Реализация потоковых вычислений с использованием формализма акторов для моделирования распределённой сортировки вставками

С.В. Востокин[1], И.В. Казакова[1]

[1]Самарский национальный исследовательский университет имени академика С.П. Королёва, 443086, Московское шоссе, 34, Самара, Россия

Аннотация. В работе исследуется вопрос применимости модели акторов для описания вычислений в виде потока независимых работ. Рассматривается программная реализация имитационной модели потоковых вычислений на примере распределенного алгоритма сортировки вставками. С использованием имитационной модели, откалиброванной по результатам натурных измерений на кластере, исследуется возможность ускорения вычислений в данном алгоритме. Рассматриваются приложения предлагаемого в работе подхода организации вычислений.

1. Введение

В современном мире параллельное программирование приобретает все большую актуальность в разработке сложных систем, предназначенных не только для научных исследований, но и для коммерческого использования. Одной из моделей, позволяющих реализовывать механизм распараллеливания вычислений, является модель акторов. К ее основным достоинствам относятся: асинхронный обмен сообщениями, возможность динамического роста системы в пределах распределенной среды [1].

Целью данной работы является исследование возможности применения модели акторов для описания вычислений в виде потока независимых операций. Такая организация вычислений используется в пакетных системах TORQUE, Slurm, композитных приложениях в сервис-ориентированных архитектурах, например, Everest [2].

В качестве примера рассматривается программная реализация имитационной модели потоковых вычислений распределенного алгоритма сортировки вставками. С использованием имитационной модели, откалиброванной по результатам натурных измерений на одном узле кластера, исследуется время вычислений в распределенном алгоритме с произвольным числом узлов кластера.

2. Методика эксперимента

В качестве основы для распараллеливания взят следующий алгоритм:

```
for (int i = 0; i<NUM_BLOCKS; i++) block_sort(i);
for (int i = 1; i<NUM_BLOCKS; i++) for (int j = 0; j<i; j++) block_merge(j, i),
```

из которого следует, что все блочные операции block_sort(i) выполняются независимо друг от друга, также являются независимыми некоторые операции слияния - block_merge(j, i). Наша цель – обеспечить такое описание параллельного алгоритма, чтобы независимые операции выполнялись параллельно.

Для реализации поставленной цели был разработан исходный код на языке программирования C++, основой которого являются. Акторы независимы друг от друга и

взаимодействуют между собой посредством передачи сообщений. В ответ на полученное сообщение они способны:

- отправлять сообщения другим акторам;
- создавать новые акторы;
- задавать свое поведение в ответ на следующее полученное сообщение.

Все вычисления, производимые при исполнении кода, организованы по определенному алгоритму, включающему управляющую подсистему и подсистему выполнения задач (рисунок 1).

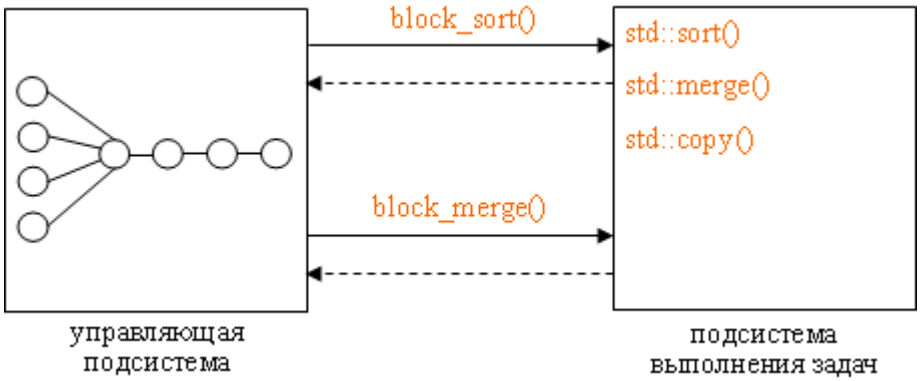

Рисунок 1. Схема организации вычислений.

Управляющая подсистема определяет стратегию вычислений и моделирует управляющий узел, механизм которого аналогичен управляющему узлу системы, приведенной в работе [3]. Ее основная задача заключается в формировании последовательности инструкций block_sort() и block_merge(), исходя из полученных сообщений для подсистемы выполнения задач. Управляющая подсистема содержит четыре основных типа акторов: sorter, merger, producer и stopper.

Типы акторов producer и stopper являются вспомогательными. Producer выполняет две функции: ожидает окончания всех блочных сортировок в sorter и для merger формирует последовательность номеров блоков. Stopper выполняет единственную функцию – останавливает вычисления.

Основная функция sorter заключается в сортировке методом Хоара полученных блоков данных. Данный метод можно назвать одним из самых быстрых. Заключается он в выборе одного элемента, передвижении его на конечное место. Все остальные элементы будут перемещены относительно выбранного в соответствии с их значениями либо в правую часть массива (если они меньше выбранного), либо в его левую часть (если они больше выбранного) [4].

Merger выполняет слияние блоков и сортировку методом вставок. Сортировка вставками – это алгоритм, в котором перемещение элементов блока происходит по одному в созданную, ранее отсортированную, последовательность. Является наиболее эффективным при слиянии ранее отсортированных блоков. Реализуется он при помощи следующего кода:

```
if (!(access(_in) && access(out)))return;
        if(is_first){
                is_first=false;    j = _in->i;
                _in->send();
        }
        else{
#if defined(SIMULATED_EXECUTION)
                double time;
                time = omp_get_wtime();
#endif
                block_merge(j,_in->i);
```

```
#if defined(SIMULATED_EXECUTION)
                    time = omp_get_wtime() - time;
                    delay(time*(1 + TRADEOFF));
#endif
                    out.i = _in->i;
                    _in->send();out.send();
} .
```

Рассмотрим процесс их взаимодействия (рисунок 2).

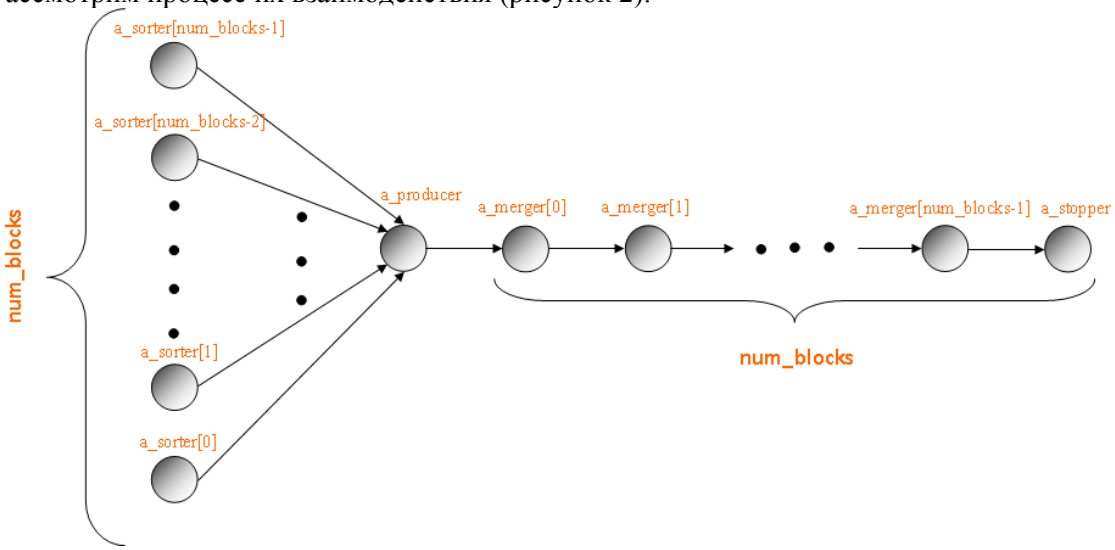

Рисунок 2. Система акторов. Алгоритм блочной сортировки.

Исходный массив состоит из 128000000 целых чисел, заполняется случайными значениями, затем разделяется на заданное количество равных блоков. Полученные блоки адресуются для выполнения быстрой сортировки типу акторов sorter. На каждый блок создается отдельный актор. Их количество варьируется в зависимости от количества блоков, в результате получается последовательность a_sorter[0], a_sorter[1], … , a_sorter[num_blocks-1]. По окончанию сортировки, в a_producer отправляется сообщение о выполнении. Как только a_ producer получает необходимое для дальнейших операций количество подтверждений о выполнении от акторов a_sorter, он формирует и передает инструкции следующему типу акторов – merger. Актор a_merger получает сообщение и производит дальнейшие действия в соответствии со следующим алгоритмом: если сообщение получено впервые, то считается, что в нём передается номер итерации по j; номер j запоминается и отправляется ответ актору слева в цепочке (рисунок 2); если сообщение поступило во второй и дальнейшие разы, то актор a_merger понимает, что в сообщении поступил номер итерации по i, производит слияние блоков block_merge(j,i), передает номер итерации по i следующему в цепочке справа актору a_merger (рисунок 2) и отправляет подтверждающее сообщение актору слева в цепочке. Как только сообщение получает актор a_stoper, этот актор останавливает вычисления.

Параллелизм в данном алгоритме осуществляется за счёт асинхронной обработки сообщений акторами. После выполнения некоторой операции, немедленно выполняется следующая, зависимая от них, операция. Одновременно могут выполняться несколько акторов sorter и merger.

3. Результаты эксперимента

Для исследования возможности ускорения вычислений в алгоритме была создана имитационная модель, откалиброванная по результатам натурных измерений на кластере. В процессе эксперимента были произведены расчеты для последовательного и параллельного алгоритмов. За исходные данные взят массив из 128000000 целых чисел и разделен согласно таблице 1.

Таблица 1. Количество и размер блоков, используемых для исследования возможности ускорения вычислений в последовательном и параллельном алгоритмах.

Количество блоков	Размер блока
2	64000000
4	32000000
8	16000000
16	8000000
32	4000000
64	2000000
128	1000000

Все этапы исследования проводились при помощи сервиса Templet Web [5], используемого для управления и мониторинга задач, запускаемых на кластере «Сергей Королёв». Кластер «Сергей Королёв» состоит из одного управляющего и 165 вычислительных серверов. Каждый узел кластера имеет жесткий диск для хранения временных данных. На узлах n1-n34 доступно 56 Гб, на n35-n42, n57-n70, n113-n168 – 120 Гб, n239-n242, n246-n250 – 244 Гб, n281-n308 - 95 Гб [6]. В первом эксперименте производилось исследование возможности ускорения вычислений в рабочем режиме. Для выполнения задачи было задействовано 8 процессоров, общее время выполнения каждого алгоритма составляло максимально 5 минут. Были задействованы узлы, приведенные в таблице 2.

Таблица 2. Задействованные узлы кластера для первого эксперимента.

Количество блоков	Размер блока	Узел кластера
2	64000000	n42
4	32000000	n128
8	16000000	n228
16	8000000	n118
32	4000000	n126
64	2000000	n118
128	1000000	n126

В эксперименте производился замер времени вычислений для последовательной сортировки, последовательной сортировки блоками и параллельной сортировки блоками с применением модели акторов. Результаты приведены на рисунке 3.

Рисунок 3. Зависимость времени выполнения сортировок от количества блоков в разделении массива (рабочий режим).

Второй эксперимент проводился в режиме симуляции. Были произведены расчеты для последовательной сортировки в рабочем режиме, последовательной сортировки и параллельной сортировки в режиме симуляции. Для выполнения алгоритма последовательной сортировки в рабочем режиме было задействовано 8 процессоров. Для замера времени вычислений в режиме симуляции были сымитированы дополнительные процессоры, количество которых ровно количеству блоков. Задействованные узлы приведены в таблице 3:

Таблица 3. Задействованные узлы кластера для второго эксперимента**.**

Количество блоков	Размер блока	Узел кластера
2	64000000	n36
4	32000000	n6
8	16000000	n35
16	8000000	n7
32	4000000	n136
64	2000000	n149
128	1000000	n145

Результаты второго эксперимента приведены на рисунке 4.

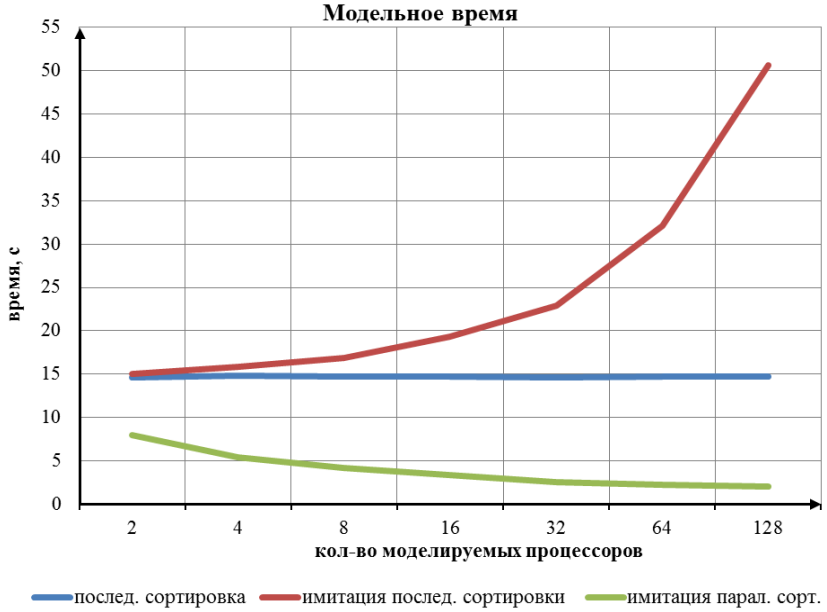

Рисунок 4. Зависимость времени выполнения сортировок от количества блоков в разделении массива (режим симуляции).

При выполнении сортировки происходит обмен данными между процессорами, что приводит к увеличению времени выполнения алгоритма. Процент накладных расходов может варьироваться в зависимости от удаленности узла, например, при передаче по сети. В третьем эксперименте был произведен замер затрачиваемого времени на взаимодействие между процессорами в режиме симуляции, выявлено влияние накладных расходов на время выполнения блочной сортировки. Для выполнения поставленной задачи взято 32 блока по 4000000 целых чисел, задействовано 8 процессоров. Для замера времени вычислений в режиме симуляции использованы 32 процессора. Задействованные узлы приведены в таблице 4.

Таблица 4. Задействованные узлы кластера для второго эксперимента.

Процент накладных расходов	Узел кластера
0	n136
10	n134
20	n147
30	n152
40	n130
50	n151
60	n145
70	n12
80	n117
90	n118
100	n228

На рисунке 5 приведены результаты третьего эксперимента, которые демонстрируют влияние накладных расходов на время выполнения блочной сортировки в режиме симуляции.

Рисунок 5. Влияние накладных расходов на время выполнения блочной сортировки в режиме симуляции.

Четвертый эксперимент демонстрирует накладные расходы на выполнение сортировки из двух блоков различных размеров с передачей данных по сети в рабочем режиме. Передача осуществлялась между двумя узлами: n36 и n136. Использовались блоки, включающие в себя 1000000, 2000000, 4000000, 8000000, 16000000, 32000000, 64000000 целых чисел. Результаты четвертого эксперимента приведены на рисунке 6.

4. Обсуждение результатов

Первый эксперимент показал, что время вычислений параллельной сортировки блоков значительно уменьшается по сравнению с последовательной сортировкой блоков. Наиболее эффективным алгоритм был для сортировки 16 блоков, состоящих из 8000000 целых чисел, время выполнения которого составляет 4,84845 с. Для последовательной сортировки блоков с такими же изначальными условиями потребовалось 19,245 с.

Особенностью второго эксперимента было имитационное выполнение вычислений. Используемый в эксперименте акторный фреймворк системы Templet Web [5] позволяет

вычислить время работы по алгоритму в предположении, что каждый актор исполняется на отдельном процессоре.

Рисунок 6. Накладные расходы на выполнение сортировки из двух блоков различных размеров с передачей данных по сети в рабочем режиме.

При этом фактическая работа алгоритма осуществляется на одном ядре реального процессора, а временные задержки отдельных операций передаются в имитационную систему для расчёта модельного времени. В коде актора Merger можно видеть вызов метода delay(), передающий измеренное время слияния двух блоков с учетом накладных расходов в систему имитационного исполнения. Таким образом, во втором эксперименте количество модельных процессоров равно количеству блоков. Результаты второго эксперимента также демонстрируют эффективность применяемого метода параллельной сортировки блоков. Для 2, 4, 8 и 16 блоков, включающих 64000000, 32000000, 16000000 и 8000000 целых чисел, результат в режиме симуляции незначительно отличается от результата в рабочем режиме. Но для 32, 64 и 128 блоков, включающих 4000000, 2000000 и 1000000 целых чисел, заметен интенсивный рост времени для алгоритма последовательной сортировки блоков и снижение временных затрат на выполнение параллельной сортировки блоков. Для 128 блоков из 1000000 целых чисел результат последовательной сортировки блоков равен 50,6568 с, а результат параллельной сортировки блоков составляет 2,0665 с.

В третьем эксперименте исследовалось влияние накладных расходов на время выполнения блочной сортировки в режиме симуляции. Из рисунка 5 видно, что увеличение процента накладных расходов приводит к увеличению времени выполнения вычислений. Если время параллельной сортировки блоков с нулевым процентом накладных расходов составляет 2,54216 с, то при увеличении процента накладных расходов до 100 %, оно составит 4,85475 с. Исходя из показателей третьего эксперимента, можно утверждать, что увеличение накладных расходов до 100% приводит к увеличению времени выполнения алгоритма на 190,97%. Однако результаты четвертого эксперимента демонстрируют, что в рабочем режиме накладные расходы составляют не более 10%, что незначительно влияет на время выполнения параллельной сортировки блоков.

5. Заключение

Целью настоящей работы было исследование возможности применения модели акторов для описания вычислений в виде потока независимых процессов. В качестве примера рассмотрена программная реализация имитационной модели потоковых вычислений распределенного алгоритма сортировки вставками.

Проведенные эксперименты продемонстрировали, что распределенная сортировка предложенным методом позволяет снизить временные затраты относительно методов последовательной сортировки, даже с учетом накладных расходов. Полученные результаты являются доказательством эффективности исследуемого в статье подхода организации вычислений.

Результаты проделанной работы и предлагаемый подход организации вычислений могут быть применены в различных алгоритмах обработки больших массивов данных, использующих методы сортировки. С исходным кодом рассмотренного метода сортировки можно ознакомиться на сервисе GitHub [7].

6. Литература

[1] Федотов, И.Е. Модели параллельного программирования / И.Е. Федотов. – Москва: Солон-Пресс, 2012. – 384 с.

[2] Sukhoroslov, O. A Web-Based Platform for Publication and Distributed Execution of Computing Applications / O. Sukhoroslov, S. Volkov, A. Afanasiev // IEEE Xplore. – 2015. – Vol. 14. – P. 175-184.

[3] Knyazkov, K.V. Clavire: e-science infrastructure for data-driven computing / K.V. Knyazkov, S.V. Kovalchuk, T.N. Tchurov, S.V. Maryin, A.V. Boukhanovsky // Journal of Computational Science. – 2012. – Vol. 3(6). – P. 504-510.

[4] Кнут, Д.Э. Искусство программирования. Том 3. Сортировка и поиск / Д.Э. Кнут. – Москва: Вильямс, 2000. – 832 с.

[5] Vostokin, S.V. Templet Web: the experimental use of volunteer computing approach in scientific Platform-as-a-Service implementation / S.V. Vostokin, Y.S. Artamonov, D.A. Tsarev // Proceedings of the Third International Conference BOINC-based High Performance Computing: Fundamental Research and Development (Petrozavodsk, Russia, August 28 - September 01, 2017). – CEUR Workshop Proceedings. – 2017. – P. 129-135.

[6] Суперкомпьютерный центр Самарского университета [Электронный ресурс]. – Режим доступа: http://hpc.ssau.ru/node/22 (18.11.2017).

[7] Алгоритм блочной сортировки [Электронный ресурс]. – Режим доступа: https://github.com/Templet-language/newtemplet/tree/master/samples/blocksort (18.11.2017).

Implementation of stream processing using the actor formalism for simulation of distributed insertion sort

S.V. Vostokin[1], I.V. Kazakova[1]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086

Abstract. In this paper we study the applicability of actor formalism in the field of distributed stream processing. The simulation model of stream processing is considered using the example of a distributed insertion sort algorithm. We built a simulation model calibrated by the results of full-scale measurements of the sorting time on the computing cluster. Possible applications of the proposed approach to the organization of calculations are considered.

Keywords: workflow, actor model, high-throughput computing, high-performance computing, simulation software, computer performance evaluation.