DOI 10.37882/2223–2966.2021.01.36

МЕТОДЫ ОЦЕНКИ И ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

METHODS FOR EVALUATING AND IMPROVING THE PERFORMANCE OF MULTIPROCESSOR COMPUTING SYSTEMS

V. Filimonov
V. Suskin

Summary. The existing methods of data analysis and processing in multicore computing systems are considered. The aim of the work is to study high-performance computing systems, parallel computing models and the most efficient algorithms for determining the optimal way to increase the speed of information processing. The problem of ensuring uniform loading and the possibility of mathematical expression of the efficiency of the computer system is presented. The key factor that determines the performance of multiprocessor computing systems is highlighted.

Keywords: performance of computing systems, optimization methods, multiprocessor computing systems, parallel computing.

Филимонов Владислав Валерьевич
Аспирант, Рязанский государственный радиотехнический университет
filvvfil@yandex.ru
Сускин Виктор Васильевич
Д.т.н., профессор, Рязанский государственный радиотехнический университет

Аннотация. Рассматриваются существующие методы анализа и обработки данных в многоядерных вычислительных системах. Целью работы является исследование высокопроизводительных вычислительных систем, моделей параллельных вычислений и наиболее эффективных алгоритмов для определения оптимального способа повышения скорости обработки информации. Представлена задача обеспечения равномерной загрузки и возможность математического выражения эффективности работы вычислительной системы. Выделен ключевой фактор, от которого зависит производительность многопроцессорных вычислительных систем.

Ключевые слова: производительность вычислительных систем, методы оптимизации, многопроцессорные вычислительные системы, параллельные вычисления.

Введение

Современный этап развития средств вычислительной техники открывает огромные возможности для разработки самых различных алгоритмов оптимизации и решения сложных высокопроизводительных задач. Технологической процесс производства полупроводниковых процессоров уже пришел к отметке 10 нм практически во всех ведущих компаниях, таких как IBM, Intel, TSMC и других, а в индустрии мобильного производства ведущие компании Apple и HiSilicon уже запустили в массовое производство процессоры на 5нм техпроцессе. Немало важную роль в развитии вычислительной техники и гонке производительности сыграла всемирная популярность блокчейн (Blockchain) технологий. На наш взгляд, наиболее востребованными на сегодняшний день являются задачи машинного обучения, т.к. именно в них требуется обработка большого количества информации и обеспечение высокой точности принятия решений. Технологии машинного обучения используются во многих сферах, их применение можно встретить как в задачах распознавания фото и видеоизображений, в области здравоохранения, обеспечения безопасности, так и в более бытовых, к примеру, прогнозирование спроса блюд и совершенствование рецептов в заведениях общественного питания [1].

Наблюдая за развитием компьютерных систем в последние несколько лет, можно заметить непрерывный рост количества ядер и потоков процессоров, при этом максимальная тактовая частота достигла определенного пика на значении ~5,5 ГГц, следовательно, при решении задач повышения производительности, будет не правильно полагаться на увеличение данного параметра. Исходя из этого, мы считаем, что повысить быстродействие обработки информации можно только за счет применения эффективных алгоритмов анализа и распараллеливания выполняемых задач. Существующие многопроцессорные системы дают возможность организовать выполнение нескольких задач в разных потоках, тем самым обеспечив высокое быстродействие при мультизадачности.

Таким образом, рассмотрев перспективные структуры и задачи, можно сделать вывод об актуальности вопроса оптимизации обработки данных и расширения функциональности систем, за счет применения новых методов планирования и информационных моделей, отображаемых в структуре современных вычислительных систем (ВС).

Таблица 1. Расчетные значения по закону Амдала

N / A(%)	2	4	8	16
0	2	4	8	16
25	1,6	2,29	2,91	3,37
50	1,33	1,6	1,78	1,89
100	1	1	1	1

Теоретическая часть

С увеличением сложности производимых вычислений и объемов входной информации, скорость обработки данных становится приоритетной задачей, для реализации которой, немало важную роль играют способы и алгоритмы организации сложных вычислений [4, с. 32; 6, с. 128]. Как говорилось ранее, при достижении процессором определенного пика скорости вычислений, вопрос распараллеливания вычислительных процессов встает довольно остро. В некоторых случаях, даже при наличии одного процессора в компьютерной системе, можно повысить его эффективность, используя технологию гиперпоточности (Hyper-Threading), которая позволяет организовать процесс вычислений на одном процессоре так, как на нескольких логических.

Для объективного рассмотрения проблемы повышения эффективности, и оценки возможного ускорения системы, будем использовать закон Амдала (1).

Общее время работы нашей программы при однопоточном выполнении обозначим за T, долю от общих вычислений, которая может быть выполнена только последовательными расчетами, обозначим A, количество максимально возможных потоков для вычислений обозначим N, таким образом, получим формулу, описывающую полученное ускорение (S_p), в сравнении с последовательным (однопоточным) решением:

$$S_p = \frac{T}{TA+\frac{T-T-A}{N}} = \frac{1}{A+\frac{1-N}{N}} \quad (1)$$

Рассмотрев данный закон, мы понимаем, что наибольшее влияние на эффективность ускорения влияет та часть программы, которая может быть выполнена параллельно. Таким образом, стоит понимать, что не для каждой задачи увеличение количества потоков и, соответственно, распараллеливание вычислений будет иметь смысл. Для наглядности, рассчитаем и запишем в таблицу выборку значений.

Из таблицы видим, что максимальный прирост производительности наблюдается в той программе, где отсутствует необходимость последовательных вычислений и их все можно выполнить параллельно. Однако, на практике, идеальные условия встречаются крайне редко, поэтому основной вывод, который можно сделать из данных расчетов, что прирост эффективности для каждой задачи ограничен объемом последовательных вычислений, а, следовательно, прирост производительности не может превышать значения. Так, при половине последовательного кода, максимальный прирост производительности не может превысить 2.

Важно отметить, что данные расчеты не учитывают, возможности повышения сложности и объема решаемых задач, а также издержек возникающих при взаимодействии параллельно работающих процессоров между собой и обращение к различным уровням памяти [7, с. 400; 9, с. 384]. Чтобы получить более точную оценку ускорения в параллельных вычислениях, предлагаем рассмотреть еще несколько законов и метрик.

Для учета возможного увеличения вычислений, воспользуемся законом Густафсона-Барсиса (2). Как и в предыдущем расчете, примем долю расчетов, которые необходимо выполнять только последователь за A, количество потоков за N, а также будем учитывать долю параллельных расчетов программы P. В результате, получаем отношение объема параллельных вычислений к последовательным за один промежуток времени (S_A):

$$S_A = \frac{PN+A}{P+A} = \frac{N(P+A)}{P+A} + \frac{A-AN}{P+A} = N + (1-N)\frac{A}{P+A} =$$
$$= N + (1-N)A \quad (2)$$

Таким образом, при подстановке реальных значений, получаем ускорение масштабирования, выраженное объемом, на который возможно увеличение выполняемых задач.

Из таблицы видим, что приспособленность алгоритма к параллельным вычислениям и их использование, позволяют не только повысить скорость вычислений определенного объема информации, но и значительно увеличить объем обрабатываемой информации за единицу времени. Тем не менее, несмотря на внушительные коэффициенты масштабирования систем, получившиеся в расчетах, стремясь смоделировать наиболее точную систему, не стоит пренебрегать существующими ограничениям.

Таблица 2. Расчетные значения по закону Густафсона-Барсиса

N / A(%)	2	4	8	16
0	2	4	8	16
25	1,75	3,25	6,25	12,25
50	1,5	2,5	4,5	8,5
100	1	1	1	1

В многопроцессорных вычислительных системах, обычно, каждый процессор имеет собственную локальную память, которая в совокупности образует общую память и используется при решении задач. Таким образом, доступная емкость памяти создает ограничение на максимальный объем выполняемых вычислений [8, с. 320; 10, с. 400]. Для математического описания данного ограничения, на наш взгляд, хорошо подходит закон Сана-Ная (3). Его постановка, как и в предыдущем расчете, заключается в том, что при увеличении числа параллельных потоков, увеличивается объем выполняемых за единицу времени задач, но в условиях ограничения объема по памяти.

В качестве переменных данного выражения, будем использовать обозначения аналогичные предыдущим: A — доля последовательных вычислений, N — количество потоков и — коэффициент масштабируемости распараллеленной части. В результате получаем ускорение производительности, ограниченное памятью (S_N):

$$S_N = \frac{A+(1-A)G_n}{A+(1-A)\frac{G_n}{N}} \quad (3)$$

Из полученного выражения следует, что при $G_N = N$ размер задачи фиксирован, а при емкость памяти пропорционально возрастает нагрузке. Таким образом, при наличии дефицита памяти ($G_N < N$) расчетное ускорение производительности будет меньше, однако при наличии запаса по памяти ($S_N > N$)оценка ускорения будет более оптимистична.

Для оптимального описания возможности увеличения производительности многопроцессорный системы, также требуется рассмотреть взаимодействие между параллельно работающими процессорами. Воспользуемся метрикой Карпа-Флэтта (4), в ней предложено использовать эквивалент объема распараллеленной части программы показатель, который получают экспериментальным путем на конкретной вычислительной машине.

$$e = \frac{\frac{1}{S_N}-\frac{1}{N}}{1-\frac{1}{N}} \quad (4)$$

Оценкой эффективности в данном выражении служит сама метрика, чем меньше ее значение, тем лучше может быть распараллелен код, в идеальных ситуациях, аналогичных постановке задачи Амдала, при увеличении числа процессоров эффективность будет уменьшаться. Также благодаря данным вычислениям с экспериментальными данными мы можем определить узкое место, которое приводит к снижению эффективности, оно может быть выражено ограничением возможности распараллеливания или наоборот издержками коммуникаций при параллельных вычислениях.

Экспериментальная часть

На практике, производительность работы нашей системы и ее скорость выполнения необходимых нам задач зависит от множества факторов. В первую очередь от аппаратной части самой системы: частоты работы процессора и его непосредственной памяти, наличия видеокарты, объема оперативной памяти, скорости коммуникационных связей (пропускной способности), чтения/записи данных и прочих факторов. Предположим, что система с определенными параметрами у нас есть, вторым важнейшим условием эффективного решения поставленной задачи является алгоритм. От его выбора зависит, сможет ли вообще наша задача быстро решаться, если выбранный алгоритм изначально не оптимален, то дальнейшее его ускорение не принесет необходимого результата. Соответственно, после выбора алгоритма, основной задачей является его оптимизация под имеющуюся аппаратную часть, с учетом производительности и пропускной способности. В этом и будет заключаться основная цель нашей работы.

Рассмотрев существующие способы оценки производительности многоядерных/многопоточных систем [2, с. 688], мы заметили, что большинство из них недостаточно точно описывают либо вообще не учитывают, активно развивающуюся на сегодняшний день, многоуровневую кэшпамять в современных процессорах. Обычно, при упоминании в литературе, классическая модель вычислительной системы с иерархической организацией памяти (Рисунок 1), содержит в себе АЛУ (Арифметико-логическое устройство), служащее, непосредственно, для вычислений, Кэшпамять, которая является частью процессора, оперативную память (ОЗУ) и общую память, представляющую собой жесткие диски, твердотельные накопители и другие хранилища информации.

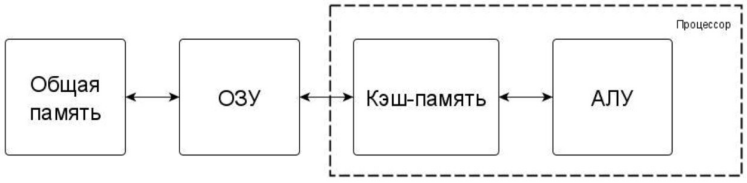

Рис. 1. Классическая модель ВС с иерархической структурой памяти

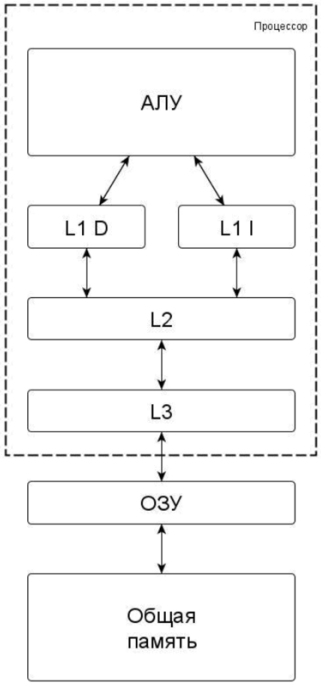

Рис. 2. Модель ВС, учитывающая уровни кэш-памяти процессора

Физическое расположение различных видов памяти, и как следствие, скорость передачи данных, на каждом уровне сильно различаются. Поэтому, мы считаем, важно учитывать в модели каждый уровень кэш-памяти процессора. Благодаря такому уточнению, можно более точно оптимизировать алгоритм выполнения задач, и соответственно, увеличить скорость их решения.

В связи с этим, предлагаем улучшенную модель вычислительной системы с иерархической структурой памяти, имеющей многоядерный процессор.

Перейдем к обоснованию данного улучшения на примере реальной компьютерной системы. Предположим, в нашей системе установлен процессор Intel Core i9–9900K, в его архитектуре содержатся 8 ядер, работающие в 16 потоков, кэш L3–16МБ, кэш L2–2МБ, кэш L1–512КБ. Скорости чтения и записи по каждому уровню кэш-памяти в пользовательской документации на процессор отсутствуют, поэтому, возьмем экспериментальные значения, полученные при помощи специальной программы AIDA64 [3]. В большинстве проведенных экспериментов, скорость записи происходит незначительно медленнее чтения, поэтому, для получения расчетов более близких к реальным значениям, будем рассматривать именно скорость записи. Для кэша L1 эта скорость 1143 ГБ/с, для L2 ~ 568 ГБ/с, и для кэша L3 ~ 220 ГБ/с. К сравнению, установленная в вычислительной системе ОЗУ, самого современного на текущее время, типа DDR4 показала максимальную скорость записи 41288 МБ/с. Следовательно, при разработке алгоритмов, важно учитывать количество и порядок вызовов функций, чтобы минимизировать лишнюю передачу данных между кэшом L1 и кэшами более высоких уровней и ОЗУ.

Задачу обеспечения сбалансированной нагрузки и оптимального обращения к памяти [5, с. 248], в идеальном случае, можно сформулировать следующим образом (5):

$$K = \frac{N_{max}}{N_p}, \quad 0 < K \leq 1 \tag{5}$$

N_{max} — максимальный объем задачи, помещающийся в память. Если размер задачи превысит данный объем,

произойдет обращение к памяти более высокого уровня и как следствие снижение производительности алгоритма.

N_p — количество ядер процессора, которые можно использовать для параллельной обработки. Учитывая загрузку каждого ядра, можно сформулировать следующее выражение (6):

$$K = \sum_{i=1}^{p} \frac{T_{ci}}{T_f} \cdot \frac{p_i(t_i)}{\sum_{j=1}^{P} P_j(t_j)} \quad (6)$$

Где i-ое ядро процессора, реализует свой объем вычислений за время. Время выполнения распределенной программы можно найти по формуле (7).

$$T_f = \max\{T_i\}, \qquad i = 1, \dots, P \quad (7)$$

P — число ядер процессора, а время учитывает в себе и время, которое показывает время вычислений и время передачи данных, $p_i(t_i)$- производительность процессора в момент времени. Приняв объем вычислений за W, получим (8):

$$T_{ci} = \frac{W_{ci}}{p_i(t_i)} \quad (8)$$

Таким образом, задача сводится к максимизации значения K, в промежутке времени T_f, при определенном количестве ядер процессора P.

Понимание данного процесса, позволяет разрабатывать оптимальные алгоритмы [11, с. 323; 12, с. 512], и оптимизировать обращение к памяти так, чтобы данные необходимые при вычислениях определенного блока информации не вытеснялись из кэша L1, тем самым замедляя общую скорость работы.

Заключение

В данной статье рассмотрены законы, на которых базируются методики оценки производительности многопроцессорных компьютеров, приведены расчетные данные по производительности распараллеленных систем с разным уровнем содержания последовательных вычислений. Предложена улучшенная модель описания вычислительной системы с иерархической структурой памяти. На реальных значениях, полученных экспериментальным путем, показана разница в объеме и скорости работы памяти одного из современных процессоров ведущей компании Intel. А также сформулировано математическое описание задачи для сбалансированной загрузки многоядерной процессорной системы. По итогу, можно сделать вывод, что для оптимизации скорости выполнения поставленной задачи, при разработке алгоритма, необходимо учитывать аппаратные возможности устройства, использовать параллельные вычисления и, в зависимости от объема имеющейся памяти, разделять вычисления на блоки таким образом, чтобы избегать вытеснения данных, необходимых для текущих расчетов, в память более высоких уровней.

ЛИТЕРАТУРА

1. Применение машинного обучения и Data Science в промышленности [Электронный ресурс] https://vc.ru/ml/79368-primenenie-mashinnogo-obucheniya-i-data-science-v-promyshlennosti
2. Цилькер Б. Я. Организация ЭВМ и систем: учебник для вузов / С. А. Орлов, Б. Я. Цилькер. СПб.: Питербург. 2011.
3. Benchmarking and Memory Tests. [Электронный ресурс] https://www.aida64.com/
4. Фаддеева В.Н., Фаддеев Д. К. Параллельные вычисления в линейной алгебре // Кибернетика. 1977. No 6.
5. Феррари Д. Оценка производительности вычислительных систем. М.: Машиностроение. 1989.
6. Ушкар М. Н. Микропроцессорные устройства в радиоэлектронной аппаратуре/ Под ред. Б. Ф. Высоцкого. М.: Радио и связь. 1988.
7. Воеводин В.В., Воеводин Вл. В. Параллельные вычисления. СПб.: «БХВ-Петербург». 2004.
8. Топорков В. В. Модели распределенных вычислений. М.: ФИЗМАТЛИТ. 2004.
9. Федотов И. Е. Модели параллельного программирования М.: Солон-Пресс. 2012.
10. Формалев В.Ф., Ревизников Д. Л. Численные методы. М.: ФИЗМАТЛИТ. 2004.
11. Шпаковский Г.И., Серикова Н. В. Программирование для многопроцессорных систем в стандарте MPI / Г. И. Шпаковский, Н. В. Серикова. Минск: Изд-во БГУ. 2002.
12. Эндрюс Г. Р. Основы многопоточного, параллельного и распределенного программирования. М.: Издательский дом «Вильямс». 2003.

Журнал «Современная наука: актуальные проблемы теории и практики»