Анализ и моделирование производительности трафаретных вычислений на ccNUMA системах

ИВМиМГ СО РАН
kalgin@ssd.sscc.ru

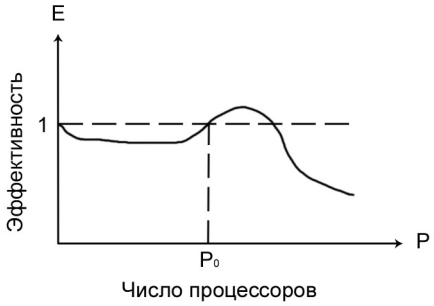
Аннотация

В работе проводится анализ и строится модель, описывающая производительность параллельной реализации трафаретных вычислений (явная разностная схема, синхронный клеточный автомат) для ccNUMA систем. Тестирование проводилось на системах с процессорами Intel Westmere и SandyBridge.

Построенная модель позволяет глубже понять динамику исполнения трафаретных вычислений на ccNUMA системах, отличия в архитектуре различных ccNUMA систем, а также автоматически находить оптимальное отображение данных и вычислений на архитектуру ccNUMA систем.

Введение

Реализация трафаретных вычислений (например, явная разностная схема, синхронный клеточный автомат) методом декомпозиции области на домены является традиционной и показательной задачей в любом начальном курсе изучения параллельного программирования. Типичный график эффективности в слабом смысле ($E_p = T_1/pT_p$, размер задачи не меняется с ростом числа процессоров p) выглядит следующим образом:



Выделим две зоны на этом графике: зона I - $p < p_0$, домен не помещается в кэш целиком; зона II - $p > p_0$, домен на каждом вычислителе помещается в кэш целиком. Получению высокоэффективных реализаций для зоны I за счёт эффективного использования кэша поясвящено множество работ (см., например, [1, 2]). В зоне II при начальном росте p наблюдается сверхлинейное ускорение ($E_p > 1$), поскольку данные помещаются в кэш, но затем эффективность достаточно быстро падает до 50% и ниже. Досканальное понимание причин быстрой деградации эффективности в зоне II представляет особый интерес, поскольку это может привести к созданию более эффективных реализаций, в том числе и других алгоритмов.

Данная работа посвящена анализу и построению модели производительности трафаретных вычислений для ccNUMA архитектур в случае, когда домен помещается в кэш (зона II). Тестирование проводилось на ccNUMA системах с процессорами Intel Westmere и SandyBridge. В качестве вычислительных схем использовались 5- и 9-точечные невзвешенные и взвешенные средние ($R_5, R_9, R_{5\alpha}, R_{9\alpha}$), также варьировался формат представления чисел с плавающей точкой - одинарный (float) и двойной (double).

Последовательная реализация

Оценка времени работы последовательной программы проводится следующим образом. На каждом ядре запускается один поток. Каждый поток вычисляет 10^5 итераций над своим доменом, синхронизация или обмен граничными значениями между потоками во время исполнения этих итераций отсутствуют, среднее время исполнения одной итерации использовалось в качестве оценки времени исполнения $T_{seq}(x, y)$ при заданных размерах домена (x, y). Загрузка всех ядер нужна для преодоления эффекта повышенных частот (Intel TutboBoost). Перед вычислением10^5 итераций каждый поток вычисляет половину секунды числа Фиббоначи для повышения частоты с пониженной до нормальной. На пониженные частоты ядро переходит во время простоя или слабой загрузки (Intel SpeedStep).

схема	A	B_1	B_2	C_1	C_2	C_3
R_5						
R_9						
$R_{5\alpha}$						
$T_{9\alpha}$						

Таблица 1: Westmere

За основу в качестве оценки времени работы была взята формула вида

$$T_{seq}(x, y) = A + By + Cxy, \tag{1}$$

где A - время вызова процедуры, реализующей итерацию, B - время, затрачиваемое на переход с одной строки домена на другую (двумерная область), C - время обработки одного узла домена.

Параметр A включает в себя не только вызов процедуры, но и загрузку первых кэш-строк в кэш L1 в начале работы функции (если домен не помещается в кэш L1). Можно записать $A = A_1 + A_2(xy \in L_2) + A_3(xy \in L_3)$, но при размерах домена $xy > L_1$ время вычисления итерации T_{seq} оказывается достаточно большим и слагаемое $A_2(xy \in L_2) + A_3(xy \in L_3)$ вносит вклад меньше 0.1%. Далее считаем $A_2 = A_3 = 0$.

Параметр B зависит от качества работы предсказателя. Например, в архитектуре Intel Westmere имеется предсказатель числа итераций цикла, работает с циклами в которых не более 64 итераций [todo:рис.]. Для процессоров Intel Westmere $B = B_1 + B_2(x > 64)$.

Параметр C зависит от того, в какой уровень кэша помещается домен

$$C = C_1 + C_2(size \in L_2) + C_3(size \in L_3).$$

При наивном подходе для оценки значений параметров $A, B_1, B_2, C_1, C_2, C_3$ обычно используется метод линейной регрессии на некоторой выборке из множества значений $T_{seq}(x, y)$, где $(x, y) \in L1 \cup L2 \cup L3$. Этот подход плох тем, что зачастую выдаёт “нефизичные” значения параметров, например отрицетельные, в также появляются “выбросы” – небольшое число экспериментов с большой погрешностью в оценке T_{seq} (ошибка достигает до 20-100%).

В данной работе значения параметров $A, B_1, B_2, C_1, C_2, C_3$ оценивались методом линейной регрессии следующим образом. В первую очередь опеределялись значения A, B_1, C_1 на выборке из $\{(x, y) \in L1,\ y \le x \le 64\}$, затем – значение B_2 на $\{(x, y) \in L1,\ y \le x \le 128\}$, C_2 на $\{(x, y) \in L2,\ y \le x \le 1024\}$, C_3 на $\{(x, y) \in L3,\ y \le x \le 1024\}$. Такой подход позволяет получать стабильные неотрицательные значения A, B_1, B_2 для разных схем, и все экспериментально полученные значения T_{seq} описываются (1) с не более чем 3% ошибкой.

схема	A	B	C_1	C_2	C_3
T_5					
T_9					
$T_{5\alpha}$					
$T_{9\alpha}$					

Таблица 2: Sandy Bridge

Барьерная синхронизация

Оценка времени, затрачиваемого на барьерную синхронизацию T_{bar} проводится следующим образом. На каждом ядре одного или всех доступных сокетов запускалась последовательная реализация на своём домене без обмена граничными значениями между потоками(так же, как при оценке T_{seq}), но потоки на каждой итерации синхронизировались. Пусть T' - среднее время исполнения одной итерации с синхронизацией, тогда $T_{bar} = T' - T_{seq}$.

Синхронизация между итерациями проводилась одним из трёх способов:

T_{b0} барьерная синхронизация средствами OpenMP (#pragma)

T_{b1} реализация барьера на volatile переменных []

T_{b2} локальная синхронизация, т.е. потоки синхронизируются при прохождении контрольной точки не со всеми потоками, а только с теми, с которыми они должны были бы иметь границы в случае полной реализации граничных обменов.

В качестве первой грубой оценки можно сказать следующее:

$$T_{b2} = T_{b1}/4 = T_{b0}/8$$

$$T_{b0}(1socket, xy \in L1) = 510ns$$

$$T_{b1}(1socket, xy \in L1) = 160ns$$

$$T_{b2}(1socket, xy \in L1) = 80ns$$

$$T_{bx}(2socket) = 2T_{bx}(1socket)$$

$$T_{bx}(xy \in L3) = 2T_{bx}(xy \in L1)$$

[Более точные оценки и графики будут добавлены позже].

Деление области на домены и обработка границ

Счетную область D можно делить на домены $D_{i,j}$ вертикальными (V) или горизонтальными (H) линиями. Размер каждого домена равен (x, y). Домены располагаются в отдельных локальных массивах (P,private) и границы

при этом копируются, или могут располагаться в едином массиве с общими границами (S,shared), тогда границы копировать не надо. Время T_{bound}, затрачиваемое на поддержку актуальных значений на границах зависит от размера домена (x, y), числа используемых сокетов (один или несколько), способа деления области (V и/или H) и расположения доменов (P и/или S).

Пусть $T^{\{1,2\}}_{\{V,H\},\{P,S\}}(x, y)$ - время копирования вертикальной (V) или горизонтальной (H) границы между локальными (P) или общей (S) памятью внутри одного (1) или между двумя сокетами (2). Оценим время копирования T_{bound} через максимальное время копирования всех границ при заданном делении (V/H) и размещении (S/P) доменов.

В работе [3] показывается, что латентность обращения и пропускная способность копирования данных в SMP и ccNUMA системах существенно зависит от уровня кэша, в котором располагаются актуальные значения. Далее используются следующие обозначения для времён копирования данных из кэшей других ядер/сокетов:

Z_x латентность обращения к данным, располагаемым в кэше Lx другого ядра

W_x время чтения одного значения из кэша Lx, при последовательном чтении (пропускная способность $W_x = sizeof(realtype)/bandwidth_x$)

Характерные значения Z_x и W_x для Intel Westmere:

[Добавить таблицу Z1~60ns для тогоже сокета, 180ns для другого сокета, BWlocal=40Gb/s BWsocket=20Gb/s BW2socket=10Gb/s].

Вертикальные границы

Локальные массивы для доменов (private). Используется дополнительный массив для временного хранения граничных значений. Значения с вертикальной границы вначале записываются в этот массив (чтение по столбцам из своего кэша, запись последовательная в свой кэш) потоком-"хозяином" домена, затем, после синхронизации, другой поток записывает значения из временного массива в свою границу (чтение последовательное из чужого кэша, запись по строкам в свой кэш).

$$T_{V,P}(x, y) = Z_1 + W_1(y - 1) + Qy,$$

где Q – время копирования значений из и в вертикальную границу домена.

Общий массив для счетной области (shared). В начале и в конце каждой строки происходит работа с граничными значениями. При чтении соответствующая кэш-строка в худшем варианте копируется из чужого кэша, в лучшем - находится в локальном кэше. Предполагая худший вариант, время оценивается как

$$T_{V,S}(x, y) = Z_k y,\ xy \in Lk.$$

Согласно [3], $Z_1 \geq Z_2 \geq Z_3$ и $W_1 \ll Z_k$, поэтому $T_{V,S} > T_{V,P}$ при $y > 1$. При $y = 1$ время $T_{V,P}$ превышает $T_{V,S}$ не более, чем на 1-3%. Далее будем считать, что для вертикальных границ оптимальной организацией поддержки актуальных значений является использование локальных массивов для доменов и дополнительных массивов для копирования граничных значений между итерациями (private).

Горизонтальные границы

Локальные массивы для доменов (private). Используется дополнительный массив для временного хранения граничных значений. Все чтения и записи последовательны.

$$T_{H,P}(x, y) = Z_1 + W_1(y - 1) + Q'y,$$

где Q' - время копирования значений из и в горизонтальную границу домена, $Q' \ll Q$.

Общий массив для счетной области (shared). В отличие от вертикальных границ, здесь мы имеем последовательное чтение из чужого кэша:

$$T_{H,S}(x, y) = Z_k + W_k(y - 1),\ xy \in Lk.$$

Согласно [3], $Z_1 \geq Z_2 \geq Z_3$ и $W_1 \geq W_2 \geq W_3$, поэтому $T_{H,S} < T_{H,P}$. Таким образом, для горизонтальных границ оптимальной организацией поддержки актуальных значений является общий массив (shared).

Заключение

Данная работа посвящена анализу и построению модели производительности трафаретных вычислений для ccNUMA архитектур в случае, когда домен помещается в кэш. Тестирование проводилось на ccNUMA системах с процессорами Intel Westmere и SandyBridge.

Построенная модель позволяет описывать время исполнения последовательной реализации с точностью 2-3%. Анализ времени исполнения показал, что традиционо используемый барьер встроенный в OpenMP достаточно медленный и даёт существенный вклад на малых доменах ($xy \in L1$), полученная реализация работет в пять раз быстрее. Построена модель времени межядерного взаимодействия на границах доменов, выявлены оптимальные способы деления счетной области (shared/private).

Работа поддержана Грантом Президента РФ для молодых учёных МК-3644.2014.9

Список литературы

[1] Lakshminarayanan Renganarayana, Manjukumar Harthikote-Matha, Rinku Dewri, and Sanjay Rajopadhye. Towards Optimal Multi-level Tiling for Stencil Computations.

[2] Shah M. Faizur Rahman, Qing Yi, Apan Qasem. Understanding Stencil Code Performance On MultiCore Architectures

[3] Daniel Molka, Daniel Hackenberg, Robert Schone and Matthias S. Muller. Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System// DOI 10.1109/PACT.2009.22.