На правах рукописи

Махнычев Владимир Сергеевич

Распараллеливание алгоритмов ретроанализа для решения переборных задач в вычислительных системах без общей памяти

Специальность 05.13.11 –
математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

Автореферат
диссертации на соискание ученой степени
кандидата физико-математических наук

Москва – 2012

Работа выполнена на кафедре автоматизации систем вычислительных комплексов факультета вычислительной математики и кибернетики Московского государственного университета имени М.В. Ломоносова.

Научный руководитель: кандидат физико-математических наук, доцент **Гуляев Анатолий Викторович**

Официальные оппоненты: доктор технических наук, профессор **Рябов Геннадий Георгиевич**

доктор технических наук, профессор **Топорков Виктор Васильевич**

Ведущая организация: Московский физико-технический институт

Защита диссертации состоится 19 октября 2012 г. в 11 часов на заседании диссертационного совета Д 501.001.44 при факультете вычислительной математики и кибернетики Московского государственного университета имени М. В. Ломоносова по адресу: 119991, ГСП-1, Москва, Ленинские горы, 2-й учебный корпус, факультет ВМК, аудитория 685. Желающие присутствовать на заседании диссертационного совета должны сообщить об этом за два дня по тел. (495) 939-30-10 (для оформления заявки на пропуск).

С диссертацией можно ознакомиться в Фундаментальной библиотеке МГУ имени М.В.Ломоносова. С текстом автореферата можно ознакомиться на официальном сайте ВМК МГУ http://cs.msu.ru в разделе «Наука» — «Работа диссертационных советов» — «Д 501.001.44».

Автореферат разослан «17» сентября 2012 г.

Заместитель председателя
диссертационного совета,
профессор В.М. Круглов

Общая характеристика работы

Введение. Параллельные вычисления в последние десятилетия развиваются бурными темпами. Суперкомпьютерные и кластерные технологии позволяют решать задачи, решение которых последовательными алгоритмами потребовало бы неприемлемо большого времени. При успешном создании параллельного алгоритма становится возможным получить совершенно новые, не достижимые ранее результаты для многих задач.

Одним из классов таких задач является построение оптимальной стратегии для игры двух противников методом ретроанализа. Оптимальная стратегия — полный план действий, который при безошибочных действиях обоих игроков позволяет либо достичь выигрыша за наименьшее число ходов (в случае, если выигрыш достижим), либо (в противном случае) максимально отдалить проигрыш или добиться ничьей, если это возможно. При построении такой стратегии требуется оптимизировать количество ходов как в сторону минимума (в случае выигрыша), так и в сторону максимума (при проигрыше), что обусловливает нетривиальность решения задачи.

Метод ретроанализа применяется к играм с полной информацией, игровой процесс которых заключается в поочередном выполнении игроками действий («ходов»), каждое из которых изменяет состояние игры; примерами таких игр могут служить шахматы, шашки, рэндзю, го (в этих играх состояние игры принято называть «положением на доске» или «позицией»). В теории, метод ретроанализа позволяет построить оптимальную стратегию для всей игры, но на практике из-за ограниченных вычислительных мощностей удается построить оптимальную стратегию лишь для некоторого класса состояний игры.

Актуальность работы. Распараллеливание и адаптация алгоритма ретроанализа к суперкомпьютерным системам без общей памяти делает возможным построение оптимальной стратегии для гораздо более широкого класса состояний игры, чем это позволяют последовательные алгоритмы. Кроме того, алгоритм ретроанализа является частным случаем динамического программирования, и подходы, примененные в данной работе при его распараллеливании, могут быть использованы при распараллеливании некоторых других задач, решаемых методами динамического программирования.

Построение оптимальной стратегии для новых классов состояний игры на персональных компьютерах упирается в огромные объемы данных и вычислений. Объем доступной оперативной памяти компьютера — одно из основных препятствий на пути решения задач большой размерности алгоритмом ретроанализа. Например, для задачи игры в шахматы при наличии на доске не более семи фигур количество позиций, данные о которых необходимо хранить одновременно, даже после применения различных

оптимизаций составляет не менее 640 миллиардов, что требует порядка 2 терабайт оперативной памяти. Такой объем недостижим для персональных компьютеров на сегодняшний день, но для современных суперкомпьютеров наличие десятков терабайт оперативной памяти является нормой. Построение параллельного алгоритма, обладающего высокой производительностью в системах с большим количеством процессоров (в частности, близкие к этой проблематике идеи содержатся в работах В. В. Топоркова), позволяет использовать всю эту память, а также выполнить все вычислительные операции (их количество в подобных задачах составляет порядка 10^{17}) за приемлемое время. При этом подавляющее большинство современных суперкомпьютеров не имеют общей памяти, поэтому наибольший интерес представляет распараллеливание алгоритма ретроанализа именно для систем без общей памяти.

Результаты, получаемые при помощи ретроанализа, традиционно имеют большую ценность как для практиков, так и для теоретиков соответствующих игр (например, построение решения для 3-4-5-6-фигурных окончаний в шахматах позволило существенно повысить силу игры шахматных программ в эндшпилях).

Цели и задачи работы. Целью работы являлась разработка эффективного метода построения оптимальной стратегии для указанного класса игр алгоритмом ретроанализа на суперкомпьютерных системах без общей памяти. Для достижения этой цели были поставлены следующие задачи:

1. Исследовать последовательный алгоритм ретроанализа и выяснить препятствия к его работе в системах без общей памяти.
2. Разработать параллельный алгоритм ретроанализа, рассчитанный на работу в суперкомпьютерных системах без общей памяти.
3. Провести апробацию и оценить эффективность предложенного алгоритма на практических задачах.
4. Исследовать масштабируемость алгоритма и разработать шаги по ее улучшению.

Научная новизна. В диссертационной работе предложен новый подход к эффективному построению оптимальной стратегии для пошаговых игр двух противников с полной информацией методом ретроанализа на суперкомпьютерных системах без общей памяти. Разработан и исследован новый параллельный алгоритм ретроанализа. Разработана схема балансировки нагрузки для этого параллельного алгоритма, рассчитанная на большое количество узлов.

Практическая значимость работы. Разработанный алгоритм построения оптимальной стратегии методом ретроанализа на суперкомпьютерной системе без общей памяти предназначен для эффективного поиска решений любых пошаговых игр двух противников с полной информацией, а также некоторых задач дискретной оптимизации, решаемых методом динамического программирования.

Реализован программный комплекс, реализующий разработанный алгоритм.

На базе этого программного комплекса реализован метод ретроанализа для решения задачи игры в шахматы. Эта реализация позволила впервые в мире построить оптимальную стратегию для задачи игры в шахматы для позиций, где на доске имеется в общей сложности не более семи фигур. Полученное решение представило большой интерес как для профессионалов в области шахмат, так и для любителей. Кроме того, предложенный алгоритм позволяет в обозримом будущем получить оптимальную стратегию для шахматных позиций, в которых на доске имеется в общей сложности восемь фигур.

Апробация работы и публикации. Результаты работы рассматривались на Всероссийской научной конференции «Научный сервис в сети Интернет» (Новороссийск, 2004 г.), на научной конференции «Ломоносовские чтения» (Москва, 2009 г.), на научной конференции «Ломоносовские чтения» (Москва, 2012 г.).

По теме работы опубликовано 7 статей (из них одна — в списке изданий, утвержденном ВАК), раскрывающих все основные научные результаты диссертации.

Структура и объем работы. Диссертация состоит из введения, четырех глав, заключения, списка публикаций и списка литературы. Текст работы изложен на 84 страницах. Список литературы включает 41 наименование. В работе содержится 9 рисунков и 3 таблицы.

Содержание работы

Во введении описаны задачи, решаемые алгоритмом ретроанализа, формулируются цель и задачи диссертации, обосновывается ее актуальность, научная новизна и практическая значимость, кратко излагается основное содержание работы.

В первой главе диссертации описан последовательный алгоритм ретроанализа для построения оптимальной стратегии для игры двух противников, а также некоторые подходы, применяемые для сокращения размерности решаемой задачи. Указаны препятствия к исполнению последовательного алгоритма ретроанализа на параллельной системе без общей памяти.

Алгоритм ретроанализа — частный случай динамического программирования. Основная идея алгоритма заключается в том, что решение строится последовательными шагами: от состояний игры с уже определенным исходом — к состояниям, где исход еще не определен.

В классическом виде алгоритм ретроанализа выглядит следующим образом.

Обозначим символом L множество состояний игры, для которых известно, что игрок, имеющий очередь хода, проигрывает, а символом W — множество состояний игры, для которых известно, что игрок, имеющий очередь хода, выигрывает. Изначально множества L и W пусты. Обозначим также символом $T(p)$ количество ходов одного игрока («полных» ходов), которое нужно сделать из состояния игры p, чтобы достичь выигрыша (если p принадлежит W), или через которое неизбежно наступит проигрыш (если p принадлежит L).

Шаг 1. Для всех возможных состояний игры p: если p — конечное, то есть в этом состоянии одним из игроков достигнут выигрыш, то p добавляется в L (если состояние игры проигрышное для того игрока, который имеет очередь хода) или в W (если победа присуждена игроку, имеющему очередь хода), а $T(p)$ полагается равным 0.

Шаг 2. Положить величину $V = 1$.

Шаг 3 (поиск выигрыша за V ходов, «мажорная итерация»). Для всех возможных состояний игры p, которые еще не принадлежат множествам W и L: если существует хотя бы один такой ход из p, который ведет в игровое состояние q, принадлежащее L, и такое, что $T(q)=V-1$ (то есть противник, оказавшийся в состоянии игры q, неизбежно проиграет через V-1 своих ходов), то добавить p в W и положить $T(p)=V$.

Шаг 4. Если $V > 1$ и на шаге 3 в множество W не было добавлено ни одного нового состояния игры, то перейти к шагу 8.

Шаг 5 (поиск проигрыша в V ходов, «минорная итерация»). Для всех возможных состояний игры p, которые еще не принадлежат множествам W и L: если все ходы из p ведут в такие состояния игры q, что q принадлежит W и $T(q) \le V$ (то есть все ходы ведут в состояния, где противник выигрывает не более чем за V своих ходов; легко видеть, что при выполнении этого условия для всех ходов найдется хотя бы одно состояние игры q, для которого $T(q)=V$), то добавить p в L и положить $T(p)=V$.

Шаг 6. Если на шаге 5 в множество L не было добавлено ни одного нового состояния игры, то перейти к шагу 8.

Шаг 7. Положить $V = V + 1$ и перейти на шаг 3.

Шаг 8 (окончание построения решения). Все возможные состояния игры, не принадлежащие ни W, ни L, являются ничейными. Оптимальная стратегия для каждого состояния игры теперь очевидна: для состояния $p \in W$ следует делать ход в такое состояние $q \in L$, чтобы $T(p)=T(q)+1$; для состояния $p \in L$ следует делать ход в такое состояние $q \in W$, чтобы $T(q)=T(p)$; для ничейных состояний игры, из которых есть возможные ходы, следует делать ход, ведущий также в ничейное состояние.

Для работы алгоритма необходимо хранение величин $T(p)$ для каждого состояния игры. Так как алгоритм предполагает неупорядоченный доступ к этим данным для произвольных состояний игры, то эффективная работа алгоритма требует хранения этих данных в оперативной памяти компьютера.

На системах без общей памяти (к которым относится подавляющее большинство современных суперкомпьютеров, и которые потенциально обладают значительно большей вычислительной мощностью, чем системы с общей памятью), где каждый вычислительный узел хранит свою часть данных, доступ к данным другого узла сопряжен со значительными задержками и по продолжительности на 3—5 порядков превышает время доступа к локальным данным.

Основным препятствием к исполнению последовательного алгоритма ретроанализа на системах без общей памяти является необходимость доступа к данным, находящимся в произвольной части оперативной памяти вычислительной системы.

Во второй главе диссертации представлен параллельный алгоритм ретроанализа. Описаны достоинства и недостатки разработанного параллельного алгоритма по сравнению с последовательным алгоритмом ретроанализа.

Для эффективной работы алгоритма ретроанализа на параллельной системе необходимо избавиться от операций доступа к памяти, требующих длительного ожидания. С этой целью в рамках данной работы был разработан *параллельный алгоритм ретроанализа*.

Суть параллельного алгоритма ретроанализа заключается в модификации последовательного алгоритма ретроанализа так, чтобы к моменту выполнения мажорных (поиск выигрыша) и минорных (поиск проигрыша) итераций вся необходимая информация уже находилась в локальной памяти вычислительного узла.

При выполнении мажорной итерации достаточно информации о том, что хотя бы один ход из обрабатываемого состояния игры ведет состояние, в котором проигрыш неизбежен в V ходов. Поэтому вычисление мажорных итераций видоизменяется следующим образом: всякий раз, когда состояние игры p помечается как проигрышное в G полных ходов, ко всем состояниям игры q, из которых p достижимо за один ход, добавляется пометка (назовем ее пометкой типа А) о том, что их необходимо рассмотреть на мажорной итерации при $V=G+1$. Добавление такой пометки к состоянию игры не требует задержки вычислений даже в том случае, если состояние игры обрабатывается другим вычислительным узлом: для добавления пометки достаточно отправить другому узлу сообщение по сети без ожидания доставки этого сообщения. Важно лишь, чтобы к началу очередной мажорной итерации все эти сообщения были доставлены.

Описанным образом удается избежать ожидания при работе с данными других узлов на мажорных итерациях алгоритма.

В минорных итерациях дело обстоит несколько сложнее. Для выполнения минорной итерации необходима информация о том, что все возможные ходы из обрабатываемого состояния игры p ведут к проигрышу, причем в наилучшем

(для проигрывающего) случае проигрыш наступает через V ходов. Для минорных итераций была разработана следующая схема.

Для каждого состояния игры r хранится величина $C(r)$ — количество ходов из r, которые не ведут к проигрышу, а также величина $M(r)$ — наибольшее расстояние (в полных ходах) до проигрыша среди тех ходов из r, о которых уже известно, что они ведут к проигрышу. В начале работы алгоритма $C(r)$ полагается равной количеству всех возможных ходов из r, а $M(r)=0$. Теперь при обнаружении выигрышного состояния игры p (пусть выигрыш в p достижим через G полных ходов) ко всем состояниям q, из которых p достижимо за один ход, добавляется пометка (назовем ее пометкой типа B) о том, что один из ходов из q ведет к неизбежному проигрышу в G полных ходов. Такая пометка также отсылается узлу, хранящему данные о состоянии игры q, асинхронно. При получении этой пометки узел, хранящий данные о q, уменьшает значение $C(q)$ на единицу (поскольку появилась информация о том, что еще один ход из q ведет к проигрышу), а значение $M(q)$ меняется на полученное значение G в том случае, если предыдущее значение $M(q)$ было меньше G: таким образом обеспечивается выбор наиболее длинного пути до проигрыша. Когда величина $C(q)$ достигает нулевого значения, это означает, что все ходы из q ведут к проигрышу, и q помечается как проигрышное состояние, где проигрыш неизбежен в $M(q)$ ходов.

С учетом вышесказанного, параллельный алгоритм ретроанализа выглядит следующим образом.

Шаг 1. Множество всех возможных состояний игры разбивается на k частей — по количеству узлов в вычислительной системе. Будем называть те состояния игры, информация о которых хранится в локальной памяти узла, *локальными.*

Шаги 2 — 8 выполняются всеми узлами системы параллельно.

Шаг 2. Положить $V=0, W=\varnothing, L=\varnothing$. Для всех возможных локальных состояний игры p: положить $C(p)$ равным количеству возможных ходов из p, $M(p)$=0.

Шаг 3. Для всех возможных локальных состояний игры p, являющихся конечными: если в p фиксируется проигрыш для игрока, имеющего очередь хода, то p добавляется в L, $T(p)$ полагается равным 0, и для всех состояний игры q, из которых p достижимо за один ход, отсылается пометка типа A со значением G=0. Если же в состоянии p победа присуждена игроку, за которым очередь хода, то p добавляется в W, $T(p)$ полагается равным 0, и для всех состояний игры q, из которых p достижимо за один ход, отсылается пометка типа B со значением G=0.

Шаг 4 (подготовка к параллельной мажорной итерации). Все вычислительные узлы увеличивают значение V на 1, и каждый узел должен убедиться в том, что все узлы завершили выполнение предыдущего шага (это шаг 3 или шаг 7), и что все предназначавшиеся ему пометки типа A, отправленные на предыдущем шаге, доставлены.

Шаг 5 (параллельная мажорная итерация). Для каждого локального состояния игры *p*, к которому имеется необработанная пометка типа A со значением *G=V*-1: если *p* уже принадлежит множеству *W* (это означает, что в *p* есть выигрыш за меньшее количество ходов), то проигнорировать пометку, иначе: добавить *p* в *W,* положить *T(p)=V*, и для каждого состояния игры *q*, из которого *p* достижимо в один ход, отослать пометку типа B со значением *G=T(p)*.

Шаг 6 (подготовка к параллельной минорной итерации). Каждый узел должен убедиться в том, что все узлы завершили выполнение предыдущего шага, и что все предназначавшиеся ему пометки типа B, отправленные на предыдущем шаге, доставлены.

Шаг 7 (параллельная минорная итерация). Для каждого локального состояния игры *p*, к которому имеется необработанная пометка типа B: если *p* уже принадлежит множеству *W* (это означает, что в *p* существует выигрыш), то проигнорировать пометку, а иначе: положить *C(p)=C(p)*-1, положить *M(p)*=max(*M(p)*, *G*), где *G* — значение, приписанное к обрабатываемой пометке. Если *C(p)* достигло нуля, то добавить *p* в *L,* положить *T(p)=M(p)*, и для каждого состояния игры *q*, из которого *p* достижимо в один ход, отослать пометку типа A со значением *G=T(p)*.

Шаг 8. Если осталась хотя бы одна необработанная пометка любого типа, то перейти к шагу 4.

Шаг 9 (окончание построения решения.) Все возможные состояния игры, не принадлежащие ни *W*, ни *L*, являются ничейными.

Представленный параллельный алгоритм не требует ожиданий при обращении к данным, хранящимся на других вычислительных узлах, что делает возможной его эффективную работу на системах без общей памяти. Кроме того, на практике этот алгоритм существенно превосходит последовательный алгоритм ретроанализа по скорости работы даже при исполнении его на одном вычислительном узле (в MPI-окружении с количеством MPI-процессов, равным 1), то есть фактически при задействовании той же вычислительной мощности, что и для последовательного алгоритма.

Однако следует отметить, что хотя представленный параллельный алгоритм имеет преимущество в скорости по сравнению с последовательным алгоритмом ретроанализа, он уступает ему по двум другим параметрам. Первый из них — требования к объему оперативной памяти: параллельному алгоритму необходимо для каждой состояния игры *p* хранить не только величину *T(p)*, но и значение *C(p)*. В условиях задач больших размерностей (сотни миллиардов состояний игры) дополнительные несколько бит на каждое состояние превращаются в лишние терабайты оперативной памяти, что может превысить возможности даже суперкомпьютеров. Второй параметр — сложность реализации: помимо реализации дополнительных структур данных для хранения пометок и величин *C(p)*, параллельному алгоритму требуется

вычислять не только ходы, возможные из состояния игры (эти ходы нужны для определения величин *C(p)*), но и ходы, ведущие в состояние игры («обратные» ходы) — для определения состояний игры, к которым необходимо сделать пометки.

Описанные приемы отправки пометок и подсчета количества возможных ходов для вычисления максимальных и минимальных значений могут быть применены и к другим задачам, решаемым методами динамического программирования.

В третьей главе диссертации описана реализация и результат применения параллельного алгоритма ретроанализа для получения точного решения задачи игры в шахматы при условии, что на доске осталось в сумме не более семи фигур.

Описан разработанный программный комплекс, позволяющий запускать параллельный алгоритм ретроанализа и не зависящий от специфики конкретной задачи. Реализация параллельного алгоритма выполнена таким образом, что она может быть легко адаптирована для решения произвольных задач методом ретроанализа.

Был полностью реализован описанный в главе 2 параллельный алгоритм ретроанализа. Для взаимодействия между вычислительными узлами использовалась библиотека MPI. Отправка и получение пометок к позициям производились при помощи асинхронных вызовов MPI_Isend и MPI_Irecv, причем для уменьшения накладных расходов пометки, предназначенные для одного вычислительного узла, объединялись перед отправкой в группы. По окончании итерации каждый вычислительный узел отправлял всем остальным узлам сообщения, включавшие еще не отправленные пометки, а также флаг, означающий, что от этого узла на этой итерации пометок больше не будет. Благодаря тому, что MPI гарантирует доставку сообщений между двумя узлами в том же порядке, в котором они были отправлены, принимающий узел, получив от всех остальных узлов сообщения с таким флагом, может быть уверен в том, что все пометки для него уже доставлены.

Помимо реализации собственно алгоритма ретроанализа, комплекс включает в себя вспомогательные средства, позволяющие автоматически подгружать на файловую систему суперкомпьютера результаты решения «предшествующих» подзадач (при разбиении множества всех состояний игры на подмножества, переходы между которыми возможны только в одном направлении — это позволяет получать решения для таких подмножеств последовательно) по мере выполнения алгоритма с их последующим удалением, а также контролировать ход вычислений.

На базе этого программного комплекса, в частности, было реализовано построение семифигурных эндшпильных таблиц.

Для сокращения объемов обрабатываемых данных применялись следующие оптимизации:

1. *Разбиение на подзадачи.* В рамках одной подзадачи рассматривались только позиции с одинаковым набором фигур. Взятие фигуры или превращение пешки соответствует переходу в «младшую» подзадачу, решение которой построено заранее.

2. *Выявление симметрий и исключение заведомо невозможных позиций.* Любую шахматную позицию, в которой игроки не имеют права рокировок, можно зеркально отразить относительно вертикальной оси, при этом решение для этой позиции не изменится. В случае же отсутствия на доске пешек возможно также зеркальное отражение позиции относительно горизонтальной оси и относительно одной из диагоналей. Были использованы дополнительные соображения, например, короли противников не могут стоять рядом.

Каждой рассматриваемой позиции *p* было поставлено в соответствие некоторое целое число («индекс позиции»), а величины *T(p)* и *C(p)* хранились в массиве, индексом которого и являлся индекс позиции. С учетом всех оптимизаций, удалось сократить максимальный размер индекса массива до $6{,}4 \cdot 10^{11}$ (такой размер получается для подзадачи «король, ладья, слон и конь против короля, ладьи и слона»).

При расчетах каждая позиция *p* представлялась в памяти 20 битами:

Биты	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	C(p)							H(p)												

где H(p) — количество полуходов до мата (эта величина равна $2 \cdot T(p)+1$ или $2 \cdot T(p)$ в зависимости от того, выигрышная позиция или проигрышная) для позиций, принадлежащих множествам W или L, и удвоенная величина *M(p)* в случае, если позиция не принадлежит этим множествам. При таком представлении данных наибольший необходимый суммарный объем оперативной памяти составил 1,6 терабайта.

По окончании работы алгоритма данные по каждой позиции переводились из 20-битного представления в 16-битное: для каждой позиции *p* в случае ее результативности хранилась только величина *H(p)*+1, а нулевое значение соответствовало ничейному результату. После этого преобразованные данные сжимались и записывались на диск.

Для хранения полученных таблиц был разработан формат данных, поддерживающий разбиение больших таблиц на несколько файлов, быстрый доступ к сжатому блоку по номеру блока, а также контроль целостности каждого блока.

С целью улучшить распределение нагрузки формула вычисления индекса для каждой позиции была изменена так, чтобы позиции с одинаковым расположением королей оказались «перемешаны» между узлами: новый индекс позиции вычислялся по формуле $I' = I \bmod 2048$, где *I* — ее старый индекс.

Испытания описанной реализации на суперкомпьютере IBM Blue Gene/P, установленном на факультете вычислительной математики и кибернетики

Московского государственного университета имени М. В. Ломоносова, дали следующие результаты при решении подзадачи «король, ферзь и ладья против короля, ферзя и слона» изображены на диаграмме 1 и рисунке 1.

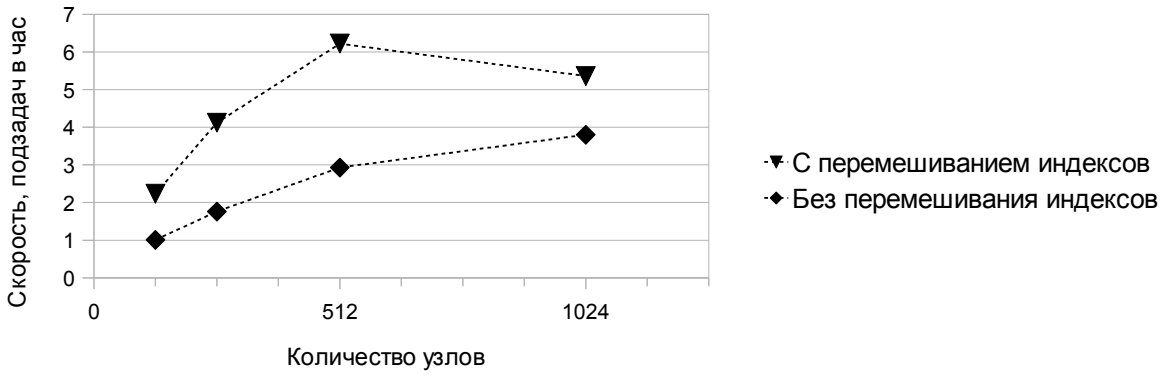

Диаграмма 1. Сравнение скоростей решения подзадачи «король, ферзь и ладья против короля, ферзя и слона», суперкомпьютер IBM Blue Gene/P

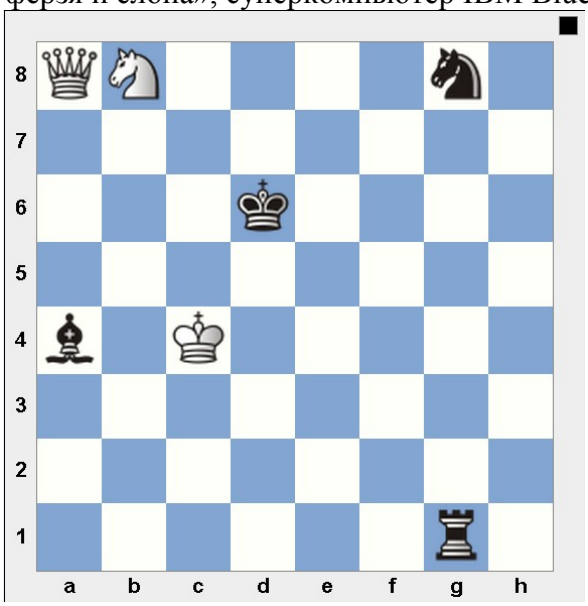

Рисунок 1. Черные начинают и не могут избежать мата в 545 ходов.

В четвертой главе диссертации предложена схема балансировки нагрузки для параллельного алгоритма ретроанализа. Описаны результаты, полученные с применением этой схемы.

С целью улучшения равномерности распределения нагрузки по вычислительным узлам была разработана и реализована схема балансировки нагрузки для параллельного алгоритма ретроанализа.

При разработке такой схемы балансировки в рамках данной работы к ней предъявлялись три основных требования:

1) эффективность перераспределения нагрузки: обработка данных на другом вычислительном узле должна требовать как можно меньше дополнительных расходов и обращений к исходному вычислительному узлу;

2) малая дополнительная нагрузка на систему: все действия, связанные с балансировкой нагрузки, не должны отнимать большое количество вычислительных ресурсов;
3) масштабируемость: схема балансировки должна эффективно работать при большом количестве вычислительных узлов.

Разработанная схема балансировки выглядит следующим образом.

Все вычислительные узлы разбиваются на группы небольшого размера G (при испытаниях был выбран размер группы G=32 узла); назовем эти группы группами нулевого уровня. В каждой группе назначается менеджер группы — один из вычислительных узлов из этой группы. Каждый вычислительный узел периодически (один раз в q секунд; при испытаниях было выбрано значение q=1) сообщает менеджеру группы о своей текущей загрузке. Менеджер группы с той же периодичностью присылает каждому участнику группы данные о загрузке всех остальных участников. Таким образом, каждый вычислительный узел имеет актуальную информацию о загрузке всех узлов внутри своей группы.

Менеджеры групп нулевого уровня также объединяются в группы — группы первого уровня, менеджеры групп первого уровня — в группы второго уровня, и так далее. Менеджеры групп уровня n ведут себя по отношению к менеджеру группы уровня n+1 так же, как и вычислительные узлы по отношению к менеджерам групп нулевого уровня: периодически отсылают ему информацию о средней загрузке своей подопечной группы (средняя загрузка вычисляется как среднее арифметическое среди всех величин загрузки в группе) и получают взамен информацию о средней загрузке в подопечных группах других менеджеров, входящих в ту же группу уровня n+1. В результате образуется древовидная структура, в которой каждый узел знает о средней загрузке своих соседей благодаря обмену данными с родительским узлом. Для каждого уровня групп n назначается нижний порог загрузки D_n — минимальное значение средней загрузки группы этого уровня, при котором еще возможно перераспределение задач с вычислительных узлов этой группы на другие узлы.

Когда вычислительный узел A выполняет практически все задачи, которые он должен был выполнить на текущей итерации (количество невыполненных задач на узле падает ниже некоторого значения Z), он начинает действия по балансировке нагрузки. Поскольку у вычислительного узла есть копия данных о загрузке других узлов из его группы, он может выбрать в своей группе наиболее загруженный узел B и отправить ему запрос на выдачу задания.

Если узел A обнаруживает, что все загрузка всех узлов в его группе ниже, чем нижний порог загрузки D_0 , то он отправляет менеджеру своей группы сообщение: «в группе нет задач».

В момент, когда некоторый менеджер группы (назовем его M) получает от подчиненного узла сообщение «в группе нет задач» или же самостоятельно

определяет, что в его подопечной группе загрузка всех узлов упала ниже порогового значения, он начинает действия по балансировке между группами. Поскольку менеджер M располагает данными о средней загрузке всех менеджеров из одной с ним группы, он может выбрать наиболее загруженного из них (назовем его N) и отправить ему сообщение «подключение к группе». После этого менеджер N начинает регулярно отправлять копию информации о загрузке внутри своей группы менеджеру M, который по сути становится дублером менеджера N. Менеджер M, в свою очередь, распространяет полученные данные среди узлов своей группы, которые могут выбирать наиболее загруженные узлы из группы менеджера N и отправлять им соответствующие запросы.

Если менеджер M обнаруживает, что загрузка всех менеджеров из одной с ним группы упала ниже заданного порогового значения, он отправляет своему менеджеру (родительской вершине в дереве балансировки) сообщение «в группе нет задач», и там выполняются все те же описанные действия.

В случае, когда самый старший менеджер, соответствующий корневой вершине дерева балансировки, получает сообщение «в группе нет задач» (или самостоятельно определяет эту ситуацию), он рассылает своей группе сообщение «останов балансировки», которое затем передается всеми менеджерами групп дальше вниз по дереву — вплоть до листьев дерева балансировки. Узлы, получившие сообщение «останов балансировки», больше не предпринимают никаких действий по перераспределению нагрузки, а после обработки всех своих задач приступают к завершению текущей итерации.

Для выбора значений D_i на всех уровнях балансировки, порога начала балансировки Z, а также для оценки эффективности разработанной схемы была построена формальная модель работы системы, выполняющей балансировку в соответствии с этой схемой.

Пусть P — количество вычислительных узлов системы, t — среднее время исполнения одной задачи (перераспределяемыми задачами в рассматриваемом алгоритме являются задачи определения всех «обратных» ходов и рассылка пометок вдоль них; для подсчета количества «прямых» ходов из состояния игры используется статическая балансировка), а l_i — количество задач, изначально назначенных на вычислительный узел i в рамках одной итерации. Рассмотрим перераспределение задач между двумя вычислительными узлами r и s, $l_s = l_r + D$, $D > 0$ (начальная загрузка узла s («отправляющего» задачи) выше, чем узла r («принимающего» узла)). Пусть M — количество задач, перераспределяемых за один запрос на выдачу задания, тогда для того, чтобы такое перераспределение приводило к уменьшению общего времени работы системы, то есть чтобы время работы каждого из этих узлов после перераспределения было меньше, чем время работы узла s при отсутствии балансировки, необходимо, чтобы выполнялось неравенство

$$\max\left(M \cdot t + \tau_r, (D - M) \cdot t + \tau_s\right) < D \cdot t,$$

где τ_r и τ_s — среднее время, затрачиваемое на одно перераспределение задач принимающим и отправляющим узлом соответственно. Поскольку максимум из двух величин не менее их среднего арифметического, то получаем

$$D > \frac{\tau_r + \tau_s}{t}.$$

Отметим, что для минимизации времени простоя принимающему узлу следует начинать действия по балансировке в тот момент, когда количество Z невыполненных задач, оставшихся на этом узле, все еще достаточно для того, чтобы узел оставался загружен, пока происходит доставка и обработка запроса на выдачу заданий: $Z = \frac{\tau_r}{t}$.

С учетом того, что менеджеры групп получают информацию о загрузке узлов группы с периодичностью q секунд, погрешность при определении уровня загрузки в группах нулевого уровня можно оценить как $\frac{q + 2\omega}{t}$, где ω — время передачи информации о загрузке между двумя узлами (от менеджера группы к вычислительному узлу). Тогда в качестве нижнего порога загрузки в группе нулевого уровня можно выбрать

$$D_0 = \frac{\tau_s + \tau_r + q + 2\omega}{t}.$$

Для группы более высокого уровня h погрешность при определении загрузки внутри этой группы (с учетом того, что загрузка группы уровня h-1 вычисляется как среднее арифметическое загрузки ее членов) составит порядка $\frac{q + (h+2)\cdot\omega}{t}$, что позволяет использовать формулу

$$D_h = \frac{\tau_s + \tau_r + q + (h+2)\cdot\omega}{t}$$

для всех уровней h. При этом следует отметить, что благодаря запросам «подключение к группе» с некоторого момента времени часть пересылок сообщений об уровне загрузки начинает производиться напрямую, минуя менеджеры верхних уровней, что уменьшает погрешность.

Для оценки эффективности разработанной схемы балансировки оценим суммарное время, затрачиваемое вычислительными узлами системы в течение одной итерации (с учетом времени простоя). В худшем для балансировки случае все группы уровня h, кроме одной, простаивают в течение времени, необходимого группе для обработки нагрузки, равной D_h, что дает максимальное суммарное время простоя на уровне h равным $D_h \cdot t \cdot G^{h-1} \cdot (G-1)$, где G — размер группы, а общее время простоя узлов системы на всех уровнях имеет оценку

$$I = \sum_{h=1}^{H} D_h t G^{h-1} (G-1),$$

где H — высота дерева балансировки, $H \approx \log_G P$. Обозначив общее количество задач на итерации $\sum_{i=1}^{P} l_i$ символом L, среднее время, затрачиваемое узлом на рассылку и получение информации о загрузке, — символом B, а количество запросов на перераспределение задач — символом S, получим суммарное время для всех узлов $T = B \cdot P + L \cdot t + S(\tau_r + \tau_s) + I$. При отсутствии же балансировки нагрузки это время составило бы $T_0 = L \cdot t + P \cdot t \cdot d$, где d — среднее значение дисбаланса нагрузки, вычисляемое по формуле $d = \frac{1}{P} \cdot \sum_{i=1}^{P} (l_{max} - l_i)$, l_{max} — максимальная загрузка одного узла, $l_{max} = \max_j l_j$. Тогда из $I \le D_H \cdot t \cdot (P-1), S \approx \frac{d \cdot P}{2M}$ и из того, что на практике $\tau_r + \tau_s$ много меньше $2 \cdot M \cdot t$, получим, что выигрыш по времени достигается схемой балансировки уже при $d > \frac{B}{t} + D_H$. Отклонение же по сравнению с идеальной балансировкой составит

$$T - T_{ideal} = B \cdot P + S(\tau_r + \tau_s) + I \le P \cdot (B + \frac{d}{2M}(\tau_r + \tau_s) + D_H \cdot t) \ ,$$

а время работы одной итерации, таким образом, составит на $B + \frac{d}{2M}(\tau_r + \tau_s) + D_H \cdot t$ секунд больше, чем в случае идеальной балансировки.

Описанная схема балансировки была реализована и испытана на задаче построения семифигурных эндшпильных таблиц для игры в шахматы. Полученные результаты представлены на диаграмме 2 (для наглядности на диаграмме изображены также результаты испытаний без балансировки нагрузки).

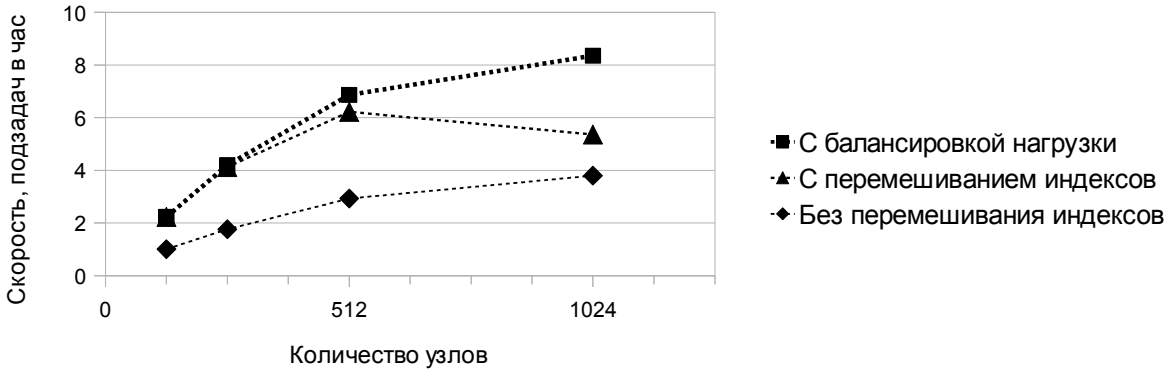

Диаграмма 2. Сравнение скоростей решения подзадачи «король, ферзь и ладья против короля, ферзя и слона», суперкомпьютер IBM Blue Gene/P

Результаты проведенных испытаний подтвердили эффективность разработанной схемы балансировки.

В заключении представлены основные результаты диссертации.

Предложен параллельный алгоритм ретроанализа, предназначенный для эффективного решения задач, решаемых методом ретроанализа, на суперкомпьютерных системах без общей памяти и нивелирующий имеющиеся для последовательного алгоритма препятствия к работе в таких системах. Исследована масштабируемость описанного алгоритма, предложены способы ее улучшения.

С целью улучшения масштабируемости алгоритма разработана схема балансировки нагрузки для параллельного алгоритма ретроанализа, рассчитанная на использование в системах с большим количеством узлов. Построена формальная модель выполнения балансировки нагрузки в соответствии с разработанной схемой, получены оптимальные значения параметров и оценка эффективности схемы.

На основе разработанного параллельного алгоритма ретроанализа и схемы балансировки нагрузки реализован программный комплекс, позволяющий эффективно решать задачи построения оптимальной стратегии для пошаговых игр двух противников с полной информацией методом ретроанализа на суперкомпьютерных системах без общей памяти и демонстрирующий хорошую масштабируемость при использовании более 4000 процессорных ядер.

На базе разработанного программного комплекса реализован и испытан алгоритм решения задачи игры в шахматы для позиций, в которых имеется не более семи фигур. Впервые в мире получена оптимальная стратегия для многих классов шахматных позиций. Опыт использования предложенного алгоритма показал его практическую значимость.

Список публикаций

1. Махнычев В.С. Распараллеливание сложных интеллектуальных задач, основанных на обходе дерева перебора. // Программные системы и инструменты. Тематический сборник № 5. — М.: Изд-во факультета ВМК МГУ. — 2005. — С. 81—93.

2. Гуляев А.В., Махнычев В. С. Об одном подходе к реализации метода ветвей и границ на распределенной системе. // Вестник Московского университета. Серия 15. Вычислительная математика и кибернетика. — 2008. — №3. — С. 60—63.

3. Махнычев В.С., Ильичев А.Б. Распараллеливание задачи о составлении вузовского расписания с использованием распределенного обхода дерева перебора. // Программные системы и инструменты. Тематический сборник № 8. — М.: Изд-во факультета ВМК МГУ, 2007. — С. 37—42.

4. Захаров В.Б., Махнычев В.С. Распараллеливание алгоритма ретроанализа для задач сверхбольшой размерности. Получение точного решения для 7-фигурных шахматных окончаний. // Отчет по использованию суперкомпьютеров МГУ в 2009-2010 гг. — М., МГУ, 2010. — С. 20—22.

5. Захаров В.Б., Махнычев В.С. Алгоритм ретроанализа в суперкомпьютерных системах на примере задачи игры в шахматы. // Программные системы и инструменты. Тематический сборник № 11. — М.: Изд. отделения ф-та ВМК МГУ, 2010. — С. 45—52.

6. Захаров В.Б., Махнычев В.С. Об опыте создания российского национального сервера «Шахматная планета». // Программные системы и инструменты. Тематический сборник № 6. — М.: Изд.отделения ф-та ВМК МГУ, 2005г. — С. 49—51.

7. Захаров В.Б., Махнычев В.С. Особенности вычислительного практикума на суперкомпьютерах. // Материалы Международной научно-практической конференции «Информатизация образования – 2011» ЕГУ, Елец, 2011г. — С. 117—120.