ВЫЧИСЛИТЕЛЬНАЯ ПРОИЗВОДИТЕЛЬНОСТЬ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ПРОГОНКИ НА КЛАСТЕРНОМ СУПЕРКОМПЬЮТЕРЕ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ И ГРАФИЧЕСКИХ УСКОРИТЕЛЯХ GPU

В.Э. Витковский

Введение.

Численные методы решения дифференциальных уравнений часто приводят к системам линейных алгебраических уравнений с ленточной (в простейшем случае -- трехдиагональной) матрицей. Подобные системы возникают при реализации решений параболических или эллиптических уравнений в одномерном случае либо в многомерном при покоординатном расщеплении решения.

Для решения систем с подобной матрицей разработан специальный алгоритм, называемый методом прогонки, который является ни чем иным, как методом Гаусса (или алгоритмом прогонки), учитывающим структуру матрицы [1].

Большинство существующих в настоящее время методов распараллеливания алгоритма прогонки, например, recursive doubling, cyclic reduction, partition metod и их модификации [2-8] эффективны для векторных компьютеров и не позволяют получить высокой производительности вычислений для систем с архитектурой DMP (distributed-memory multiprocessors) и SMP (symmetric-memory multiprocessors), вследствие высокой интенсивности межпроцессорных обменов для этих алгоритмов. В данной работе изучается производительность параллельного алгоритма прогонки, предложенного в работе [9], на высокопроизводительных вычислительных системах с архитектурой (DMP) и на графических ускорителях (GPU) на примере решения нелинейного уравнения Шредингера (НУШ) с кубической нелинейностью в цилиндрических координатах, моделирующего самофокусировку лазерного импульса в прозрачных диэлектриках, с начальным условием типа чирпованного гауссового (G) (m=0) и кольцевого распределения (R1) (m=1).

$$i\frac{\partial\Psi}{\partial z}+\Delta_r\Psi+|\Psi|^2\Psi=0,\quad \Delta_r=\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\Psi}{\partial r}\right),$$

$$\Psi(0,r)=A_0 r^{\mathrm{m}}\exp\left(-\frac{(1+C^2)r^2}{2a_0^2}\right),\quad m=0,1. \qquad (1)$$

Здесь Psi(z,r) -- медленная огибающая электрического поля, a_0 -- поперечный радиус пучка, C -- параметр чирпа.

Численные расчеты производились по разностной схеме Кранка-Николсон sigma=0.5, образующей трехдиагональную матрицу, с использованием сетки с переменным шагом по r и z.

$$i\cdot\frac{\hat{\Psi}_{\mathrm{n}}-\Psi_{\mathrm{n}}}{\Delta z}+\sigma\cdot L\hat{\Psi}_{\mathrm{n}}+(1-\sigma)\cdot L\Psi_{\mathrm{n}}+|\Psi_{\mathrm{n}}|^2\cdot(\sigma\cdot\hat{\Psi}_{\mathrm{n}}+(1-\sigma)\cdot\Psi_{\mathrm{n}})=0,$$

$$L\Psi_{\mathrm{n}}=\frac{1}{r_{\mathrm{n}}}\left[\frac{\left(r_{\mathrm{n}}+\frac{h_{\mathrm{n+1}}}{2}\right)\left(\frac{\Psi_{\mathrm{n+1}}-\Psi_{\mathrm{n}}}{h_{\mathrm{n+1}}}\right)-\left(r_{\mathrm{n}}-\frac{h_{\mathrm{n}}}{2}\right)\left(\frac{\Psi_{\mathrm{n}}-\Psi_{\mathrm{n-1}}}{h_{\mathrm{n}}}\right)}{\frac{h_{\mathrm{n}}}{2}+\frac{h_{\mathrm{n+1}}}{2}}\right], \qquad (2)$$

$$h_{\mathrm{n+1}}=h_{\mathrm{n}}+\frac{h_{\mathrm{n}}}{M},\quad M=25000,\quad h_1=10^{-6},\Delta z=\frac{\Delta z_0\cdot|\Psi(0,0)|^2}{|\Psi(z,0)|^2}.$$

Здесь |Psi(z,0)|^2 - интенсивность в центре пучка, Delta z_0 — начальный шаг по эволюционной переменной z. Такой способ построения сетки (очевидно не единственный) позволяет учесть для данной задачи возможность образования особенности в центре пучка в фокусе.

Алгоритм параллельной прогонки.

В работе используется параллельный алгоритм предложенный в [9] и исследованый в работах [10-14]. Алгоритм заключается в следующем: область определения r, в данном случае одномерный вектор, состоящий из N элементов, разбивается на SIZE произвольных интервалов размером M_{P}, тогда вектор решение уравнения

$$a_i x_{i-1} - b_i x_i + c_i x_{i+1} = d_i, \quad x_i \in X, \quad x_0 = \alpha_0 x_1 + \beta_0, \quad x_N = \alpha_N x_{N+1} + \beta_N, \qquad (3)$$

имеет вид

$$X = (x_0^1 \dots x_{M_1-1}^1; \dots; x_0^P \dots x_{M_P-1}^P; \dots; x_0^{SIZE} \dots x_{M_{SIZE}-1}^{SIZE}) = (X^1, \dots, X^P, \dots, X^{SIZE}). \quad (4)$$

Здесь SIZE — число процессоров, P - индекс процессора. В данной работе предполагается, что интервалы вектора X равны.

Представим вектор интервал в виде

$$X^P = R^P + P^P \cdot W_{P-1} + Q^P \cdot W_P, \qquad (5)$$

W_P=x_0^{P+1} - значение на границе интервала.

Чтобы получить решение X нужно найти все P^P, Q^P, R^P, W_P, поэтому для каждого P методом прогонки решаются следующие уравнения:

$$a_i P_{j-1}^P - b_i P_j^P + c_i P_{j+1}^P = 0, \quad P_0^P = 1, \quad P_M^P = 0,$$

$$a_i Q_{j-1}^P - b_i Q_j^P + c_i Q_{j+1}^P = 0, \quad Q_0^P = 0, \quad Q_M^P = 1, \qquad (6)$$

$$a_i R_{j-1}^P - b_i R_j^P + c_i R_{j+1}^P = d_i, \quad R_0^P = 0, \quad R_M^P = 0.$$

Каждому P соответствует свой индекс

$$j \in [0, M_P - 1] \text{ т.е. } i = (P-1) \cdot M_P + j.$$

Затем, подставляя (5) в (3), получаем уравнения на обменные граничные условия.

$$A_k W_{k-1} - B_k W_K + C_k W_{k+1} = D_k, \quad W_0 = \xi_0 W_1 + \eta_0, \quad W_{SIZE-1} = \xi'_{SIZE-1} W_{SIZE} + \eta'_{SIZE-1},$$

$$A_k = a_i P_{P-1}^{M-1},$$

$$B_k = b_i - a_i Q_{P-1}^{M-1} - c_i P_P^1, \qquad (7)$$

$$C_k = c_i Q_P^1,$$

$$D_k = d_i - a_i R_{P-1}^{M-1} - c_i R_P^1.$$

Граничные условия слева

$$\xi_0 = \frac{\alpha_0 Q_1^1}{1 - \alpha_0 P_1^1} \text{ и } \eta_0 = \frac{\alpha_0 R_1^1 + \beta_0}{1 - \alpha_0 P_1^1}.$$

Граничные условия справа, в случае если значение W_{SIZE} равно какой-либо величине, в нашем случае W_{SIZE}=Psi_{LSE}, где Psi_{LSE} - решение линейного уравнения Шредингера на правой границе. Если граничное условие справа дано в виде соотношения

$$W_{SIZE-1} = \xi'_{SIZE-1} W_{SIZE} + \eta'_{SIZE-1},$$

то значение W_{SIZE} получаем, решая систему уравнений

$$\left.\begin{aligned} W_{SIZE-1} &= \xi'_{SIZE-1} W_{SIZE} + \eta'_{SIZE-1} \\ W_{SIZE-1} &= \xi_{SIZE-1} W_{SIZE} + \eta_{SIZE-1} \end{aligned}\right\}$$

где xi_{SIZE-1} и eta_{SIZE-1} получены из прямого хода прогонки для начальных значений xi_0 и eta_0.

Выполняемое число операций в этом алгоритме, при расчете трех матриц в прямом и обратном ходе, в два раза меньше трехкратного выполнения расчетов для одной матрицы за счет повторяемости элементов

вычисления. Однако, число операций для переопределения элементов матрицы может быть значительно больше числа операций для прогонки.

Оценочное выражение для ускорения S параллельного алгоритма (закон Амдала) имеет следующий вид [15,16]

$$S = \frac{N_{\mathrm{M}}}{\frac{aN_{\mathrm{M}}}{N_{\mathrm{CPU}}} + b(N_{\mathrm{CPU}}) + c}. \qquad (8)$$

Здесь a - коэффициент пропорциональный числу операций на один элемент сетки размером N_{M}, N_{CPU} - число используемых процессоров, b(N_{CPU}) - функция характеризующая задержки сети, зависящая от числа процессоров, c - коэффициент, учитывающий неускоряемую часть программы. Этот же результат справедлив и для расчетов на кластере из GPU. В этом случае N_{GPU} - число используемых GPU. Вычисления на GPU характеризуются большим числом используемых потоков, вплоть до десятков тысяч, в то время как на кластере из обычных процессоров сотни потоков, поэтому коэффициент c (последовательная часть алгоритма) для кластера из GPU составляет значительную величину по сравнению с параллельной частью алгоритма, если не используются большие сетки и параллельная часть достаточна велика. Величина b(N_{GPU}) характеризует скорость взаимодействия хоста с графическими ускорителями. Данный алгоритм позволяет последовательно взаимодействовать каждому ядру хоста с соответствующим GPU, что позволяет эффективно расходовать пропускную способность PCI Express, не деля её с соседним процессом копирования. Дело в том, что пока одно ядро копирует, другое общитывает последовательную часть алгоритма, определенную соответствующим GPU. Этот процесс особенно эффективен в случае b<c.

Результаты и обсуждения.

Алгоритм параллельной прогонки был реализован на языке Fortran 77 с использованием технологии MPI для реализации на кластере и на языке C с использованием CUDA для GPU. Анализ производительности выполнялся на кластерном суперкомпьютере на основе процессора Intel Itanium 2, 1.6 GHz, cache 3 Mb и сетевого коммутатора InfiniBand (10 Гбит/сек, Cluster Interconnect, t_L=10 mu s) с использованием компилятора Intel Fortran 10.1 с оптимизационной опцией -fast и на графических ускорителях с процессором T10 (Tesla C1060 и Tesla S1070) и GeForce 9600GT с использованием CUDA 2.1 и OpenMP для хоста в случае подключения нескольких GPU. Ускорение параллельного алгоритма вычислялось как отношение среднего времени выполнения одного шага прогонки последовательного алгоритма к среднему времени одного шага параллельной прогонки для различных по величине сеток N_M (76000, 140000, 198000, 255000, 998000 ячеек). Для всех сеток расчеты времени выполнения одного шага выполнены до условий, когда основное расчетное время занимают обменные операции пересылок. В обменные операции при расчетах на кластере входят две коллективные операции обмена по 1024 и 256 бит на каждый процессор (обменные граничные условия), неблокирующие отложенные операции обмена по 256 бит на каждый процессор (синхронизация шага по эволюционной переменной z), неблокирующие операции обмена на 3-х смежных процессорах по 512 бит на каждый процессор (элементы на краях интервала для схемы Кранка-Николсон).

На рисунке (Рис.1.a.) представленны кривые ускорения S(N_{CPU}) для различных N_{M} на процессорах Itanium. Здесь обнаруживается соответствие с законом Амдала (8). Максимальное ускорение ограничивается функцией описывающей задержки сети b(N_{CPU}), т.е. пропускной способностью и латентностью.

На рисунке (Рис.1.a.) наклон кривой ускорения для сетки 998000 в 2.5 раза меньше наклона S(N_{CPU}) для остальных сеток. Здесь дело в том, что при увеличении числа процессоров N_{CPU} уменьшается размер интервала M_P на каждом процессоре и тем большая доля вычислений происходит в кэше первого уровня, вследствие чего, постепенно увеличивается наклон S для сетки с N_M=998000. Скачки ускорения могут быть связанны с перепадами в пропускной способности сети, вследствие того, что пропускная способность является основным фактором, ограничивающим ускорение при большом числе процессоров. При анализе полученных данных наблюдается уменьшение эффективности E и ускорения S параллельного алгоритма по мере роста размера расчетной сетки (за исключением случаев при малых временах t одного шага). Задержки сети сказываются при малых N_M, т.е. На грубых сетках и при большом числе процессоров N_{CPU} при больших N_M, когда интервал расчетной сетки, вычисляемый на одном процессоре достаточно мал и большая часть вычислений происходит в кэше первого уровня так, что время вычислений сравнимо с временем сетевых обменов. Максимальное наблюдаемое ускорение, полученное для обсуждаемого алгоритма равно 30.4 для N_M=998000 и N_{CPU}=102. Наиболее оптимальные условия для ускорения достигаются при вычислениях на расчетной сетки, состоящей из 198000 ячеек на 30 процессорах, при этих условиях достигается ускорение S=20.6 с вычислительной эффективностью равной 0.7. Алгоритм был применен в численных исследованиях свойств локального коллапса решений нелинейного уравнения Шредингера. В процессе расчетов были определены, например, критические мощности локального коллапса для гауссового и кольцевого распределений P_{Cr,G}=1.900 и P_{Cr,R1}=2.132 [18]. Из-за возникновения численной особенности при

коллапсе, для увеличения точности расчетов необходимо значительное измельчение сетки по эволюционной переименной z и поперечному радиусу r. Рабочая сетка по переменной r состояла из 255000 ячеек. Расчеты выполнялись на процессорах Itanium, с использованием 36 процессоров. Ускорение алгоритма при этом состовляло S=22.3. Число используемых процессоров выбиралось исходя из условий наибольшего наклона кривой S(N_{CPU}) для данной расчетной сетки. При моделировании процесса лазерной записи объемных наноструктур мощными фемтосекундными импульсами с использованием обсуждаемого алгоритма были получены линейные ускорения в десять и более раз [19].

При реализации параллельного алгоритма на графических ускорителях, один шаг по z включал три вычислительных ядра CUDA, копирование на хост и копирование на GPU. Создание трех ядер CUDA, необходимость, возникнувшая вследствие отсутствия быстрой функции синхронизации блоков потоков. Использование возможной функции, позволяющей быстро последовательно обработать данные со всех потоков, исключило бы взаимодействие графического ускорителя и хоста. В то же время, данные хранящиеся в регистровых переменных каждого потока, доступны с большими скоростями, чем данные из кэша обычного процессора. Здесь мы видим скрытую (еще не доступную) возможность эффективно реализовать расчет последовательной части алгоритма.

Результаты измерения ускорения расчетов на GPU обнаружили, что характерные ускорения на одном GPU соизмеримы с результатами на кластере (Таблица. 1.). Ускорение для одинарной точности в два раза больше чем для двойной точности (регистров в два раза больше), несмотря на то, что вычислительная мощность в десять раз выше. Также наблюдается прирост в ускорении для GPU нового поколения. Ускорения, полученные на кластере из GPU

S		76000	140000	198000	255000	998000
Tesla C1060	DP	9.3	10.9	19.8	19.8	17.4
Tesla C1060	SP	14.9	18.8	36.2	30.7	34.5
GeForce 9600GT	SP	13.8	15.1	29.5	25.6	24.2
cluster NCS-160	DP	12.4(36)	14.6(48)	28.4(60)	28(60)	30.4(108)
t(s)	DP	0.0051	0.0085	0.021	0.0241	0.07

Таблица. 1. Ускорения параллельного алгоритма на GPU для двойной DP и одинарной SP точности полученные на разных сетках, максимальное ускорение параллельного алгоритма прогонки на кластере S_{t{MAX}} и соответствующее число процессоров, время выполнения одного шага последовательного алгоритма t.}

На рисунке (Рис. 1. б.) представлены кривые ускорения для сетки N_M=980000, полученные на кластере и графических ускорителях. При этом, ускорения для двух и более GPU оценены из модельной программы (код для хоста написан на языке C с использованием технологии OpenMP), протестированной на двух GPU (доступные ресурсы) на многоядерном процессоре (хост). В модели учтено, что время взаимодействие хоста с GPU скрывается последовательной частью алгоритма. Пока одно ядро копирует данные с GPU, другое расчитывает обменные граничные условия. Последовательная часть для кластерной версии пренебрежимо мала параллельной части, в реализации для GPU используются тысячи потоков, поэтому расчет с обменными граничными условиями в последовательной части занимает большую часть общего времени расчета до 100%. Таким образом, при оптимальном числе потоков, когда общее время расчета достигает своего минимума, по мере увеличения используемых GPU, уменьшается время расчета параллельной части алгоритма (объем данных делится на несколько GPU), но увеличивается последовательная часть пропорционально числу GPU. Вследствие необходимости уменьшения последовательной части, при уменьшении числа потоков GPU падает эффективность вычислений (Рис. 1.в.). Здесь приведен график времени вычисления одного шага алгоритма на один элемент сетки t_{E} в зависимости от числа используемых потоков для различных расчетных сеток. Замедление ускорения наблюдается для кривой включающей последовательную и параллельную часть для GPU и кластера по мере наращивания вычислительных мощностей (Закон Амдала). Каждому GPU соответствует 80 GFlops для DP и 1 Tflops для SP, каждому процессору в кластере - 6.25 GFlops. На графике также показана кривая ускорения на GPU без последовательной части (так, как это для кластерной версии) и без взаимодействия хоста с GPU. Последнее из вышеперечисленного является узким местом для многих приложений, особенно, этот фактор заметен при низких скоростях канала связывающего хост и GPU: устаревшая версия PCI Express, использование канала одновременно для нескольких GPU, проблемы на отрезке до PCI Express (планировщик для многоядерной системы). Как видно из графика для кода без последовательной части и без обменов с хостом достигаются значительно лучшие результаты. Подобная зависимость проявляет себя в большей мере для одинарной точности (Рис.1.г.) и грубых сетках.

Ускорения, полученные на кластере из GPU, превышают ускорения на кластере из обычных процессоров для двойной точности и в большей мере для одинарной точности. Причем параллельная часть на

GPU ускорилась приблизительно в три раза быстрее. Так, для N_M=998000 S^{MAX}_{DP}(4 GPU)=44.3(72.8), S^{MAX}_{SP}(4 GPU)=65.4(155.5), в скобках указано значение для параллельной части алгоритма.

Операции с памятью внутри GPU гораздо эффективнее межпроцессорных обменов внутри кластера, но маштабируемость хоста из многоядерных процессоров с несколькими GPU значительно ограничивается возможностями канала PCIExpress.

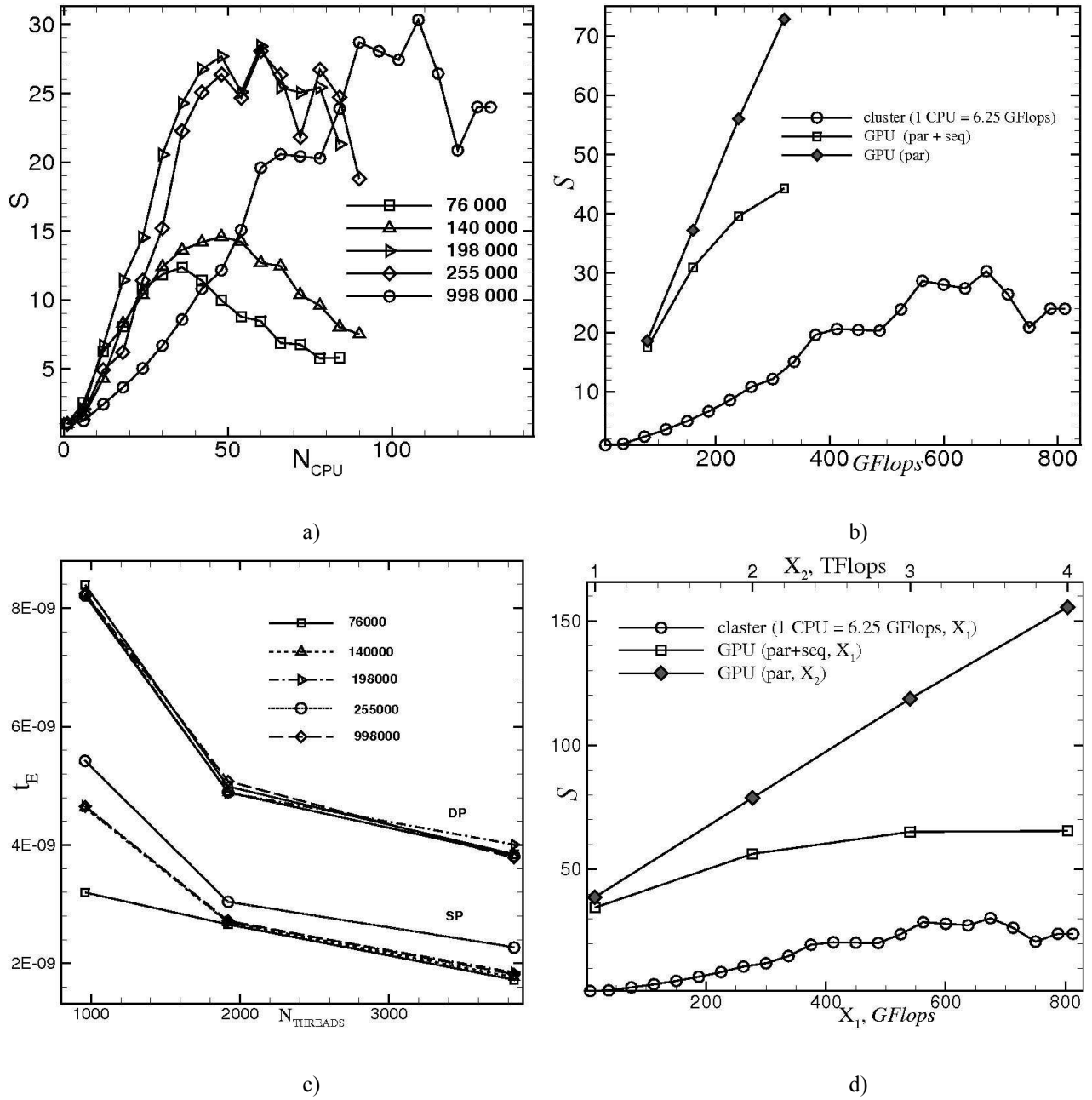

Рис.1. а - ускорение S параллельной прогонки относительно последовательного алгоритма для разных сеток на процессорах Itanium, б - кривые ускорения S для кластера и GPU в зависимости от используемой мощности для сетки N_M = 998000 в DP, в - удельное время выполнения одного шага алгоритма на GPU на один элемент сетки, тоже что и б, но в SP.

Выводы.

Анализ производительности параллельного алгоритма прогонки на кластерном суперкомпьютере показал, что основным фактором, ограничивающим ускорение, являются не только задержки сети, но и эффекты кэширования данных в процессоре, как и сама производительность вычислительного модуля. Задержки сети сказываются при малых N_{M}, т.е. на грубых сетках и при большом числе процессоров N_{CPU} при больших N_{M}, когда интервал расчетной сетки, вычисляемый на одном процессоре достаточно мал и большая часть вычислений происходит в кэше первого уровня так, что время вычислений сравнимо с временем сетевых обменов. При больших N_{M} ожидалось получить лучшие результаты, вследствие

меньшего влияния сетевых задержек и зависимости S^{MAX}(N_{M}) из формулы (5) в работе [20], но из-за влияния размера кэша, оказалось, что с ростом размера расчетной сетки уменьшается эффективность и ускорение параллельного алгоритма (различие, до в 2.5 раз). Максимальное ускорения, полученное для обсуждаемого алгоритма, следующие: в 30.4 раз для N_{M}=998000 и N_{CPU}=102. Наиболее оптимальные условия для ускорения достигаются при вычислениях на расчетной сетки, состоящей из 198000 ячеек на 30 процессорах, при этих условиях достигается ускорение S=20.6 с вычислительной эффективностью равной 0.7. Ускорения достигаемые на одном графическом ускорителе для двойной точности соизмеримы с максимальными ускорениями достигаемыми на кластере, и превышают их на кластере из GPU. Ускорения для одинарной точности в два раза больше чем для двойной. Высокая эффективность вычислений на GPU достигается при большом числе потоков, вследствие чего, последовательная часть алгоритма занимает значительное время расчета одного шага. Кроме того, версия для кластера из GPU имеет значительные ограничения по маштабируемости из-за ограниченных возможностей PCIExpress.

Благодарности.
Выражаем благодарность В.И. Паасонену и Д.Л. Чубарову за многочисленные полезные обсуждения.

ЛИТЕРАТУРА:

1. L. H. Thomas. Elliptic problems in linear difference equations over a network. Technical report, Columbia University, New York, 1949.
2. R. W. Hockney , C. R. Jesshope. Parallel Computers: Architecture, Programming and Algorithms, IOP Publishing Ltd., Bristol, UK, 1981
3. C. L. Cox. Implemention of a divide and conquer cyclic reduction algorithm on the FPS T-20 hypercube, Proceedings of the third conference on Hypercube concurrent computers and applications, p.1532-1538, January 1989, Pasadena, California, United States
4. J. M. Ortega and R. G. Voigt. Solution of partial differential equations on vector and parallel computers,SIAM Rev., pp. 149-240, June 1985.
5. X.-H. Sun, H. Zhang, L.M. Ni. Efficient Tridiagonal Solvers on Multicomputers, IEEE Transactions on Computers, vol. 41, no. 3, pp. 286-296, Mar., 1992 SIAM Rev., pp. 149-240, June 1985.
6. X.-H. Sun, S. Moitra. A Fast Parallel Tridiagonal Algorithm for a Class of CFD Applications, NASA Technical Paper 3585, August 1996.
7. M. M. Chawla, K. Passi, R. A. Zalik. A recursive partitioning algorithm for inverting tridiagonal matrices, Internat. J. Computer Math., 35 (1990), 153-158.
8. Povitsky A. Parallel Directionally Split Solver Based on Reformulation of Pipelined Thomas. Algorithm,Technical Report 45, ICASENASA, 1998.
9. Yanenko N.N., Konovalov A.N., Bugrov A.N., Shustov G.V.} About organization of parallel calculations and parallelizing of Gauss algorithm// Numerical Methods for Continuum Mechanics 9, No. 7, 139-146(1978).
10. Paasonen V. I. Paasonen V.I. Boundary conditions of high-order accuracy at the poles of curvilinear coordinate systems // Russian Journal of Numerical Analysis and Mathematical Modelling. 1999. Vol. 14. N 4. P. 369-382. (In Russian).
11. Paasonen V.I. The parallel algorithm for the compact schemes in inhomogeneous areas // Computational technolgies. 2003. V. 8. 3. P. 98-106 (In Russian).
12. Paasonen V.I. The studies of the parallel algorithm\\ for the compact schemes in inhomogeneous areas // Computational technolgies. 2005.V. 10. 5. P. 81-89 (In Russian).
13. Voevodin, A.F. and Shugrin, S.M. The numerical methods of calculating of one-dimensional systems.- Novosibirsk, Nauka, 1981. (In Russian).
14. T.A. Kudryashova, S.V. Polyakov About some methods of high performance solution of boundary value problem// IV International conference on mathematical simulations. 2. STANKIN, 2001.
15. Gene Amdahl Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS Conference Proceedings, (30), pp. 483-485, 1967.
16. Rychkov A. D. Parallel computational technologies, Siberian State University of Telecommunications and Informatics, 2007. (In Russian).
17. Chubarov D., Prokopyeva L.} Performance Evaluation of a parallel tridiagonal solver on a cluster of quad-core processors, Institute of Computational Technologies, Short Technical Report, 2007. (In Russian).
18. Vitkovskiy V. E., Fedoruk M. P. Numerical investigations the properties of solutions to Schrodinger equation during the propagation of laser pulses in an optical fibers. // Computational technolgies. 2008. V. 6. 13. P. 40-49 (In Russian).
19. Vitkovskiy V. E., Fedoruk M. P. Mathematical simulation of the femtosecond laser inscription of optical waveguides. // Laser Physics. 2008. V. 18. No. 11. P. 1268-1278.
20. Витковский В. Э., Федорук М. П. Вычислительная производительность параллельного алгоритма прогонки на кластерных суперкомпьютерах с распределенной памятью // Вычисл. методы и программирование. 2008. Т. 9, № 1. С. 305---310. (http://num-meth.srcc.msu.ru/).