы алгоритма ММПК все узлы решетки будут разделены по их принадлежности тому или иному кластеру (принадлежность определяется меткой – все узлы с одинаковыми метками принадлежат одному и

1	0	0	1	1	1	1	1	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1	1	0	0	0	1	0	1
0	1	0	0	1	0	1	1	1	0	0	0	1	0	1
1	1	0	0	1	0	1	1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	0	0	0	1	1	1	1	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	1	0	0
0	0	1	1	1	1	0	0	0	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1	0	0	0	0	0	1
0	1	1	1	1	0	1	1	1	0	0	0	1	0	1
1	1	1	1	1	0	1	1	1	1	1	1	1	0	1
1	1	1	1	1	0	1	1	1	1	1	1	1	0	1
1	0	1	1	1	0	0	0	0	1	1	1	1	0	0

1	0	0	2	2	2	2	2	2	2	0	2	2	2	0
0	4	0	0	2	0	2	2	2	0	0	0	2	0	5
0	4	0	0	2	0	2	2	2	0	0	0	2	0	5
4	4	0	0	2	0	2	2	2	2	2	2	2	0	5
4	4	0	0	2	0	2	2	2	2	2	2	2	0	5
4	0	0	0	2	0	0	0	0	2	2	2	2	0	0
0	0	0	0	2	0	0	0	0	0	0	0	2	0	0
0	0	0	0	2	0	0	0	0	0	0	0	2	0	0
0	0	2	2	2	0	0	0	0	0	0	0	2	0	0
0	0	2	2	2	2	0	0	0	0	0	0	2	2	0
0	2	2	2	0	0	6	6	6	0	0	0	0	0	7
0	2	2	2	2	0	6	6	6	0	0	0	6	0	7
2	2	2	2	2	0	6	6	6	6	6	6	6	0	7
2	2	2	2	2	0	6	6	6	6	6	6	6	0	7
2	0	2	2	2	0	0	0	0	6	6	6	6	0	0

Рис. 1. Исходная и конечная решетки при работе алгоритма Хошена–Копельмана

Fig. 1. Initial and final lattices during the operation of the Hoshen–Kopelman algorithm

тому же графу). На рисунке 1 приведен пример работы алгоритма Хошена–Копельмана.

Параллельный вариант алгоритма ММПК

При наличии у компьютера большого количества процессоров выгоднее использовать параллельный алгоритм ММПК [9]. Каждому процессору назначается группа узлов решетки. Параллельный вариант алгоритма совпадает с обычным алгоритмом ММПК за одним исключением: вместо прохода по всей решетке каждый процессор выполняет действия алгоритма ММПК только на назначенной ему группе узлов с последующим обменом информацией между процессорами. Алгоритм завершается тогда, когда на очередном шаге после обмена информацией метки всех групп узлов перестают изменяться.

Работу алгоритма можно разделить на три этапа.

1. *Инициализация*. Первый процесс загружает решетку в оперативную память из файла, преобразует ее по входным параметрам, распределяет узлы в группы по процессам и отсылает им. Остальные процессы ждут получения своей группы узлов. Получив такую группу, процессы выделяют из них подгруппу внешне связанных узлов, то есть узлов, связанных с узлами из групп других процессов. Для каждого узла своей группы задаются начальные значения меток в соответствии с абсолютным (в рамках всей решетки) номером узла. Каждый процесс создает группу внешних узлов, связанных с узлами своей группы, и инициализирует их метками связанных узлов своей группы.

2. *Работа собственно алгоритма*. Каждый процесс запускает алгоритм ММПК на своей группе узлов.

3. *Обмен информацией*. Происходит в несколько шагов до тех пор, пока после очередного обмена метки узлов всех процессов не перестанут изменяться. Обмен информацией можно разделить на три этапа. На первом этапе каждый процесс посылает значения меток узлов из подгруппы внешне связанных узлов тем процессам, с которыми эти узлы связаны. На втором этапе каждый процесс принимает значения меток от процессов, имеющих узлы, связанные с узлами данного. Эти значения присваиваются меткам узлов из группы внешних узлов. Если хотя бы одна из меток узлов внешней группы была изменена, процесс должен повторить обмен информацией. Все метки узлов своей группы, равные замененным меткам внешней группы, также должны быть заменены. На третьем этапе выполняется отправка сообщения первому процессу о том, должен ли данный процесс повторить обмен информацией с другими процессами. Первый процесс получает от всех процессов подобную информацию. Если все процессы не нуждаются в повторном обмене, первый процесс рассылает всем остальным сигнал о завершении работы всего алгоритма и начинает процедуру сбора данных по меткам их узлов. Если хотя бы один процесс прислал сообщение, что ему необходимо продолжить обмен, всем процессам придется повторить процедуру обмена информацией.

Важным моментом в работе алгоритма является создание механизма линковки меток. Во время работы алгоритма ММПК при последовательном прохождении узлов на каждом из них при различии меток данного узла и его соседей приходится заменять метки этих узлов на минимальную среди них. Все замененные таким образом метки должны быть также заменены среди всех остальных меток решетки. То есть, если, например, данный узел имеет метку F, а его сосед метку Y, то ему и его соседу присваивается метка min(F,Y), к тому же надо заменить метки всех узлов решетки, имеющие значение F или Y, на min(F,Y). Таким образом, получается, что время работы алгоритма зависит от размера решетки N как $O(N^2)$. Механизм линковки меток позволяет добиться скорости O(N). Смысл этого механизма в следующем (рис. 2).

Массив ссылок		Массив ссылок		Массив ссылок		Массив ссылок		Метки
1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3
4	4	4	4	4	1	4	1	4
5	5	5	5	5	5	5	5	5
6	6	6	2	6	2	6	2	6
7	7	7	4	7	4	7	1	7
8	8	8	8	8	6	8	2	8
		Шаг 1		Шаг 2		Шаг 3		Шаг 4

Рис. 2. Механизм линковки меток

Fig. 2. Parallel cluster multiple labeling technique

Если пронумеровать все узлы решетки, то им можно поставить в соответствие массив меток и массив ссылок. Массивы меток и ссылок инициализируются номерами узлов. Массив

ссылок – массив адресов в рамках массива. Например, если необходимо узнать метку узла под номером 7, обратимся к массиву ссылок к элементу 7. Значение ссылки элемента 7 есть 7. Если значение ссылки в массиве ссылок равно номеру элемента этого массива, то по ссылке не надо никуда переходить, остается обратиться к тому же самому номеру элемента в массиве меток для получения значения метки, то есть обратиться к элементу 7 массива ссылок и получить значение метки 7.

В процессе изменения меток в алгоритме ММПК изменяются не сами метки, а значения элементов массива ссылок. Например, если узел 2 связан с узлом 6 (рис. 2, шаг 2), то узлу 6 в массиве ссылок проставляется адрес на узел 2. Далее, если узел 1 связан с узлом 4, то узлу 4 проставляется адрес на 1 (рис. 2, шаг 3). Аналогично, если узел 8 связан с узлом 6, то узлу 8 ставится адрес 6. После этих операций для получения метки узла 8 надо выполнить описанные выше действия: обращаемся к массиву ссылок по номеру 8 – получаем адрес 6, обращаемся к номеру 6 – получаем адрес 2, обращаемся к номеру 2 – получаем адрес 2. Так как адрес совпал с номером элемента, надо обращаться в массив меток под номером 2. Получаем для узла 8 метку, равную 2.

После достаточно длительного процесса перелинковки следует привести массив ссылок к такому виду, при котором количество переходов по адресам не превышает 1.

Схема имитационных экспериментов

Схема имитационных экспериментов, проводимых с помощью многопроцессорных вычислительных систем на основе параллельного алгоритма ММПК [6, 10], состоит из этапов, представленных на рисунке 3.

При помощи алгоритма сбора информации о численности населения (обработчик карт MapManager) были получены данные о городах мира и сохранены в БД в формате «номер города, численность населения, широта, долгота». БД городов содержит информацию о 56 976 городах с общей численностью населения 6 114 628 510 человек.

С помощью алгоритма формирования решетки взаимодействия представителей популяции (построитель графа GridBuilder) была создана исходная решетка (реализация на java) и сохранена в трех файлах: grid, edges1, edges2. Для каждого значения входного изменяемого параметра вероятности заражения при контакте с больным (параметр p = 0,01–…–1,00 с шагом 0,01) исходной решетки в оперативной памяти формируется анализируемая решетка.

Далее проводилась разбивка графа соседних городов на связанные подмножества алгоритмом Хошена–Копельмана (маркировка кластеров Load). Для каждой анализируемой решетки запускался параллельный алгоритм ММПК.

В качестве результата маркировки кластеров были получены массивы кластерных меток (с индексами от 1 до 100).

Важным моментом работы параллельного алгоритма ММПК является правильный подбор количества процессорных ядер, на которых будет обрабатываться исходная решетка.

В ходе работы алгоритма она загружается в оперативную память узла, и, принимая во внимание ее достаточно большой размер, было бы логично распараллелить процесс ее обработки на большое количество частей.

Но, с другой стороны, в ходе работы алгоритма нужно проводить обмен данными между пограничными ячейками частей исходной решетки. Если таких частей будет слишком много, время обмена данными может превысить отведенный на обработку задания лимит.

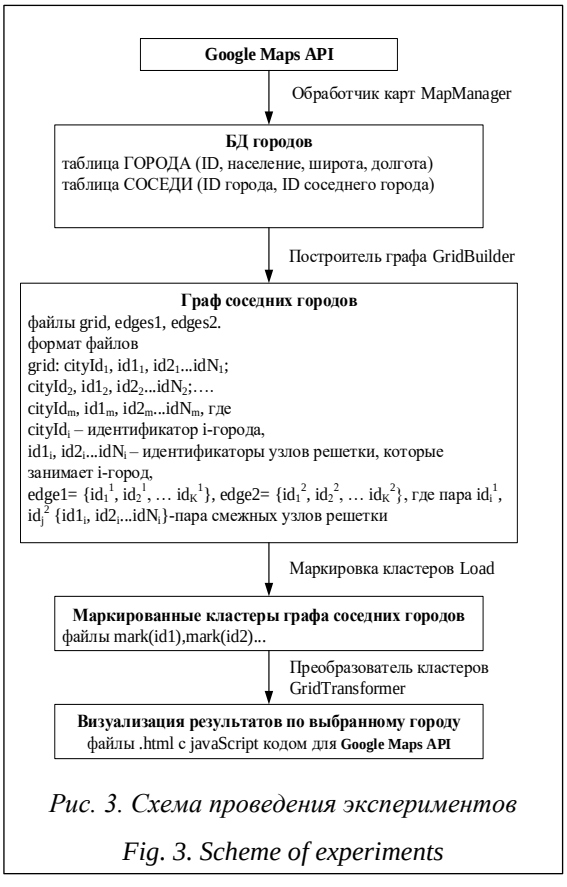

Рис. 3. Схема проведения экспериментов

Fig. 3. Scheme of experiments

Анализ оптимального выбора количества процессорных ядер

Для исследования оптимального количества запрашиваемых процессорных ядер для запуска алгоритма программа маркировки кластеров Load запускалась в МСЦ с входным параметром вероятности p от 0.01 до 1 с шагом в 0.01 при постоянных значениях модельного времени t = 30 дней на 48–304 процессорных ядрах на суперкомпьютерах МВС-10П ОП, МВС-10П МП2, МВС-10П Торнадо, МВС-100К.

МВС-10П ОП предоставляется пользователям Центра в режиме коллективного доступа к трем разделам:

– Haswell (42 вычислительных модуля на базе процессоров Intel Xeon E5-2697 v3, 128 ГБ оперативной памяти на модуль, пиковая производительность модуля – 1.1648 TFLOPS, 1 176 ядер в разделе);

– Broadwell (136 вычислительных модулей на базе процессоров Intel Xeon E5-2697 v4, 128 Гб оперативной памяти на модуль, пиковая производительность модуля – 1.3312 TFLOPS, 4 352 ядра в разделе);

– Skylake (58 вычислительных модулей на базе процессоров Intel Xeon Gold 6154, 192 ГБ оперативной памяти на модуль, пиковая производительность модуля – 3.456 TFLOPS, 2 088 ядер в разделе).

Общим для установок на МВС-10П ОП является использование в качестве коммуникационной среды низколатентной сети Intel Omni-Path.

МВС-10П МП2 KNL – суперкомпьютер из 38 вычислительных модулей на базе процессоров Intel Xeon Phi 7290, 96 ГБ оперативной памяти на модуль, пиковая производительность модуля – 3.456 TFLOPS, 2 736 ядер в системе.

МВС-10П Торнадо – суперкомпьютер из 207 вычислительных модулей, каждый модуль имеет в своем составе два процессора Xeon E5-2690, 64 ГБ оперативной памяти, два сопроцессора Intel Xeon Phi 7110X, пиковая производительность модуля – 371.2 GFLOPS, 3 312 ядер в системе.

МВС-100К – суперкомпьютер из 110 вычислительных модулей на базе процессоров Intel Xeon E5450, 8 ГБ оперативной памяти на модуль, пиковая производительность модуля – 96 GFLOPS, 880 ядер в системе.

На рисунке 4 показан график зависимости времени работы программы Load от количества запрашиваемых процессорных ядер на различных разделах МВС-10П ОП.

На МВС-10П ОП среднее время расчета составило: раздел Haswell – 354 сек., раздел Broadwell – 375 сек., раздел Skylake – 418 сек.

Минимальное время запуска: раздел Haswell – 320 сек. на 128 ядрах, раздел Broadwell – 351 сек. на 208 ядрах, раздел Skylake – 370 сек. на 128 ядрах.

На рисунке 5 показан график зависимости времени работы программы Load от количества запрашиваемых процессорных ядер на МВС-10П МП2 KNL. Среднее время расчета составило 1 199 сек. (почти в три раза больше, чем на любом из разделов МВС-10П ОП), минимальное время запуска – 1 173 сек. на 128 ядрах.

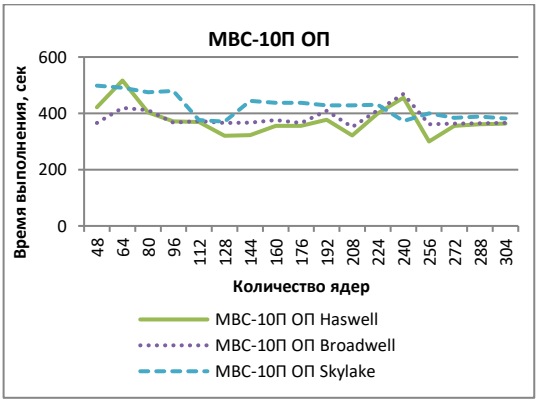

Рис. 4. Проведение расчета на разделах МВС-10П ОП

Fig. 4. Calculation on sections of the MVS-10P supercomputer

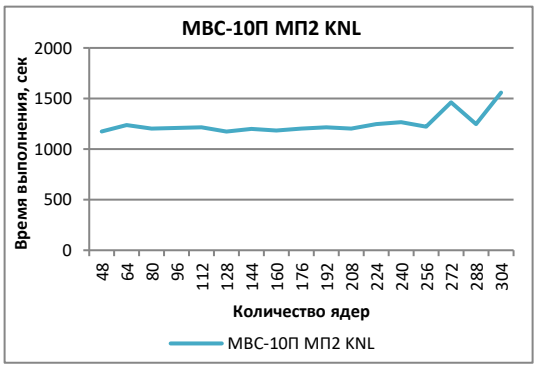

Рис. 5. Проведение расчета на МВС-10П МП2 KNL

Fig. 5. Calculation on the MVS-10P MP2 KNL supercomputer

На рисунке 6 показан график зависимости времени работы программы Load от количества запрашиваемых процессорных ядер на МВС-10П Торнадо. Среднее время расчета со-

ставило 264 сек. (примерно на 25 % меньше, чем на любом из разделов МВС-10П ОП), минимальное время запуска – 234 сек. на 160 ядрах.

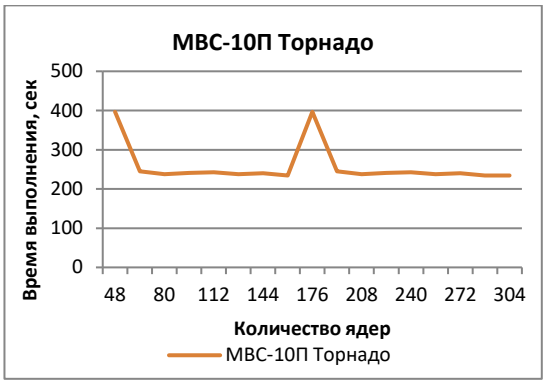

Рис. 6. Проведение расчета на МВС-10П Торнадо

Fig. 6. Calculation on the MVS-10P Tornado supercomputer

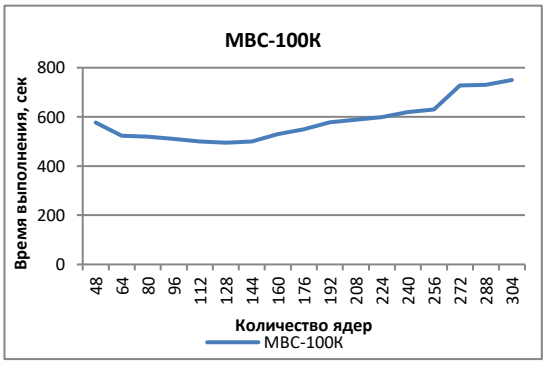

Рис. 7. Проведение расчета на МВС-100К

Fig. 7. Calculation on the MVS-100K

На рисунке 7 показан график зависимости времени работы программы Load от количества запрашиваемых процессорных ядер на МВС-100К. Среднее время расчета составило 572 сек. (примерно на 50 % больше, чем на разделах МВС-10П ОП), минимальное время запуска – 495 сек. на 160 ядрах.

На рисунке 8 показан сводный график зависимости времени работы программы Load от количества запрашиваемых процессорных ядер на основных системах МСЦ РАН. Минимальное время расчета показывает МВС-10П Торнадо. Для большинства суперкомпьютеров минимальное время счета достигается при использовании 128–208 ядер.

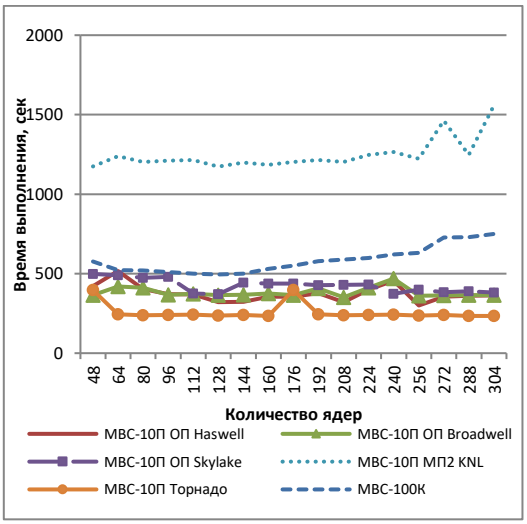

Рис. 8. Сводный график зависимости времени расчета от количества запрашиваемых процессорных ядер

Fig. 8. Summary graph of calculation time dependence on the number of requested processor cores

Расчеты проводились на высокопроизводительных вычислительных системах МВС-10П МП2 KNL, МВС-10П ОП, МВС 10П Торнадо, МВС-100К в Межведомственном суперкомпьютерном центре Российской академии наук.

Работа выполнена в МСЦ РАН в рамках государственного задания № 0065-2019-0014 и проекта РФФИ № 19-07-00861.

Литература

1. Боев Б.В. Прогнозно-аналитические модели эпидемий. М., 2005. 10 с. URL: https://armscontrol.ru/course/lectures05a/bvb050324.pdf (дата обращения: 15.08.2019).
2. Карпов Ю.Г. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5. СПб: БХВ-Петербург, 2005. 400 с.
3. Palyukh B., Ivanov V., Sotnikov A. Evidence theory for complex engineering system analyses. Advances in Intelligent Systems and Computing. 2019, vol. 874, pp. 70–79. DOI: 10.1007/978-3-030-01818-4_7.
4. Korneev V., Semenov D., Kiselev A., Shabanov B., Telegin P. Multiagent distributed grid scheduler. Proc. FedCSIS, 2011, pp. 577–580.

5. Телегин П.Н. Настройка выполнения параллельных программ // Программные продукты и системы. 2012. № 4. С. 25–30.

6. Лапшина С.Ю. Мультиагентное моделирование процессов распространения эпидемии с использованием суперкомпьютеров // Программные продукты и системы. 2018. Т. 31. № 3. С. 640–644. DOI: 10.15827/0236-235X.123.640-644.

7. Hoshen J., Kopelman R. Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm. Phys. Rev. B, 1976, vol. 14, pp. 3438–3445. DOI: 10.1103/PhysRevB.14.3438.

8. Тарасевич Ю.Ю. Перколяция: теория, приложения, алгоритмы. М.: УРСС, 2002. 64 с.

9. Lapshina S. Parallel cluster multiple labeling technique. LJM, 2019, vol. 40, iss. 5, pp. 555–561. DOI: 10.1134/S1995080219050123.

10. Lapshina S. High-performance computations in multi-agent simulation problems of percolation cluster's behavior. LJM, 2019, vol. 40, iss. 3, pp. 341–348. DOI: 10.1134/S1995080219030144.

Software & Systems
DOI: 10.15827/0236-235X.128.573-580

Received 21.08.19
2019, vol. 32, no. 4, pp. 573–580

Investigation of the optimal number of processor cores for parallel cluster multiple labeling on supercomputers

S.Yu. Lapshina [1], *Senior Researcher, lapshina@jscc.ru*
A.N. Sotnikov [1], *Dr.Sc. (Physics and Mathematics), Professor, asotnikov@jscc.ru*
V.E. Loginova [1], *Leading Engineer-Programmer, vl@jscc.ru*
C.Yu. Yudintsev [1], *Research Associate, climenty@jscc.ru*

[1] *Joint Supercomputer Center of RAS – Branch of Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences" (JSCC RAS – Branch of SRISA), Moscow, 119334, Russian Federation*

Abstract. The article considers the optimum number of processor cores for launching the Parallel Cluster Multiple Labeling Technique in the course of conducting simulation experiments on the problem of multi-agent modeling of the spread of mass epidemics on modern supercomputer systems installed in the JSCC RAS.

This algorithm can be used in any field as a tool for differentiating large lattice clusters, because he is given input in a format independent of the application. At the JSCC RAS, this tool was used to study the problem of the spread of epidemics, for which an appropriate multiagent model was developed.

The model considers an abstract disease transmitted by contact. During the simulation, the threshold value of the probability of infection is determined (i.e., the probability of infection itself is a variable parameter), at which the percolation effect appears on the distribution grid of the disease. If this value is close to the contagiousness index of a particular disease, then there is every chance of expecting an epidemic to spread on a planetary scale.

In the course of imitation experiments, a variant of the Parallel Cluster Multiple Labeling Technique for percolation Hoshen-Kopelman clusters related to the tag linking mechanism, which can also be used in any area as a tool for differentiating large-size lattice clusters, was used to be improved on a multiprocessor system.

The article provides an estimate of the execution time of the Parallel Cluster Multiple Labeling Technique for Hoshen-Kopelman percolation clusters for various values of input parameters on high-performance computing systems installed in the JSCC RAS: MVS-10P MP2 KNL, MVS-10P OP, MVS 10P Tornado, MVS-100K.

Keywords: multi-agent simulation, percolation's cluster, parallel cluster multiple labeling technique, high-performance computing systems, processor cores.

Acknowledgements. *The research was performed at the MSC RAS, the state assignment no. 0065-2019-0014, RFBR grant no. 19-07-00861.*

References

1. Boev B.V. *Predictive and Analytical Models of Epidemics*. 2005. Available at: https://armscontrol.ru/course/lectures05a/bvb050324.pdf (accessed August 15, 2019) (in Russ.).

2. Karpov Yu.G. *Simulation of Systems. Introduction to Modeling with AnyLogic 5*. St. Petersburg, BHV-Peterburg, 2005, 400 p. (in Russ.).

3. Palyukh B., Ivanov V., Sotnikov A. Evidence theory for complex engineering system analyses. *Advances in Intelligent Systems and Computing*. 2019, vol. 874, pp. 70–79. DOI: 10.1007/978-3-030-01818-4_7.

4. Korneev V., Semenov D., Kiselev A., Shabanov B., Telegin P. Multiagent distributed grid scheduler. *Proc. FedCSIS*, 2011, pp. 577–580.

5. Telegin P.N. Tuning execution of parallel programs. *Programmnye produkty i sistemy* [*Software and Systems*]. 2012, no. 4, pp. 25–30 (in Russ.).

6. Lapshina S.Yu. Multi-agent simulation of epidemics' distribution on supercomputers. *Programmnye produkty i sistemy* [*Software and Systems*]. 2018, vol. 31, no. 3, pp. 640–644 (in Russ.). DOI: 10.15827/0236-235X.123.640-644**.**

7. Hoshen J. and Kopelman R. Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm. *Phys. Rev. B.* 1976, vol. 14, pp. 3438–3445. DOI: 10.1103/PhysRevB.14.3438.

8. Tarasevich Yu.Yu. *Percolation: Theory, Applications, Algorithms*. Moscow, URSS Publ., 2002, 64 p.

9. Lapshina S. Parallel Cluster Multiple Labeling Technique. *LJM*, 2019, vol. 40, iss. 5, pp. 555–561. DOI: 10.1134/S1995080219050123.

10. Lapshina S. High-performance computations in multi-agent simulation problems of percolation cluster's behavior. *LJM*. 2019, vol. 40, iss. 3, pp. 341–348. DOI: 10.1134/S1995080219030144.

Для цитирования

Лапшина С.Ю., Сотников А.Н., Логинова В.Е., Юдинцев К.Ю. Исследование оптимального количества процессорных ядер для алгоритма многократной маркировки перколяционных кластеров на суперкомпьютерных вычислительных системах // Программные продукты и системы. 2019. Т. 32. № 4. С. 573–580. DOI: 10.15827/0236-235X.128.573-580.

For citation

Lapshina S.Yu., Sotnikov A.N. Loginova V.E., Yudintsev C.Yu. Investigation of the optimal number of processor cores for parallel cluster multiple labeling on supercomputers. *Software & Systems*. 2019, vol. 32, no. 4, pp. 573–580 (in Russ.). DOI: 10.15827/0236-235X.128.573-580.