Параллельный алгоритм рекурсивной обработки динамических древовидных структур данных на примере усеченного квадродерева

Е.А. Юсов

Предложен метод эффективной параллелизации рекурсивных алгоритмов обработки динамической нерегулярной структуры усеченного квадродерева. В основе реализации лежит определение на предварительном этапе для узлов верхних уровней дерева специальных величин, характеризующих вычислительную сложность обработки поддерева, растущего из данного узла. Анализатор сложности составляет список стартовых узлов для каждого потока исполнения с заданным допуском на неравномерность распределения вычислительной нагрузки. Представлен метод организации данных по памяти, увеличивающий эффективность использования процессорного КЭШа. Приведены результаты экспериментов, подтверждающие эффективность описанного подхода.

1. Введение

В настоящее время стратегия увеличения мощности вычислительных устройств методом наращивания частоты, преобладавшая в компьютерных технологиях до последних лет, достигла технологического предела, и производители были вынуждены перейти на путь увеличения числа вычислительных блоков. Здесь можно отметить как увеличение числа ядер, расположенных на одном кристалле (двух и четырехядерные процессоры уже давно стали стандартом, а в ближайшее время появятся и 8-ядерные), так и значительный рост числа процессоров, входящих в кластерные системы и суперкомпьютеры. При этом однопроцессорные машины, а вместе с ними и программы, основанные на последовательном исполнении единственного потока команд, постепенно отходят в прошлое. Сейчас эффективная реализация практически любого алгоритма должна обеспечивать по возможности наиболее равномерную загрузку всех вычислительных блоков системы, что выдвигает новые требования как при разработке новых алгоритмов, так и при переложении существующих на современные аппаратные средства. А задача параллелизации вычислений стала особенно актуальной не только для исследователей, решающих сверхсложные задачи на мощных вычислительных системах, но также и для разработчиков многих других приложений (математических пакетов, редакторов изображения и видео, архиваторов, компьютерных игр и т.п.), которые выполняются на персональных компьютерах.

Иерархические структуры данных и рекурсивные алгоритмы их обработки весьма распространены и находят применение в широком классе разнородных областей, среди которых сжатие информации, распознавание образов, многомасштабное моделирование и пр. Построение высокоэффективных параллельных решений для таких алгоритмов, ввиду вышесказанного, в настоящее время является активно исследуемой областью [1, 4, 6, 8, 9].

2. Постановка задачи

Довольно часто встречается задача, связанная с эффективной обработкой динамически меняющихся во времени иерархических структур данных. В качестве примера может служить описание адаптивной триангуляции поверхности рельефа усеченным квадродеревом, где участкам с грубой детализацией соответствуют узлы дерева из низших уровней, а областям в высоком разрешением – узлы из высших уровней [5, 7]. При перемещении камеры в пространстве уровни детализации модели должны быть соответствующим образом изменены, для чего в процессе подготовки каждого кадра необходимо выполнять обход усеченного дерева. При отсутствии скачков в траектории движения камеры (что является наиболее распространенным случаем) уровень детализации от кадра к кадру, и, следовательно, структура дерева, меняется незначительно. В таких случаях говорят о высокой когерентности кадров. Для эффективной

реализации подобных алгоритмов на современных многоядерных процессорах и удовлетворения требований систем визуализации реального времени таких приложений, как авиасимуляторы, видеотренажеры, компьютерные игры и др. необходимы методы их параллелизации.

3. Обзор близких по тематике работ

Решение схожей задачи, а именно метод параллельного обхода вглубь дерева поиска для решения задачи о рюкзаке предложен В.А.Беляевым и Н.Е.Тимошевской в работе [1]. В основе метода лежит динамическое распределение поддеревьев дерева поиска между рабочими процессами, управляемое специально выделенным для этого процессом. Процессы, закончившие обход своего поддерева, подключаются «в помощь» к другим процессам, при этом им для обработки выделяется поддерево в дереве занятного процесса. Параллельные методы обхода *k*-ичных деревьев, деревьев поиска сочетаний и оптимального назначения, ориентированные на использование в многопроцессорных системах кластерного типа, описаны Н.Е.Тимошевской в статье [4]. В работе предложены два подхода. Первый предполагает разбиение всего дерева на множество небольших поддеревьев, число которых намного превосходит число процессоров, и, как и в [1], выделение специального диспетчерского процесса, распределяющего нагрузку между остальными, рабочими процессами в порядке их освобождения. При этом все процессы работают независимо, каждый над своим поддеревом до тех пор, пока поддерево не будет полностью обработано. Преимуществом такого подхода является сокращение взаимодействий между процессами, однако балансировка нагрузки оказывается не всегда равномерной. Второй алгоритм является обобщением представленного ранее в [1] и реализует динамическое перераспределение задач между процессами, при этом разбиение на подзадачи происходит не сразу, а по мере выполнения обхода. Алгоритм обеспечивает более равномерную балансировку вычислительной нагрузки за счет более интенсивного обмена данными между процессами. Представленный в настоящей работе подход обладает преимуществами обоих из представленных в [1] методов: он позволяет добиться практически оптимальной балансировки нагрузки при полном отсутствии взаимодействия между рабочими процессами.

Решение, близкое к предлагаемому в данной работе, приведено Г.И.Малашонком и Ю.Д.Валеевым в статье [6]. Авторами разработана схема организации параллельных вычислений для рекурсивных символьно-численных алгоритмов. В основе метода лежит создание «темпорированного» дерева алгоритма, у которого входные данные и результаты вычислений находятся в корневой вершине, а вычислительный процесс развивается от корня к листьям, а затем обратно. Для построенного дерева задача распараллеливания сводится к разворачиванию этого графа на кластере с закреплением поддеревьев за отдельными процессорами. Как и в [1] и [4], в [6] предусмотрен специальный процесс-планировщик, который динамически перераспределяет задачи от наиболее загруженных процессоров к свободным.

Еще один метод динамического распределения нагрузки между процессами применяется Г.Б.Сушко и С.А.Харченко в [8] для решения систем линейных уравнений, описываемых неструктурированными разреженными матрицами большой размерности. Рекурсивно-параллельный алгоритм решения задачи о ближайшей паре приведен В.Н.Терещенко в [9].

Ни один из перечисленных выше подходов не предназначен для обработки динамически меняющихся структур данных, кроме того, перечисленные методы, как правило, показывают невысокую эффективность на небольшом числе параллельных потоков (4-8), типичном для современных многоядерных процессоров. Поставленная же задача состоит в поиске эффективного способа параллелизации рекурсивных алгоритмов обхода динамически изменяющихся деревьев, обеспечивающего эффективную реализацию на многоядерных процессорах.

4. Описание алгоритма

4.1. Вычислительный вес поддерева

Одним из возможных вариантов решения поставленной задачи является предварительный последовательный обход усеченного дерева с накоплением в некотором массиве индексов уз-

лов, которые должны быть обработаны, разделение полученного списка на равные части и обработка каждой части в отдельном потоке исполнения. Хотя сам процесс обработки при таком подходе оказывается идеально сбалансированным, предварительный этап составления списка, как показали эксперименты, занимает существенную часть времени, и, кроме того, требуются значительные затраты памяти для хранения индексов.

Наиболее эффективным способом параллелизации алгоритма представляется запуск рекурсивного обхода на каждом вычислительном ядре таким образом, чтобы нагрузка была распределена как можно более равномерно. С этой целью для каждого потока исполнения составляется список узлов, с которых вместо корневой вершины начинается обработка поддеревьев. В отличие от подходов [1, 4, 6], распределение нагрузки выполняется на предварительном этапе, а не динамически в процессе обхода дерева, что позволяет исключить передачу данных между потоками и повышает тем самым эффективность параллелизации.

Вычислительная сложность обработки каждого поддерева определяется текущей структурой усеченного дерева (динамически изменяющейся от кадра к кадру). Чтобы обеспечить возможность равномерного распределения нагрузки предлагается узлам дерева приписывать величину, характеризующую сложность обработки всех узлов поддерева, растущего из данной вершины – *вычислительный вес поддерева* (Рис. 1). Обозначим через $w_{i,k}^{(j)}$ вес поддерева, растущего из узла с индексами (i,k) и лежащего в уровне j, а через $c_{i,k}^{(j)}$ – вычислительные затраты, связанные с обработкой только данного узла. Для листового узла усеченного дерева вес $w_{i,k}^{(j)}$ определяется непосредственно как $c_{i,k}^{(j)}$. Для всех нелистовых узлов значение $w_{i,k}^{(j)}$ определяется рекуррентно как сумма собственной вычислительной сложности и суммарного вычислительного веса всех потомков:

$$w_{i,k}^{(j)} = \begin{cases} c_{i,k}^{(j)}, & \text{для листовой вершины} \\ c_{i,k}^{(j)} + \sum\limits_{m,n=\overline{0,1}} w_{2i+m,2k+n}^{(j+1)}, & \text{для нелистовой вершины} \end{cases} \quad (1)$$

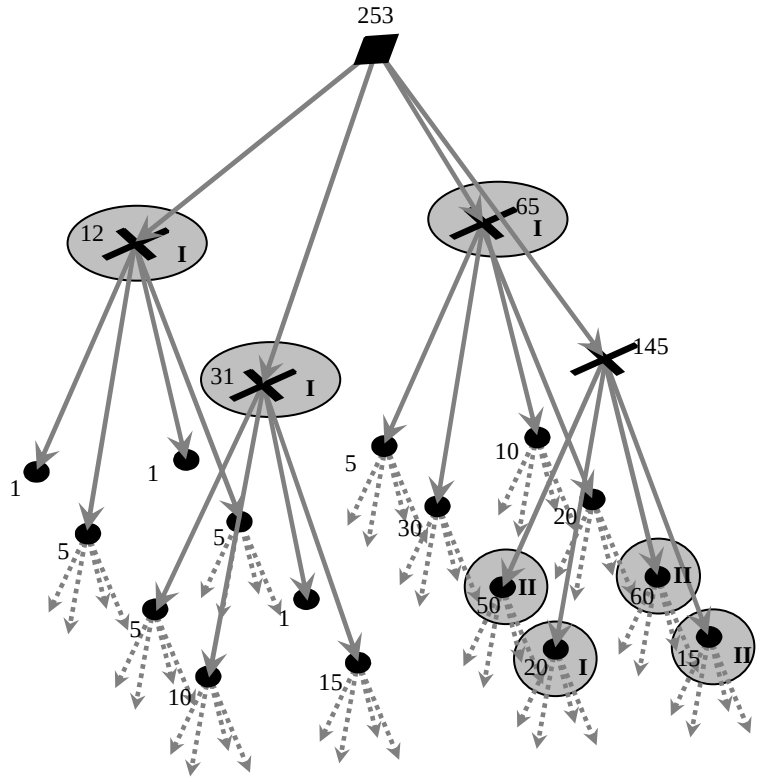

Рис. 1. Пример вычислительной нагрузки, приписанной узлам квадродерева и распределения стартовых узлов между двумя потоками исполнения (I и II)

Вес $w_{0,0}^{(0)}$ корневой вершины дерева будет соответствовать сложности обработки всего дерева. Тогда при наличии *N* ядер в системе средняя вычислительная сложность $\tilde{W}$, приходящаяся на одно ядро, будет определяться по формуле:

$$\tilde{W} = w_{0,0}^{(0)} / N \tag{2}$$

Чтобы оптимально распределить нагрузку между ядрами, необходимо составить *N* списков S_q индексов стартовых узлов:

$$S_q = \{(i_1^q, k_1^q, j_1^q),...,(i_{M_q}^q, k_{M_q}^q, j_{M_q}^q)\},\ q = \overline{1,N} \tag{3}$$

таких, что суммарная сложность обработки всех узлов в каждом списке, обозначаемая W_q и определяемая по формуле:

$$W_q = \sum_{t=1}^{M_q} w_{i_t^q, k_t^q}^{(j_t^q)} \tag{4}$$

была максимально близка к среднему значению $\tilde{W}$:

$$|W_q - \tilde{W}| < \delta w_{0,0}^{(0)},\ q = \overline{1,N} \tag{5}$$

где величина δ определяет допуск на неравномерность распределения вычислительной нагрузки между ядрами и для удобства определяется как доля вычислительного веса всего дерева.

Введем дополнительно величину η, характеризующую максимальную степень неравномерности распределения веса по ветвям дерева:

$$\eta = \max_{\substack{i,k,j \\ m,n=\overline{0,1}}} \left(\frac{w_{2i+m,2k+n}^{(j+1)}}{w_{i,k}^{(j)}} \right) \tag{6}$$

Из определений (1) и (6) следует, что $\eta \le 1$. Для полного сбалансированного квадродерева $\eta = 1/4$, что означает полностью равномерное распределение вычислительной сложности по поддеревьям. Из (6) непосредственно вытекает, что

$$\forall i,k \rightarrow w_{i,k}^{(j)} \le w_{0,0}^{(0)} \eta^j$$

Тогда если

$$j \ge j^* = [\log_\eta \delta] + 1, \tag{7}$$

где через $[\bullet]$ обозначена операция взятия целой части, то

$$w_{i,k}^{(j)} \le \delta w_{0,0}^{(0)} \tag{8}$$

Формулы (7) и (8) означают, что при рекурсивном спуске вглубь по дереву вычислительный вес любого узла на уровне $j \ge j^*$ будет гарантированно меньше заданного допуска на неравномерность распределения. Этот факт играет важную роль для алгоритма распределения стартовых узлов дерева, рассматриваемого ниже.

4.2. Алгоритм составления списков стартовых узлов

Код на языке C++ рекурсивного алгоритма, составляющего списки S_q, приведен на Рис. 2.

```
void RecursiveDistributeNodes (int i, int k, int j)
{
    int CurrSubTreeCompWeight = g_ppCompWeights[j][i + (k<<j)];

    //Проверяем, будет ли нарушено условие (5), если добавить текущий
    //узел в формируемый список
    if( CurrSubTreeCompWeight + g_iCurrThreadCompLoad <
        g_iAverageCompLoad + g_iCompLoadDistributionAccuracy )
    {
        //Добавляем узел в список стартовых узлов текущего потока
        g_ppThreadRootNodesList[g_iCurrThread][g_iNumRootNodes]
             = PackIKJ(i,k,j);
```

```
        g_iNumRootNodes++;
        g_iCurrThreadCompLoad += CurrSubTreeCompWeight;
        if( g_iCurrThreadCompLoad >= g_iAverageCompLoad )
        {
            //Суммарная вычислительная нагрузка текущего потока превысила
            //среднюю -> начинаем заполнение списка следующего потока
            g_ppThreadRootNodesList[g_iCurrThread][g_iNumRootNodes]
                  = 0xFFFFFFFF;
            g_iCurrThreadCompLoad = 0;
            g_iNumRootNodes = 0;
            g_iCurrThread++;
        }
    }
    else
    {
        //Вычислительная нагрузка данного узла слишком большая, чтобы
        //быть добавленной в список -> переходим к дочерним узлам
        RecursiveDistributeNodes (i*2,   k*2,   j+1);
        RecursiveDistributeNodes (i*2+1, k*2,   j+1);
        RecursiveDistributeNodes (i*2,   k*2+1, j+1);
        RecursiveDistributeNodes (i*2+1, k*2+1, j+1);
    }
}
```

Рис. 2. Рекурсивный алгоритм, составляющий списки стартовых узлов

В приведенном алгоритме массив `g_ppCompWeights` содержит вычислительные веса узлов дерева, а выражение `g_ppCompWeights[j][i+(k<<j)]` позволяет получить вычислительную нагрузку $w_{i,k}^{(j)}$. Переменная `g_iCurrThread` отражает номер q текущего потока, в список которого добавляются узлы, а переменная `g_iCurrThreadCompLoad` накапливает суммарную вычислительную нагрузку W_q, приходящуюся на текущий поток. Двумерный массив `g_ppThreadRootNodesList` реализует списки S_q стартовых узлов, значение `0xFFFFFFFF` отмечает конец очередного списка. Функция `PackIKJ()` выполняет запаковку трех индексов *i,k,j* узла в одно 32-битное целочисленное значение для экономии памяти. Переменная `g_iAverageCompLoad` содержит среднюю нагрузку $\tilde{W}$. Точность распределения нагрузки между ядрами контролируется переменной `g_iCompLoadDistributionAccuracy` (хранящей значение $\delta w_{0,0}^{(0)}$).

В процессе работы алгоритм накапливает в списке S_q каждого потока узлы до тех пор, пока общая вычислительная сложность W_q всех узлов в списке не превысит среднюю сложность `g_iAverageCompLoad` ($\tilde{W}$), но не более, чем на величину $\delta w_{0,0}^{(0)}$, хранящуюся в переменной `g_iCompLoadDistributionAccuracy` (что будет гарантировать выполнение условия (5)). Алгоритм рекурсивно спускается вниз от корня дерева. При этом текущий узел добавляется в формируемый список, если не нарушается условие (5), в противном случае выполняется дальнейший спуск вниз по дереву, где возможно более точное распределение. При этом из (7) и (8) следует, что для того, чтобы выполнить условие (5), алгоритм никогда не будет спускаться ниже, чем до уровня $j = j^*$, так как на этом уровне любой узел гарантированно будет добавлен в список.

После того, как общая нагрузка одного потока превысит среднюю (при этом условие (5) будет выполнено), алгоритм переходит к заполнению списка следующего ядра. При таком методе составления списков последний поток исполнения получит меньшую нагрузку, чем все остальные. Однако, это частично компенсируется тем, что он будет запущен в последнюю очередь, когда остальные уже начнут свою работу.

Нерешенным остался только вопрос о получении вычислительных весов $w_{i,k}^{(j)}$ узлов дерева перед запуском алгоритма распределения стартовых узлов. Как уже отмечалось, при высокой когерентности кадров структура дерева от кадра к кадру меняется незначительно. Поэтому в качестве весов может использоваться информация, накопленная при обработке дерева на предыдущем кадре. При таком подходе вычисление весов $w_{i,k}^{(j)}$ может выполняться одновременно с рекурсивным обходом дерева основным алгоритмом, при минимальном его усложнении (эксперименты показывают потерю производительности менее 1%).

4.3. Обсуждение алгоритма

При идеально точном распределении ($\delta \to 0$) нагрузка на каждый поток исполнения будет максимальна близка к среднему значению $\tilde{W}$, однако это приведет к необходимости обхода всего дерева, что потребует слишком много времени, а существенного выигрыша в производительности не принесет. Для $N \le 8$ значение $\delta = 0.01$ позволяет сбалансировать нагрузку на ядра практически равномерно, при этом, как показали эксперименты, в списке стартовых узлов каждого потока оказывается в среднем только 6 элементов ($M_q \approx 6$).

Из принципа работы алгоритма, а также выражений (7) и (8) следует, что при любой глубине исходного дерева, заданном допуске на неравномерность распределения δ и известной величине η, в списках стартовых узлов S_q потоков будут присутствовать только узлы, лежащие в уровнях квадродерева от 0 до j^*. Главное заключение, которое из этого вытекает, состоит в том, что время работы алгоритма распределения не зависит от глубины исходного дерева, а определяется только точностью распределения нагрузки δ.

Другой вывод, который отсюда следует, состоит в том, что хранить вычислительную сложность всех узлов дерева нет необходимости; это достаточно делать только для уровней от 0 до j^*-1. Например, для задачи построения адаптивной триангуляции рельефа [5, 7] эмпирически установлено, что $\eta = 1/2$. Поэтому при $\delta = 0.01$ $j^* = [\log_{0.5} 0.01] + 1 = 7$. Т.е. вычислительные веса достаточно хранить только для самых грубых 7 уровней иерархии. Дополнительные затраты памяти оказываются в этом случае весьма незначительными. Так, в случае, если дерево имеет 11 уровней, на хранение всех вычислительных весов для уровней дерева от 0 до 6 требуется только 5489 элементов, что составляет менее 0.1% от общих затрат памяти. Веса узлов всех остальных уровней определяются в процессе рекурсивного обхода дерева основным алгоритмом и хранятся временно на стеке.

На Рис. 1 приведен пример того, как будут распределены стартовые узлы между двумя потоками для изображенного дерева. Общая вычислительная нагрузка на первый поток будет равна $W_1 = 31 + 12 + 65 + 20 = 128$, на второй – $W_2 = 50 + 15 + 60 = 125$. Соответствующие списки стартовых узлов для потоков будут следующими: $S_1 = \{(0,0,1), (0,1,1), (1,1,1), (2,0,2)\}$ $S_2 = \{(2,1,2), (3,0,2), (3,1,2)\}$.

Следует еще раз отметить, что ранее представленные методы [1, 4, 6] основаны на выделении диспетчерского процесса, что связано с необходимостью решения нескольких проблем, снижающих эффективность параллельного алгоритма, таких как определение процесса, задача которого будет разбиваться на подзадачи; организации обмена данными между процессами; синхронизация работы процессов. В предлагаемом подходе благодаря наличию данных, накопленных при предыдущем обходе дерева, возможно вначале один раз выполнить очень быструю процедуру распределения нагрузки, время работы которой несущественно по сравнению со временем работы основного алгоритма. Это дает возможность всем потокам выполнять свою часть работы абсолютно независимо, при отсутствии обмена данными.

5. Реализация предложенного алгоритма

Важную роль при эффективной реализации алгоритмов играет организация доступа к памяти. Архитектурные особенности современных процессоров таковы, что если одно ядро про-

изводит запись в некоторую линию КЭШа, а другое ядро выполняет чтение этого же участка памяти, возникает необходимость синхронизации КЭШей, на что требуется чрезвычайно много времени. Эти затраты настолько велики, что, даже при идеальном распределении нагрузки между ядрами и, одновременно, неэффективной организации работы с памятью, параллельная версия алгоритма может выполняться даже медленнее (!), чем последовательная. Чтобы максимально увеличить эффективность работы ядер предлагается упорядочивать данные вершин согласно рекурсивному обходу дерева вглубь, при этом для еще большей эффективности четыре элемента, имеющие общего предка, предлагается размещать подряд. Представленный подход близок к предложенному в [3] методу обхода отсчетов изображения в порядке обхода двоичного дерева без физической реорганизации исходных данных в древовидные структуры. Если обозначить данные узла с индексами (*i*,*k*), лежащего в уровне *j* через $v_{i,k}^{(j)}$, то предлагаемое их расположение в памяти будет следующим:

$v_{0,0}^{(0)}$, $v_{1,0}^{(0)}$, $v_{0,1}^{(0)}$, $v_{1,1}^{(0)}$, $v_{0,0}^{(1)}$, $v_{1,0}^{(1)}$, $v_{0,1}^{(1)}$, $v_{1,1}^{(1)}$, $v_{0,0}^{(2)}$, $v_{1,0}^{(2)}$, $v_{0,1}^{(2)}$, $v_{1,1}^{(2)}$, …, $v_{2,0}^{(1)}$, $v_{3,0}^{(1)}$, $v_{2,1}^{(1)}$, $v_{3,1}^{(1)}$, …, $v_{0,2}^{(1)}$, $v_{1,2}^{(1)}$, $v_{0,3}^{(1)}$, $v_{1,3}^{(1)}$,…, $v_{2,2}^{(1)}$, $v_{3,2}^{(1)}$, $v_{2,3}^{(1)}$, $v_{3,3}^{(1)}$,….

При такой раскладке данных несколько усложняется алгоритм определения адреса, по которому расположены данные конкретного узла, но его можно вычислять непосредственно при рекурсивном обходе дерева.

Описанный в статье подход был применен для распараллеливания рекурсивных алгоритмов обработки усеченного квадродерева в решении задачи построения адаптивной триангуляции ландшафта [5, 7]. В [5, 7] приведены два рекурсивных алгоритма. Первый осуществляет перестроение усеченного квадродерева в соответствии с изменившимся положением камеры, второй выполняет построение триангуляции по полученному дереву. В качестве вычислительного веса узла для первого алгоритма принимается суммарное число раз, которое вычислялась экранная значимость при обходе поддерева, являющаяся наиболее сложной операцией (см. [5, 7]). Для алгоритма построения триангуляции весом является общее количество треугольников, которые порождаются всеми узлами, находящимися в поддереве [7]. Тестовое приложение было реализовано в среде Microsoft Visual Stuido .NET на языке C++.

6. Результаты экспериментов

Исследование эффективности предлагаемого алгоритма производилось на платформе, оснащенной двумя четырехядерными процессорами Intel® Xeon с частотой 2.66 ГГц и 8 Гб оперативной памяти. Исходная сетка высот имела размер 2048x2048 вершин и включала в себя 8380418 треугольников. Упрощение модели осуществлялось до уровня порядка 350 тыс. треугольников, что составляет около 4% от исходной сложности модели. Время работы каждого из двух рекурсивных алгоритмов, суммарное время, а также ускорение [2] приведены в таблице 1.

Таблица 1. Время работы алгоритмов при различном числе потоков.

Количество потоков	Обход дерева, мс	Построение триангуляции, мс	Общее время, мс	Ускорение
1	14.2	27.8	42.0	1.0
2	7.4	14.2	21.6	1.94
3	5.2	9.2	14.4	2.91
4	4.0	7.6	11.6	3.62
5	3.2	6.0	9.2	4.56
6	2.8	5.3	8.1	5.18
7	2.5	4.5	7.0	6.0
8	2.4	4.0	6.4	6.56

Полученное ускорение алгоритма при запуске на различном числе потоков также изображено на Рис. 3. На Рис. 4 представлен коэффициент эффективности [2] параллельной реализации.

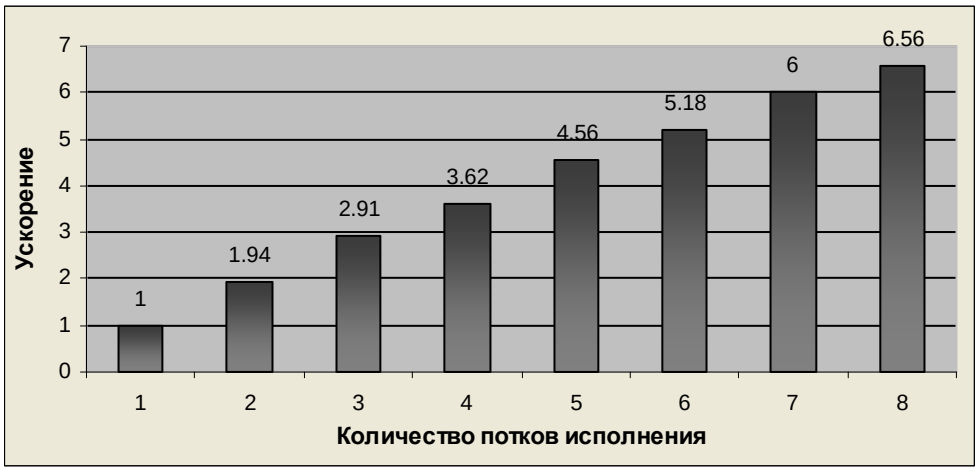

Рис. 3. Ускорение алгоритма при запуске на различном числе потоков

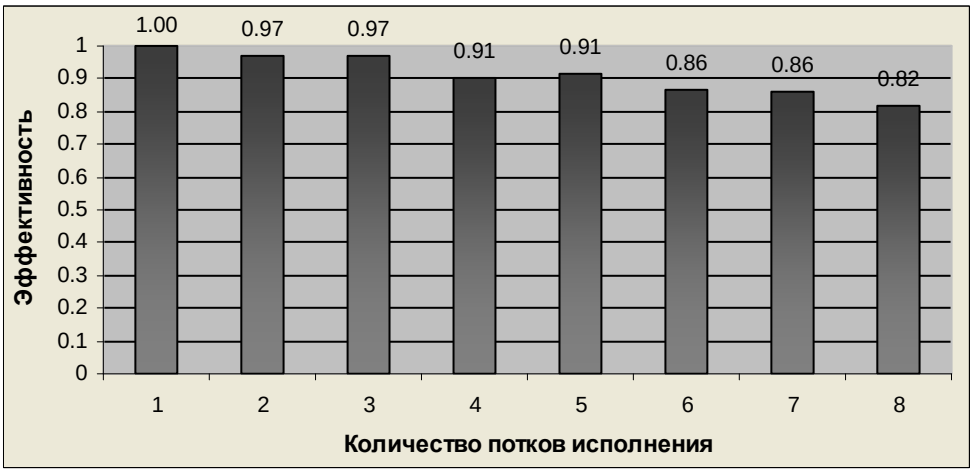

Рис. 4. Эффективность метода при запуске на различном числе потоков

Была также исследована степень дисбаланса ветвей алгоритма при помощи профилировщика потоков Intel® Thread Profiler при выполнении программы на платформе, оборудованной четырехядерным процессором Intel® Core2 Quad. Показатели снимались при движении камеры над поверхностью рельефа на большой скорости, что вызывало постоянное динамическое перестроение структуры дерева. Результаты анализа в разных масштабах времени приведены на Рис. 5 и Рис. 6.

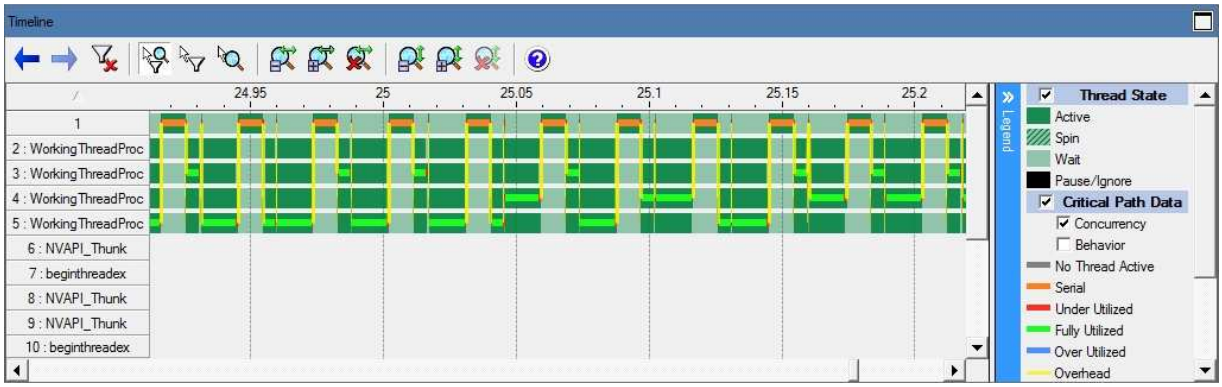

Рис. 5. Степень дисбаланса ветвей, масштаб времени 0.3 с.

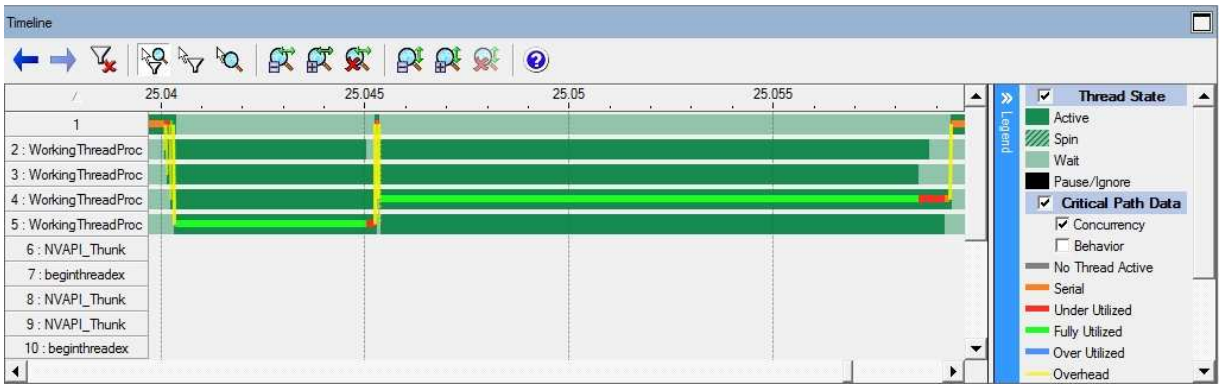

Рис. 6. Степень дисбаланса ветвей, масштаб времени 0.02 с.

Как показывают Рис. 5 и Рис. 6, дисбаланс ветвей весьма незначителен. Последовательная часть алгоритма, хорошо видная на Рис. 5, соответствует не связанным с рекурсивным обходом дерева действиям (командам отображения сцены на экране, времени, расходуемому системой DirectX, обработке сообщениям с клавиатуры и мыши и решению прочих задач). Параллельный участок, как видно из Рис. 6, состоит из двух частей – рекурсивного алгоритма обхода и перестроения дерева, точки синхронизации и рекурсивного алгоритма построения триангуляции. Как показали наши эксперименты, степень использования ядер процессора на параллельных участках составляет более 95%.

Дополнительное увеличение производительности на 10-15% достигается за счет описанного в Разделе 5 способа упаковки данных вершин квадродерева.

Проведенные вычислительные эксперименты показывают, что выбранный подход позволяет добиться высокой эффективности распределения вычислений между потоками. В частности, при числе потоков от 2 до 8 представленный алгоритм обеспечивает большую эффективность по сравнению с методами [4] на аналогичном количестве потоков. Так, согласно представленным в [4] данным, эффективность параллельного обхода полных k-ичных деревьев (которые являются наиболее простым случаем) методами [4] составляет 0.62-0.82 при выполнении на 3-8 процессах (см. для сравнения Рис. 4).

7. Заключение

В статье представлен метод эффективной параллелизации рекурсивных алгоритмов обработки нерегулярной иерархической структуры усеченного квадродерева, динамически меняющегося во времени. Реализация основана на вычислении на предварительном этапе для узлов верхних уровней иерархии дополнительных величин, характеризующих вычислительную сложность обработки поддерева, растущего из данного узла. Анализатор сложности составляет список узлов для каждого потока исполнения с заданным допуском на неравномерность распределения вычислительной нагрузки. Распределение задач выполняется на предварительном этапе, перераспределение на этапе исполнения основного алгоритма отсутствует, что позволяет полностью исключить обмен данными между различными потоками исполнения и повысить тем самым эффективность параллелизации. Показано, что при заданном допуске на неравномерность распределения при любой глубине исходного дерева в списки стартовых узлов потоков будут входить только узлы из нескольких самых грубых уровней иерархии. Поэтому время работы алгоритма распределения не зависит от глубины исходного дерева, а определяется только точностью распределения нагрузки. При высокой когерентности кадров в качестве весов узлов дерева для кадра n можно использовать значения, накопленные при обработке предыдущего кадра n-1. Алгоритм демонстрирует эффективность параллелизации от 0.97 до 0.82 при запуске на 2-8 потоках исполнения. Эксперименты показали весьма незначительный дисбаланс ветвей представленного алгоритма: степень неиспользованных вычислительных ресурсов процессора составляет менее 5%. Приведены детали организации данных по памяти, повышающих эффективность использования процессорного КЭШа, что позволяет дополнительно увеличить производительность на 10-15%. Представленный подход может быть успешно ис-

пользован для параллелизации рекурсивного обхода деревьев, имеющих произвольную структуру, к примеру, усеченных k-ичных деревьев.

Литература

1. В.А.Беляев, Н.Е.Тимошевская. Распараллеливание обхода дерева поиска для решения задачи о рюкзаке на кластерной системе // Материалы Международного научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах» с. 16-20, Нижний Новгород, 20-24 ноября, 2001.
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002, 608с.
3. Жерздев С.В. Разработка алгоритмов адаптивного сжатия видеоинформации на основе иерархических структур для задач оперативного отображения: Дис. … канд. техн. наук. Нижегородский государственный университет им. Н.И.Лобачевского, Нижний Новгород – 2004. – 113 с.
4. Н.Е.Тимошевская. Параллельные методы обхода дерева // Математическое моделирование. – №1. – 2004. – с. 105-114. Томск.
5. Е.Юсов, В.Турлапов. Динамическая оптимизация ландшафта на базе преобразования Хаара и квадродерева вершин // Труды XVI международной конференции по компьютерной графики и ее приложениям Графи-Кон’2006, с. 198-205, Новосибирск, 1-5 июля, 2006.
6. Г.И. Малашонок, Ю.Д. Валеев. Организация параллельных вычислений в рекурсивных символьно-численных алгоритмах // Труды II международной научной конференции «Параллельные Вычислительные Технологии (ПаВТ’2008)», с. 153-165, Санкт-Петербург, 28 января – 1 февраля, 2008 г.
7. Е.А.Юсов, В.Е.Турлапов. Адаптивная триангуляция ландшафта с представлением квадрадерева вершинной текстурой и вейвлет-оценкой значимости вершин // Программирование. –№5. –2008. – с. 3-18. (англоязычная версия доступна по ссылке http://www.maik.ru/contents/procom/procom5_8v34cont.htm)
8. Г.Б.Сушко, С.А.Харченко. Многопоточная параллельная реализация итерационного алгоритма решения систем линейных уравнений с динамическим распределением нагрузки по нитям вычислений // Труды II международной научной конференции «Параллельные Вычислительные Технологии (ПаВТ’2008)», с. 452-457, Санкт-Петербург, 28 января – 1 февраля, 2008 г.
9. В.Н.Терещенко. Построение рекурсивно-параллельных алгоритмов решения задач вычислительной геометрии на основе стратегии «разделяй и властвуй» // Труды II международной научной конференции «Параллельные Вычислительные Технологии (ПаВТ’2008)», с. 476-481, Санкт-Петербург, 28 января – 1 февраля, 2008 г.