УДК 004.3

Влияние временных характеристик параллелизуемых блоков последовательных алгоритмов на эффективность автоматического динамического распараллеливания в многопроцессорных системах со слабой связью

Р.И. Левченко, А.А. Судаков, С.Д. Погорелый, Ю.В. Бойко
Киевский национальный университет имени Тараса Шевченко
rmn01@mail.ru

Abstract

R.I. Levchenko, O.O. Sudakov, S.D. Pogorilyy, Y.V. Bojko .Influence of time characteristics of sequential algorithms parallel blocks on automatic dynamic parallelization efficiency in multiprocessor systems with the loosely- coupled interface. *Influence of sequential program graph characteristics on efficiency of automatic dynamic parallelizing is analyzed. Practical investigation is performed on the base of authors' system of automatic dynamic parallelizing (DDCI). On the example of two matrixes product the possible reasons of degradation of dynamic programs parallelizing efficiency are investigated. Minimal demands to parallelizable blocks of sequential programs that provide calculations efficiency optimization are made.*

Введение

Быстрое развитие многопроцессорных компьютерных систем создает предпосылки для возникновения проблем использования методов распараллеливания для облегчения написания параллельных программ. Для систем с общей памятью эта проблема является уже решенной и такие системы как OpenMP [1] являются хорошим решением проблемы. При этом для многопроцессорных компьютерных систем со слабой связью [2] эта проблема еще полностью не решена [3]. В то же время такие системы являются основными представителями на рынке суперкомпьютеров и позволяют добиваться на порядок более высокой производительности в сравнении с трудно масштабируемыми многопроцессорными компьютерными системами с общей памятью [4].

Целью данной работы является исследование проблемы блочности последовательных алгоритмов при автоматическом динамическом распараллеливании вычислений на основе DDCI-системы [5]. Используемая система автоматического динамического распараллеливания (DDCI) является авторской и уже используется для решения задач компьютерного моделирования [6,7].

Данные экспериментальные исследования влияния блочности последовательного алгоритма на эффективность автоматического динамического распараллеливания последовательной программы проводились на примере модифицированного блочного алгоритма произведения двух матриц [8]. Выбор данного алгоритма в качестве экспериментальной задачи связан со следующими преимуществами:

- Предложенная задача легко масштабируется для различного числа параллельных потоков, что позволяет проводить качественное исследование эффективности автоматического распараллеливания вычислений для многопроцессорных компьютерных систем разной размерности.
- Задача легко распараллеливается для разного количества потоков без ощутимых изменений вычислительной сложности алгоритма, данная особенность сильно упрощает анализ результатов эксперимента
- Изменение размерности блока матрицы в алгоритме позволяет гибко контролировать «эффективность чистых функций» [9] выполняющих произведение двух блоков исходных матриц.
- Задача имеет масштабируемый граф алгоритма с минимумом последовательных областей. Граф содержит большое количество чистых функций и сложные перекрестные связи между вершинами.

В работе все результаты представлены как усредненные не менее чем по 10 идентичным экспериментам. Общее время непрерывного моделирования проанализированных в работе результатов составило около 1 месяца для 32 процессорной системы.

Оптимизация экспериментальной задачи

Для эксперимента был взят алгоритм произведения двух матриц с оптимизированной нагрузкой на процессорный кэш. Для оптимизации использовалось двойное разбиение перемножаемых матриц на блоки. Фактически при обычном разбиении (рис. 1) параллельно перемножаются блоки, помеченные на рисунке серыми прямоугольниками.

Для изменения характеристик параллелизма в алгоритме при разбиении его на блоки приходится менять размеры перемножаемых блоков, а это, в свою очередь, сильно сказывается на эффективности работы процессорного кэша.

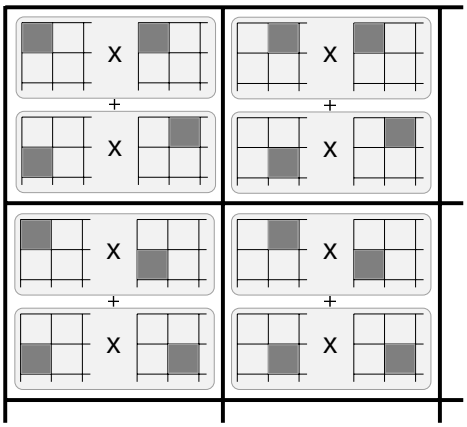

Рисунок 1 – Стандартная реализация алгоритма блочного произведения двух матриц.

Двойное разбиение (рис. 2) основано на разделении параллельно перемножаемых блоков из предыдущего подхода на более мелкие последовательно перемножаемые подблоки, которые имеют уже фиксированные размеры.

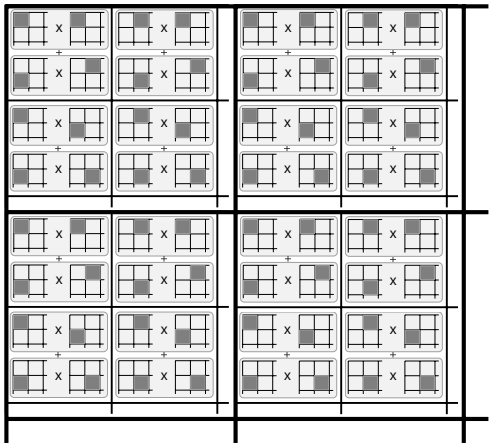

Рисунок 2 – Оптимизированная реализация алгоритма блочного произведения двух матриц.

В результате проведенных модификаций меняющиеся размеры основных параллельно перемножаемых блоков уже не оказывают ощутимого влияния на эффективность процессорного кэширования в задаче.

Дальнейшая идея состоит в том, что, меняя размеры малых блоков, можно контролировать порядок, в котором последовательный алгоритм вычисления произведения двух матриц обращается к разным ячейкам оперативной памяти. При изменении порядка обращений алгоритма к оперативной памяти меняется и эффективность использования процессорного кэша. Результаты экспериментов по изменению размеров малых блоков приведены на рисунке 3.

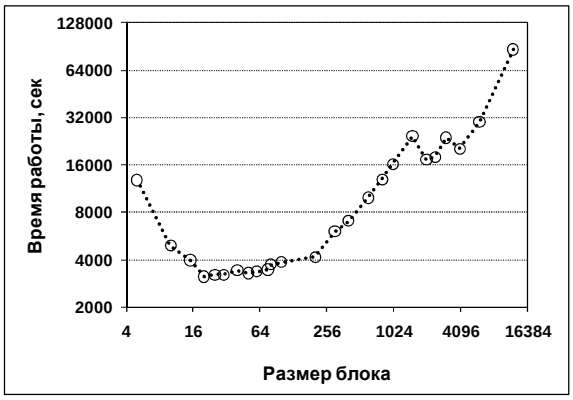

Рисунок 3 – График изменения времени работы алгоритма произведения двух матриц в зависимости от размера малых блоков.

Из результатов эксперимента видно, что при размерности малых блоков, соответствующей примерно 25*25 элементов, достигается максимальная эффективность использования процессорного кэша. Такая размерность малых блоков использовалась в дальнейших этапах эксперимента.

Показанные экспериментальные результаты были получены на процессорах Intel(R) Xeon(TM) CPU 3.20GHz @ 3192.205 MHz, для других моделей процессоров результаты оптимизации могут отличаться.

Последствиями использования оптимизации процессорного кэша в реализации задачи стали, следующие факторы:

- Ускорение работы последовательного алгоритма с двойным разбиением матриц составило примерно 10 единиц в сравнении с классическим блочным алгоритмом произведения матриц.
- Данная оптимизация позволила избежать проблем анализа дальнейших экспериментальных результатов. Проблемы были бы связаны с разной эффективностью процессорного кэширования для

последовательной и параллельной реализаций алгоритма.

Характеристики вычислительной системы и программное обеспечение

Экспериментальное исследование проводилось на гетерогенном кластере Киевского национального университета имени Тараса Шевченко [10] и на гомогенном кластере Института клеточной биологии и генетической инженерии. В составе кластера использовалось 8 четырехядерных вычислительных станций на основе процессоров Intel(R) Xeon(TM) CPU 3.20GHz @ 3192.205 MHz, с объемом оперативной памяти 4Гб на станцию.

Все станции были объединены сетью Gigabit Ethernet, экспериментальная пиковая производительность которой составила 80..90 Мб/сек. При тестировании в качестве транспортной библиотеки использовался OpenMPI [11] версии 1.2.8, пиковая производительность при передаче данных между станциями составляла 45 Мб/сек.

Потери производительности при передаче данных для OpenMPI в сравнении с первоначальной производительностью сети Gigabit Ethernet были связаны с затратами времени на следующие операции:

- Поддержка внутреннего протокола MPI, то есть дополнительный уровень инкапсуляции сетевого протокола.
- Поддержка конвертации типов данных для совместимости разных процессорных архитектур
- Потери времени на интерфейсных функциях библиотеки MPI

Тем не менее, MPI предоставляет единый интерфейс для передачи данных между потоками параллельной программы в независимости от типа используемого межпроцессорного транспорта и задействованных протоколов связи. По этой причине транспорт в DDCI-системе реализован на основе MPI, а потери эффективности связанные с этим являются оправданными [12].

Для компиляции использовался компилятор GCC со следующими настройками оптимизации: -m64 -O3 -mtune=opteron. Использование режима оптимизации при компиляции позволило ускорить работу последовательного алгоритма примерно в 5 раз в сравнении с программой, скомпилированной без оптимизации.

Используемая модель планировщика DDCI-системы

Для динамического распараллеливания на основе DDCI-системы использовался нераспределенный однопоточный планировщик, занимающий одну вычислительную станцию.

Планировщик в DDCI-системе выполняет задачи сервера для всех потоков и занимается анализом графа параллельной программы. Он берет на себя проблемы по балансировке нагрузки на вычислительных станциях и при этом учитывает меняющиеся во времени характеристики гетерогенного кластера [12].

В основе использованного в экспериментах планировщика лежат быстрые алгоритмы анализа графа параллельной программы со сложностью, не превышающей O(N*log(N)), где N – количество одновременно планируемых к выполнению чистых функций.

Вычислительные станции в используемой сборке DDCI-системы не имеют собственных планировщиков, а только выполняют команды централизованного планировщика.

Упрощенная блок-схема работы запущенной DDCI-системы приведена на рисунке 4. На рисунке «Планировщик» и «Расчетные станции» занимают по одному процессору/ядру. Количество расчетных станций может меняться. Все потоки ввода вывода в программе перенаправляются через планировщик на расчетные станции.

Рисунок 4 – Упрощенная блок-схема работающей DDCI-системы.

Экспериментальная реализация задачи

Размерность матриц для эксперимента составила в среднем 12000 * 12000 элементов. В расчетах использовались числа с плавающей точкой двойной точности, в результате чего одна матрица занимала примерно 1Гб оперативной памяти. Для расчета всей задачи в последовательном режиме требовалось примерно 3 Гб оперативной памяти на один процессор.

Время работы последовательного алгоритма, учитывая все проведенные оптимизации, составило примерно 3 часа 17 минут. Для параллельного алгоритма минимальное время работы составило примерно 7 минут 20 секунд, что соответствует ускорению

в 26.5 раз. Данное ускорение было получено для динамически распараллеливаемой программы ,запущенной на 31-ом процессоре, с разбиением двух матриц на основные блоки размерностью 1333*1333 элементов.

Во всех экспериментах для упрощения блочной реализации алгоритма произведения матриц размеры основных матриц менялись в диапазоне от 12000 до 11968, что соответствовало разбросу вычислительной сложности задачи в диапазоне +/- 0.4%. Данные погрешности при получении конечных результатов по эффективности распараллеливания алгоритма были учтены и скомпенсированы.

В экспериментах матрицы разбивались на блоки размерностями от 300*300 до 6000*6000. Такие разбиения соответствуют изменению общей размерности графа параллельной программы от 8 до 64000 вершин соответственно.

Экспериментальное исследование пиковой эффективности динамического распараллеливания вычислений

Под пиковой эффективностью динамического распараллеливания алгоритма для заданного числа процессоров подразумевается эксперимент с такой размерностью основного блока матрицы, которая позволяет добиться максимальной скорости работы параллельной программы в условиях динамического планирования потоков.

Ниже приведены графики, демонстрирующие изменение времени работы параллельной программы (рис. 5) и утилизации процессорного времени (рис. 6) в зависимости от числа потоков в параллельной программе.

Рисунок 5 – График изменения времени работы параллельной программы в зависимости от числа потоков.

На рис. 5 демонстрируется время работы параллельной программы при идеальном распараллеливании (пунктирная кривая), на практике такая эффективность недостижима. Показаны два графика, описывающие пиковую эффективность динамического распараллеливания задачи (точечная кривая) и эффективность распараллеливания алгоритма при статично заданной размерности блока, соответствующей 1333*1333 элементов (серая кривая). Из графика видно, что в среднем подход динамического распараллеливания вычислений позволяет добиться высокой эффективности работы параллельных программ, при этом полностью решается проблема масштабируемости параллельных программ под разное количество процессоров.

Рисунок 6 – График изменения утилизации вычислительной мощности в параллельной программе в зависимости от числа потоков.

График утилизации процессорного времени был приведен для более наглядной демонстрации существующих потерь процессорного времени при распараллеливании вычислений. Из графика видно, что пиковая эффективность при динамическом распараллеливании задачи всего на 5..6 % выше, в сравнении с алгоритмом, не оптимизированным под конкретное число процессоров.

В задаче наблюдается явное падение эффективности распараллеливания для 23 и более вычислительных станций. Падение связанно с использованием всего одного планировщика для управления одновременно всеми вычислительными станциями. Проблема состоит в том, что пропускная способность транспортного канала не позволяет в данном примере одному планировщику контролировать одновременно более 23 вычислительных станций, не внося задержек в работу задачи. По причине того, что данное экспериментальное исследование, в первую очередь, посвящено проблеме блочности алгоритмов, а не проблемам распределенного планирования потоков, то более сложные распределенные планировщики в эксперименте не участвовали. Кроме того распределенное планирование вносит свои

погрешности в эксперимент и поэтому не желательно.

В результате анализа эффективности распараллеливания можно утверждать, что используемая система для автоматического динамического распараллеливания вычислений в многопроцессорных компьютерных системах с распределенной памятью позволяет решать проблемы масштабируемости и переносимости алгоритмов. Кроме того, предложенное решение позволяет добиваться высокой скорости работы параллельных программ, используя автоматическое распараллеливание, и не задумываться о представлении алгоритма в распределенной среде.

Теоретические основы проблемы выбора размерности блоков при динамическом распараллеливании

Основная проблема выбора размерности блоков в последовательном алгоритме заключается в противоречивости двух основных характеристик любого блока:

- Объем необходимых данных для вычисления одного блока должен быть минимален
- Время работы алгоритма в блоке должно быть как можно больше

В результате, противоречивость состоит в том, что высказанные требования зачастую являются взаимосвязанными, ярким примером чего является график изменения этих величин для рассматриваемой в статье задачи (рис. 7).

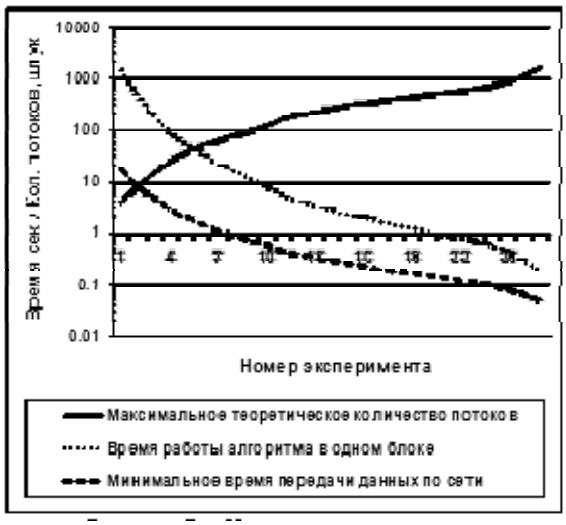

Рисунок 7 – Изменение временных характеристик одного блока в алгоритме произведения двух матриц в зависимости от максимального числа потоков для всей задачи.

Вторая проблема в том, что характеристика «максимальное теоретическое количество потоков» является жестким ограничителем максимальной эффективности распараллеливания алгоритма при росте количества станций в используемой многопроцессорной компьютерной системе.

Рассмотрим изменение коэффициента эффективности чистых функций. (рис. 8).

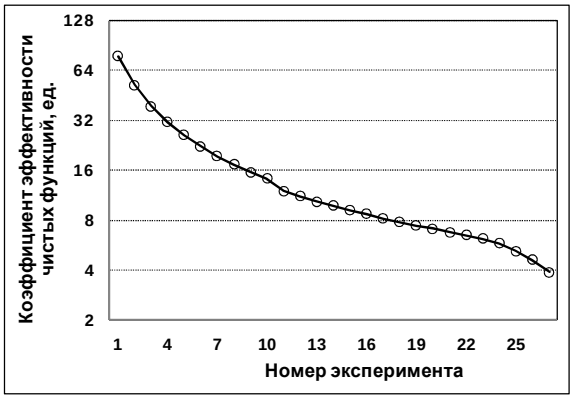

Рисунок 8 – Значение коэффициента эффективности чистых функций для всех проведенных экспериментов.

Из рисунка видно, что на плечи программиста ложится тяжелая задача ограничения максимального теоретического количества потоков для алгоритма. Так, если максимального теоретического количества потоков для алгоритма будет слишком мало, то алгоритм будет плохо масштабироваться при переносе его между разными мультипроцессорными системами. В то же время, в большинстве случаев повышение теоретического количества потоков ведет к быстрому падению эффективности распараллеливаемых функций, что может привести к низкой эффективности распараллеливания любой программы в целом, независимо от размерности используемой многопроцессорной системы.

Все эти неоднозначности и приводят к необходимости экспериментального анализа проблемы выбора размеров блоков в параллельных программах, подлежащих динамическому распараллеливанию.

Экспериментальный анализ влияния размерности блоков на эффективность динамического распараллеливания

Для начала рассмотрим все графики моделированного эксперимента (рис. 9).

На рисунке продемонстрировано 8 графиков, каждый из них соответствует определенному числу реально использованных параллельных потоков. Номера экспериментов из рисунка соответствуют номерам экспериментов из предыдущей теоретической части (рис. 7 и 8). Это означает, что первый эксперимент соответствует всего 4 максимально

возможным потокам, а последний эксперимент соответствует уже 1600 максимально возможным потокам.

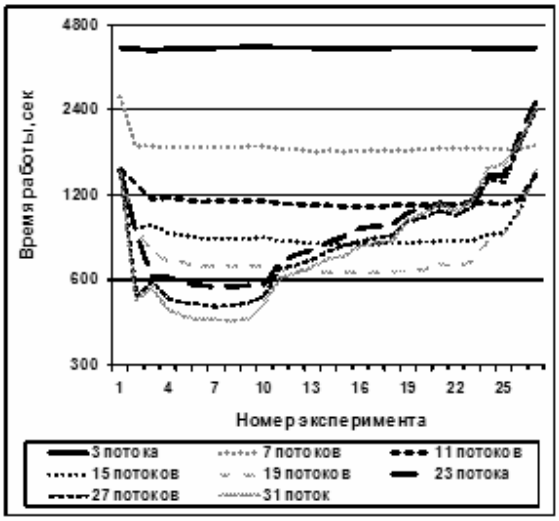

Рисунок 9 – Изменение времени работы параллельного алгоритма в зависимости от теоретического и реального количества потоков.

Рассмотрим первый график, соответствующий трем реально используемым потокам в программе. Он вырожден в дпрямую. Причина вырожденности связанна с тем, что на кластере были установлены 4 ядерные вычислительные станции, и значит, что все параллельные потоки данного эксперимента выполнялись на одной вычислительной станции. В результате гладкость данного графика показывает только то, что распараллеливание вычислений для систем с общей памятью проходит намного проще, в сравнении с системами, реализующими распределенную память.

Рисунок 10 – Изменение времени работы задачи в эксперименте с тремя реальными потоками.

На рисунке 10 показано изменение времени работы алгоритма для трех реальных потоков в разных экспериментах. Из графика видно, что значения времени работы алгоритма «случайным» образом меняется в 3% диапазоне. Стоит заметить, что при повторном проведении этого эксперимента данный график не изменит своей формы, а значит должна быть причина такого «случайного» разброса времени работы задачи в зависимости от максимального количества потоков в алгоритме. Причина такому поведению есть и она связана с «кратностью» общего числа распараллеливаемых блоков алгоритма и количества реальных параллельных потоков. Под «кратностью» подразумевается проблема, связанная с тем, что даже при оптимальном динамическом распараллеливании вычислений всегда как минимум один последний блок графа алгоритма считается в последовательном режиме, так как параллельно с этим блоком считать уже больше нечего. На практике таких блоков, как правило, несколько и это уже связанно с формой графа программы и динамически меняющимися характеристиками кластера.

Теперь рассмотрим следующих четыре графика, демонстрирующих схожие особенности при моделировании (рисунок 11). Первое, что бросается в глаза, это почти идеальное совпадение трех графиков в первом эксперименте (это графики для 11,15 и 19 реальных потоков) и такое же совпадение двух графиков во втором эксперименте (это графики для 15 и 19 реальных потоков). Такое поведение алгоритма является демонстрацией того, как максимальное теоретическое количество потоков ограничивает эффективность распараллеливания программ в больших многопроцессорных компьютерных системах.

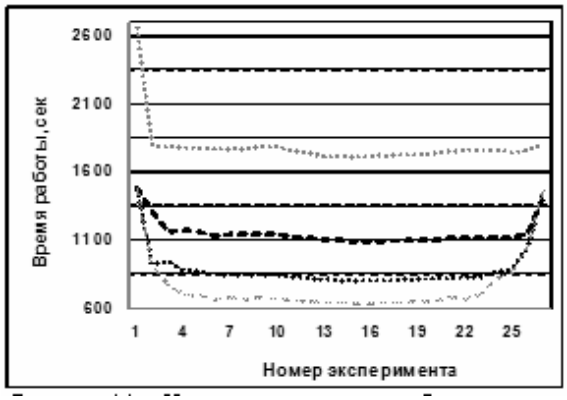

Рисунок 11 – Изменение времени работы задачи в экспериментах с 7, 11, 15, 19 потоками.

В центральной части графиков для экспериментов от 13 до 19 видно достижение максимальной эффективности распараллеливания задачи для всех четырех рассматриваемых примеров. Это, в свою очередь, означает, что при динамическом распараллеливании вычислений существует оптимальная «эффективность чистых функций»,

которая не зависит от теоретического максимального числа потоков, от реального числа потоков и от формы графа алгоритма. По результатам эксперимента данное оптимальное значение эффективности чистых функций соответствует примерно 8 единицам (эффективность чистых функций в 13-ом эксперименте была равна примерно 10 ед., а в 19-ом эксперименте 7 ед.).

Последняя часть рассматриваемых графиков демонстрирует проблемы, связанные с неэффективностью распараллеливания задачи (начиная с 22 эксперимента и до конца). Такое поведение связано с чрезмерным разбиением задачи на блоки, что привело к следующим последствиям:

- Блоки, на которые разбит алгоритм, стали слишком мелкими. Эффективность реализующих их чистых функций соответствует менее 5 единицам.
- Размерность графа стала слишком большой для данной задачи. Размерность графа для рассматриваемых экспериментов меняется от 20 до 70 тысяч вершин в графе.
- Максимальное теоретическое количество потоков велико, и меняется в диапазоне от 500 до 1500 потоков.

Описанные причины потери эффективности распараллеливания привели к тому, что система в основном занимается не расчетом задачи в параллельном режиме, а планированием распараллеливания данной задачи.

Рассмотрим последние три графика из эксперимента (рисунок 12).

Рисунок 12 – Изменение времени работы задачи в экспериментах с 23, 27, 31 потоками.

На этих графиках наблюдается возрастание желательной эффективности чистых функции до 13 единиц, а также резкое ухудшение параллельности в задачах для экспериментов от 10-го и выше. Причина таких изменений связанна с задержками в централизованном планировщике при обслуживании более 20 вычислительных станций одновременно. При использовании распределенного планировщика данная проблема исчезает, но остаются повышенные требования к эффективности чистых функций, около 10..11 ед.

Выводы

Автоматическое динамическое распараллеливание вычислений позволяет добиваться высокой эффективности работы параллельных программ на компьютерных системах, использующих распределенную память. При этом получаемая эффективность параллельных программ может составить серьезную конкуренция для программ, реализованных на основе стандартных транспортных библиотек, таких как MPI, использующих неавтоматическое статическое распараллеливание вычислений.

Параллельные программы, использующие динамическое распараллеливание, отличаются легкостью масштабирования для разного числа параллельных потоков. Более того, они автоматически подстраиваются под характеристики используемого транспортного решения между вычислительными станциями. Минимальные требования к пропускной способности транспортных каналов в динамически распараллеливаемых программах зависят только от «эффективности чистых функций» используемых в параллельной программе.

Динамическое распараллеливание программ можно использовать в системах с общей памятью. При этом динамическое распараллеливание вычислений может оказаться идеальным подходом для программирования многопроцессорных компьютерных систем класса NUMA[13].

Введенный коэффициент «эффективность чистой функции» достаточно удобно использовать при написании параллельных программ, а также во время статического анализа минимальных требований динамически распараллеливаемой программы к транспортной системе. Также использование данного коэффициента при написании параллельной программы позволяет контролировать размерность графа параллельной программы методом объединения слишком мелких блоков алгоритма в один.

Проблема максимального теоретического количества потоков должна решаться самим программистом на этапе написания параллельной программы, так как данная характеристика алгоритма является жестким ограничителем масштабируемости параллельной программы. При этом желательно, чтобы параллельная программа всегда имела 3-ех .. 5-ти

кратный запас максимального количества потоков относительно реально использованного.

Оптимизация динамически распараллеливаемых программ под конкретное число потоков является неэффективной, так как дает слабо ощутимый прирост в скорости работы задачи (около 5%), а так как динамически балансируемым программам свойственен разброс в скорости работы (около 3%) без явно контролируемых причин, такая оптимизация теряет всякий смысл.

Для эффективного динамического распараллеливания программ используемые в них «чистые функции» должны иметь эффективность не менее 7 единиц. Если данный коэффициент превышает 15 единиц, то для кластера на основе транспортной архитектуры «Gigabit Ethernet» это уже не приведет к ускорению работы параллельной программы. Но может быть сигналом, указывающим на слишком слабое разбиение последовательного алгоритма на блоки.

Использование распределенных планировщиков в динамически распараллеливаемых программах при количестве потоков не более 2-х .. 3-х десятков является необязательным и может быть скомпенсировано подбором чистых функций с повышенной эффективностью.

Литература

1. Frumkin M., Jin H., and Yan J. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. NAS Technical Report NAS-99-011, NASA Ames Research Center, Moffett Field, CA, 1999.
2. W. Gropp, E. Lusk,T. Sterling. Beowulf Cluster Computing with Linux, 2nd Edition, MIT Press, 2003, 504 p.
3. R.I. Levchenko, O.O. Sudakov, S.D. Pogorelij, Y.V. Bojko. A System of Automatic Dynamic Paralleling of Computations for Multiprocessor Computer Systems with Weak Connection (DDCI) / USiM. – 2008. – N 3. – P. 66-72.
4. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. // Journal of Network and Computer Applications, 23:187-200, 2001 (based on conference publication from Proceedings of NetStore Conference 1999).p 187-200.
5. R.I. Levchenko, O.O. Sudakov, S.D. Pogorilyy. DDCI: Interface for transparent parallelizing of calculations. // Proceedings of the ninth international young scientists' conference on applied physics, p. 112.
6. R.I. Levchenko, O.O. Sudakov, Yu.L. Maistrenko. Parallel software for modeling complex dynamics of large neuronal networks // Proc. 17th International Workshop on Nonlinear Dynamics of Electronic Systems, Rapperswil, Switzerland,. June 21-24, 2009, P. 34-37.
7. Yu.L.Maistrenko, O.O.Sudakov, and R.I.Levchenko. Parallel software for nonlinear dynamics and information processing in large neuronal networks of the brain. // Third Vogt-Brodmann Symposium. Jülich, Germany. 4th - 6th December 2009.
8. Keren, A. Barak. Opportunity Cost Algorithms for Reduction of I/O and Interprocess Communication Overhead in a Computing Cluster. IEEE Tran. Parallel and Distributed Systems 1 (14), (2003), pp. 39-50.
9. R.I. Levchenko, O.O. Sudakov, S.D. Pogorilyy, Yu. V. Boyko. Efficiency Problems of Automatic Dynamic Parallelizing of Computations for Multiprocessor Computer Systems with Weak Connections // The 7-th International Scientific and Practical Conference on Programming UkrPROG'2010. Ukraine, Kiev, May 25-27, 2010. pp. 208-214.
10. Yu.V. Boyko, O.O. Vystoropsky, T.V. Nychyporuk, O.O. Sudakov. Kyiv National Taras Shevchenko University High Performance Computing Cluster. In Proc. Third International Young Scientists Conference on Applied Physics. June, 18-20, 2003, Kyiv, Ukraine, P. 180-181.
11. Jeffrey M. Squyres, Bill Saphir, and Andrew Lumsdaine. The Design and Evolution of the MPI-2 C++ Interface. In Proceedings, 1997 // Internantional Conference on Scientific Computing in Object-Oriented Parallel Computing, Lecture Notes in Computer Science, 1997. Springer-Verlag. pp. 57-64.
12. R.I. Levchenko, O.O. Sudakov, S.D. Pogorelij. DDCI: Simple Dynamic Semiautomatic Parallelizing for Heterogeneous Multicomputer Systems // Proceedings of the 5-th IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. Rende (Cosenza), Italy. September 21-23, 2009. pp. 208-211.
13. R. Yang, J. Antony, A. P. Rendell. A Simple Performance Model for Multithreaded Applications Executing on Non-uniform Memory Access Computers // Proceedings of the 2009 11th IEEE International Conference on High Performance Computing and Communications. pp. 79-86.

Поступила в редакцию 23.02.2010