УДК 004.021, 004.252, 004.75

Д. А. ФАДЕЕВ

ОСОБЕННОСТИ ЧИСЛЕННОГО РЕШЕНИЯ ЭВОЛЮЦИОННЫХ ЗАДАЧ РАСПРОСТРАНЕНИЯ КОРОТКИХ ЛАЗЕРНЫХ ИМПУЛЬСОВ НА СИСТЕМАХ АРХИТЕКТУРЫ NUMA

Рассмотрены приемы адаптации алгоритмов решения эволюционных задач распространения коротких лазерных импульсов к особенностям вычислительной архитектуры NUMA.

Ключевые слова: *NUMA, прогонка, волновое уравнение, сверхкороткий импульс.*

Архитектура NUMA (Non Uniform Memory Access) определяет класс параллельных вычислительных систем с логически общей, но физически раздельной памятью [1]. Существует достаточно много вариантов реализации такой архитектуры. В частности, компания Advanced Micro Devices ввела подход, при котором чипы RAM подключаются непосредственно к чипу CPU, содержащему встроенный контроллер памяти. В случае использования MultiCPU-платформ каждый CPU подключается к своему набору планок RAM. При этом память остается логически общей благодаря наличию шины HyperTransport, имеющей существенно больший пропускной канал, чем связка CPU-RAM. Более того, серверные процессоры AMD имеют N–1 HyperTransport каналов (N — число процессоров (чипов) в одном вычислительном узле (материнской плате)) и каждый чип CPU соединен с каждым. Таким образом, достигается реализация логически общей памяти. При этом скорость доступа данного CPU к памяти любого нода (связки CPU-RAM) одинакова, если требуемая память не занята другим запросом. Таким образом, в NUMA ширина канала увеличивается во столько же раз, сколько нодов имеется на вычислительном узле. В настоящее время архитектура NUMA поддерживается и на персональных компьютерах, в частности, использующих платформу Nehalem [2].

В настоящей статье рассматриваются особенности реализации задачи эволюционного (по координате z) численного моделирования распространения лазерного импульса $A(z,\tau,r)$ с широким спектром:

$$\frac{2}{c}\frac{\partial^2 A}{\partial\tau\partial z}+\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial A}{\partial r}\right)=0. \qquad (1)$$

Такое соотношение может быть получено из уравнений Максвелла для условий малоуглового распространения короткого импульса вдоль оси z, где $\tau = t - z/c$ – сопутствующее время. В фурье-пространстве уравнение (1) перейдет в следующее:

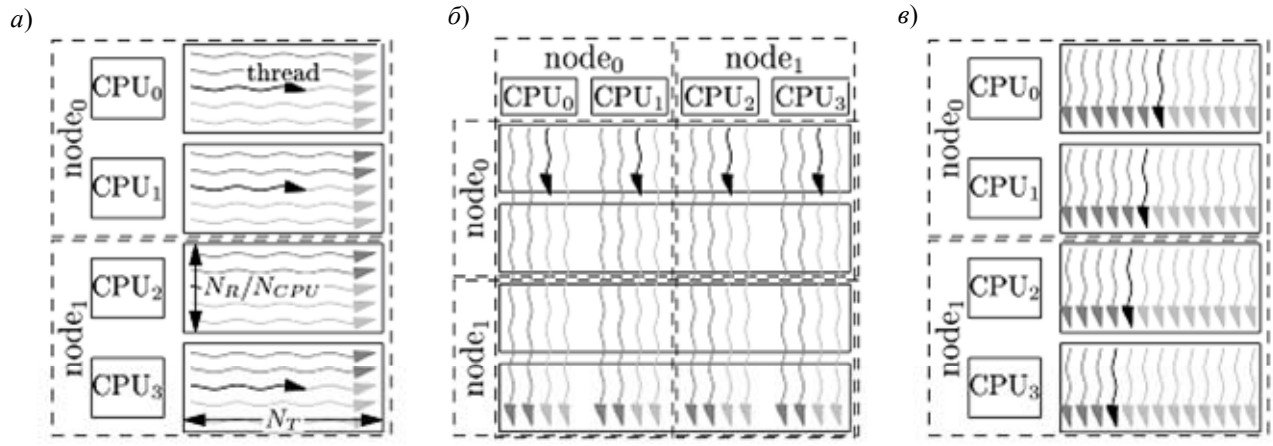
$$-\frac{2}{c}i\omega\frac{\partial B}{\partial z}+\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial B}{\partial r}\right)=0, \qquad (2)$$

которое, в свою очередь, аппроксимируется посредством N_T независимых конечно-разностных уравнений:

$$-i\omega_n\frac{\partial B}{\partial z}+\hat{L}B=0, \quad n\in[0,N_T-1], \qquad (3)$$

где оператор $\hat{L}$ аппроксимирует дифференциальный оператор в соотношении (2). Данный оператор может быть записан в трехдиагональном виде как обобщение обычного двумерного пятиточечного лапласиана. Уравнение (3) решается численно методом Крэнка—Никольсона для каждой гармоники ω_n.

С точки зрения эффективного использования архитектуры NUMA элементы массива $\mathbf{A}(r_i,\tau_j)$ целесообразно расположить так, чтобы соседние элементы по второй (временной) координате оказались соседними в памяти. Это удобно для выполнения преобразования Фурье, которое при таком размещении данных может выполняться построчно. Пусть далее в системе есть N_{NUMA} нодов, тогда все массивы $\mathbf{A}$, $\mathbf{B}$ размером $N_R \times N_T$ разбиваются на N_{NUMA} отдельных частей размером $(N_R/N_{\text{NUMA}})\times N_T$. На рисунке, *а* представлено параллельное выполнение преобразования Фурье; *б* — первая часть прогонки (прямая прогонка) с конфликтом памяти; *в* — прямая прогонка без конфликта памяти. Для выполнения преобразования Фурье по строкам достаточно запустить N_{NUMA} независимых нитей — thread (*а*). В соответствии с предложенным расположением массивов в памяти соседние по r элементы оказываются удалены на N_T элементов в памяти. Более того, элементы из разных подмассивов оказываются на разных нодах, поэтому простейший цикл по столбцам матрицы $\mathbf{B}$, реализуемый при выполнении прогонки, оказывается неоптимальным, так как при этом физические процессоры будут синхронно обращаться к памяти одного нода, как это представлено на рисунке, *б*.



Выходом в данном случае может быть „лестничная“ процедура, представленная на рисунке, *в*. Процесс прямой прогонки (обнуление одной из боковых лент матрицы обращаемого оператора) можно начинать выполнять на одном ядре (CPU_0), после того как элементы

$\mathbf{B}(r_0, \tau_0), ..., \mathbf{B}(r_{N_R/N_{\text{CPU}}}, \tau_0)$ обработаны, можно переходить к обработке элементов $\mathbf{B}(r_0, \tau_1), ..., \mathbf{B}(r_{N_R/N_{\text{CPU}}}, \tau_1)$, а CPU_1 может начать обработку элементов $\mathbf{B}(r_{N_R/N_{\text{CPU}}}, \tau_0), ..., \mathbf{B}(r_{2N_R/N_{\text{CPU}}-1}, \tau_0)$. Таким образом, все члены CPU будут задействованы с некоторой задержкой. Однако большую часть данных ядра будут обрабатывать одновременно, не обращаясь к данным чужого нода. Если окажется что блок памяти, привязанный к данному ядру, помещается в собственный кэш ядра то конфликт между CPU_0, CPU_1 и CPU_2, CPU_3 будет также разрешен. Вторая часть (обратная прогонка) выполняется так же, только процесс начинается с последнего вычислительного ядра.

Для практической реализации предложенной выше процедуры необходима некоторая модификация. Так как обращение к любому элементу памяти, отсутствующему в кэше, вызывает считывание целой кэш-линии первого уровня (L1 cache line), следует выполнять модифицированную прогонку, реализующую один шаг с номером i сразу для 16 элементов ($\mathbf{A}(r_i, \tau_j), ..., \mathbf{A}(r_i, \tau_{j+16})$). При этом время ожидания в начале предложенного „лестничного“ процесса несколько возрастает, однако если выполнять прогонку для каждого столбца отдельно, неоптимальное обращение к памяти приведет к еще большей потере производительности.

Дополнительной особенностью моделирования распространения лазерного импульса является необходимость оптимизации распределения нагрузки, поскольку в области слабых полей вычисления можно не производить. Но в таком случае возникает простой отдельных вычислительных узлов на каждом эволюционном шаге. Одним из способов создания равномерной нагрузки может быть варьирование размеров подмассивов, рассчитываемых на отдельных CPU. При этом размер массива N_T по координате τ остается прежним, изменяется только количество строк N_{Rk} в подмассиве (исходно $N_{Rk} = N_R / N_{CPU}, \forall k$). Так как при каждом изменении размера подмассива выделяются и высвобождаются большие объемы памяти, данную операцию следует производить через некоторое, достаточно большое, число эволюционных шагов.

Рассмотренные подходы являются достаточно перспективными благодаря разработке NUMA-систем. На данный момент на рынке широко представлены системы с максимальным числом нодов 2—4. В условиях увеличения числа ядер на один NUMA-нод (до 12 в последних предложениях от AMD) эти системы остаются, на наш взгляд, по-прежнему несбалансированными по соотношению вычислительная производительность—пропускная способность памяти. Эксперименты с вычислительной станцией на базе Intel Nehalem 2 CPU X5550 показали, что увеличение каналов памяти (3 канала в Nehalem) не оправдывает заявленной производительности и при этом не допускает контроля со стороны программиста. В этом смысле использование особенностей NUMA более эффективно, чем применение в рамках модели глобальной общей памяти, разделяемой между ядрами.

СПИСОК ЛИТЕРАТУРЫ

1. A NUMA API for LINUX* 2005 [Электронный ресурс]: <developer.amd.com/assets/LibNUMA-WP-fv1.pdf>.

2. [Электронный ресурс]: <http://download.intel.com/pressroom/pdf/nehalem-ex.pdf>.

Сведения об авторе

Даниил Александрович Фадеев — Институт прикладной физики РАН, отдел нелинейной электродинамики, Нижний Новгород; младший научный сотрудник;
E-mail: fadey@appl.sci-nnov.ru

Рекомендована НИИ НКТ

Поступила в редакцию
15.05.11 г.