

## ВОЗМОЖНОСТИ СОВРЕМЕННЫХ АППАРАТНЫХ АРХИТЕКТУР ДЛЯ УСКОРЕНИЯ РАСЧЕТОВ

*М. М. Лаврентьев, А. А. Романенко*

Новосибирский государственный университет

Мы будем говорить о катастрофических волнах-наводнениях цунами. Цунами – волна на поверхности океана, распространяющаяся за счет гравитации, вызванная подводным землетрясением или иным геологическим процессом на дне океана (извержением вулкана, оползнем). Большинство цунами (до 90 % – в районе Тихого океана) порождается землетрясениями, гипоцентры которых находятся в так называемых зонах субдукции, в определенной степени «опоясывающих» Тихий океан.

Само слово цунами имеет японское происхождение и означает «портовая волна» или «большая волна в гавани». Высота волны цунами в открытом океане, как правило, не превышает 1 метра и, следовательно, не представляет опасности. В то же время, в зависимости от величины области деформации морского дна, длина волны цунами измеряется десятками и более километров. При подходе к береговой линии волна становится короче при одновременном росте ее высоты. В береговой зоне зафиксированы волны высотой 35 метров и более. Глубина проникновения на сушу может достигать километров.

На протяжении всей истории человечества люди, живущие на берегу океана, время от времени страдали от атак цунами. Противостоять этому грозному стихийному явлению человечество пока не в состоянии, хотя в Японии, более всех государств подверженной атакам цунами, воздвигались разного рода защитные сооружения в виде мощных бетонных стен, береговых дамб. Эти сооружения могут несколько уменьшить ущерб от цунами небольшой высоты, но практически бесполезны в случае катастрофических цунами.

Учитывая то, что чаще всего волны цунами возникают вследствие сильных подводных землетрясений, служба предупреждения может объявлять тревогу сразу после регистрации сейсмометрами любого достаточно сильного подводного землетрясения. Однако при таком подходе будет объявлено слишком много ложных тревог, стоимость эвакуации может на порядки превышать материальные потери от атаки самой волны цунами. Так, после Чилийского землетрясения 27 февраля 2010 (пятого по силе за всю историю инструментальных наблюдений), был сделан прогноз высоты волны на побережье Японии. На части побережья ожидалась высота волны цунами более 3 м, время прихода соответствовало наиболее высокой фазе прилива. Были проведены масштабные эвакуационные мероприятия, прервано железнодорожное сообщение. Фактически, ни в одной точке побережья не было зафиксировано волны, превышающей 120 см, то есть никаких специальных действий не требовалось.

В случае объявления нескольких ложных тревог население перестанет на них реагировать, что приведет к жертвам в результате действительно опасной волны цунами. Более того, высокие волны цунами могут возникнуть в результате схода подводного оползня, спровоцированного относительно слабым землетрясением. Поэтому игнорировать слабые землетрясения службы предупреждения цунами также не должны.

Современные вычислительные технологии позволяют с достаточной точностью рассчитать распространение волны над глубоководной акваторией, если известны параметры смещения морского дна в очаге цунами. Отметим два наиболее распространенных программных пакета MOST и TUNAMI для расчета стадий генерации волны, ее распространения над глубоководной акваторией и наката на берег. TUNAMI-N2 [1] представляет собой программу численного моделирования волны цунами с линейной моделью распространения на глубокой воде и использующей приближение теории мелкой воды при подходе к береговой линии. Размер шага сетки одинаков во всей расчетной области. TUNAMI был первоначально разработан в 1993 Ф. Имамурой для пакета Tsunami Inundation Modeling Exchange (TIME) и успешно применялся и применяется в расчетах происходящих со-

битий. MOST (Method of Splitting Tsunami) [2, 3], разработанный в Тихоокеанской лаборатории окружающей среды (Pacific Marine Environmental Laboratory), г. Сиэтл, США, позволяет делать прогноз зон затопления в реальном режиме времени с использованием данных глубоководных датчиков. Этот пакет используется в США для создания карт возможных зон затопления [4].

Мы будем говорить об ускорении расчетной части пакета MOST, отвечающей за моделирование распространения волны над основной (глубоководной) акваторией океана. Вопросы генерации волны и наката на берег не рассматриваются.

### Модельные уравнения, схема расчета

Для численного расчета распространения волны цунами используется нелинейная система дифференциальных уравнений мелкой воды в следующем виде:

$$H_t + (uH)_x + (vH)_y = 0,$$

$$u_t + uu_x + vu_y + gH_x = gD_x,$$

$$v_t + uv_x + vv_y + gH_y = gD_y,$$

где  $H(x, y, t) = \eta(x, y, t) + D(x, y, t)$ ,  $\eta$  – высота волны, вычисляемая от невозмущенного уровня,  $D$  – функция, описывающая рельеф дна,  $u(x, y, t)$ ,  $v(x, y, t)$  – скорости вдоль  $x$  и  $y$  соответственно,  $g$  – ускорение свободного падения. Приведенная модель мелкой воды хорошо описывает процесс распространения волн цунами в открытом океане при условии, что горизонтальные размеры подвижки океанического дна, генерирующие эту волну, значительно (на порядок) превосходят глубину океана в этом месте. Эта система решается методом расщепления с использованием явной разностной схемы.

При использовании расчетной сетки с расстоянием между узлами 4 географические минуты (это соответствует физическому расстоянию между узлами сетки 3,6 км в средних широтах) акватория Тихого океана требует массива  $2581 \times 2879$  точек. Для моделирования распространения волны от побережья Чили до Японии (более 20 часов физического времени прохождения волны) требуется примерно 8600 итераций по времени. Один шаг по времени требует приблизительно  $110 \times 2581 \times 2879$ , т. е.  $10^9$  элементарных арифметических операций. Одна итерация по времени в пакете MOST требует около 3 сек при использовании 2,8 GHz процессора. Таким образом, при последовательном исполнении программы моделирование этапа распространения волны занимает около 7 часов. Для эффективного решения текущей задачи нужны современные вычислительные системы и оптимизирующие компиляторы.

### Схемы параллелизации, результаты на разных платформах

В соответствии с математической моделью, расчет движения волны происходит в два этапа: на первом шаге производится вычисление смещения волны вдоль оси  $X$ , на втором – вдоль оси  $Y$ . Используется подход расщепления направлений. При этом расчет  $i$ -ой и  $j$ -ой строки вдоль оси  $X$  можно производить независимо. Аналогичная ситуация и для расчета вдоль оси  $Y$ . Такая схема расчетов даст повод к переносу алгоритма на параллельные архитектуры.

Нами был выполнен перенос алгоритма на системы с общей (SMP) и распределенной памятью (кластера). Для кластеров наилучшее ускорение составило 6 раз на 12 вычислительных узлов. Для SMP систем – 16 раз на 8 вычислительных ядрах. Однако более существенное ускорение удалось получить при использовании процессоров IBM Cell BE и графических процессоров.

### Cell Broadband Engine (Cell/BE)

К современным вычислительным архитектурам можно отнести системы на базе процессоров IBM CELL BE. Эти процессоры, в частности, составляют вычислительное ядро в игровых приставках SONY PlayStation3, и на их базе был построен один из самых мощных кластеров в мире BlueGeneL. Эти процессоры состоят из 9 ядер, одно из них – общего назначения, которое отвечает за ввод/вывод данных и раздачу управления, а остальные 8 ядер являются векторными процессорами (рис. 1). Все ядра соединены высокоскоростной шиной передачи данных.

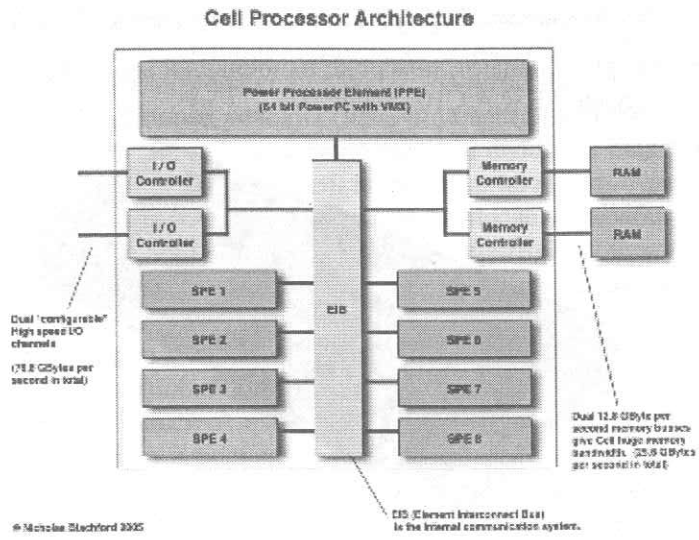


Рис. 1. Архитектура Cell/BE

Архитектура процессора позволяет писать приложения, которые эффективно реализуются как на конвейерной топологии, так и на топологии с общей или распределенной памятью. Заявляемая пиковая мощность одного процессора IBM CELL BE – порядка 110 Гигафлопс.

Программа для процессора IBM CELL BE состоит из двух подпрограмм: одна для управляющего ядра (PPE) и одна для каждого из восьми векторных ядер (SPE). Такая необходимость вызвана тем, что ядра в процессоре имеют разную архитектуру. В конечном итоге эти два модуля все равно собираются в один исполняемый модуль.

Программа для управляющего модуля занимается загрузкой данных и распределением работ по вычислительным ядрам процессора, которые собственно и производят вычисления.

Разбиение данных определяется наличием малого объема локальной памяти внутри векторных ядер. Для небольших сеток это может быть одна строка расчетной области, для больших – половина или треть строки. В случае моделирования распространения волны по Тихому океану одна строка расчетной области легко размещается в памяти SPE.

После адаптации программы под заданную архитектуру и проведения оптимизаций получено 50-ти кратное ускорение по сравнению с исходной последовательной программой (рис. 2).

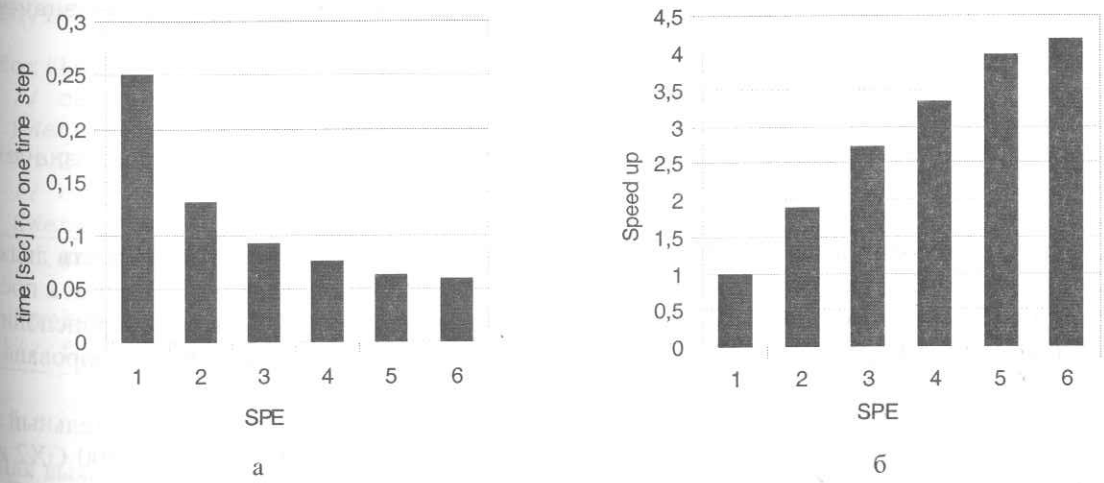


Рис. 2. а – время расчета одной итерации по времени в зависимости от количества SPE (Размер области моделирования 2048 × 2048); б – полученное ускорение в зависимости от количества SPE

### Использование графических процессоров TESLA

Графические процессоры обладают, пожалуй, наибольшей вычислительной мощностью на одно ядро – 1 Терафлопс в случае TESLA C1060 (рис. 3).

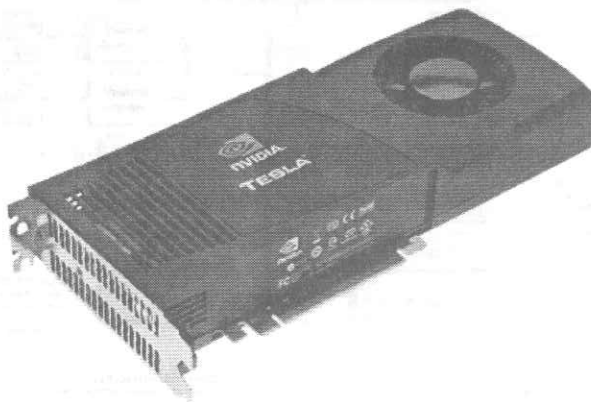


Рис. 3. Общий вид платформы TESLA C1060

В противовес разработчикам процессоров общего назначения разработчики графических процессоров пошли по пути упрощения вычислительных ядер и блока управления (этим от части занимается компилятор), а сэкономленное пространство кристалла использовали на вычислительные ядра. Результатом изменения подхода и архитектуры стало то, что далеко не все задачи, которые решаются на центральном процессоре, могут быть эффективно решены на графическом процессоре. Но те задачи, которые все-таки хорошо «ложатся» на целевую архитектуру, решаются в десятки сотни раз быстрее.

Задача моделирования распространения цунами как раз хорошо соответствует архитектуре GPU. Особенность архитектуры позволяет исключить необходимость транспонирования матриц, что требует существенного времени. Однако при этом требуется предпринимать действия, связанные с тем, что вычисления проводятся в одинарной точности.

Основной цикл схематично можно представить следующим образом:

```
[для каждого временного шага]
// расчет вдоль оси X
[для каждого столбца поля решения - i]
[вычислить инварианты для столбца i]
[произвести расчет - функция swater]
[записать результаты расчета - перевести инварианты в исходные значения]
// расчет вдоль оси Y
[для каждого ряда поля решения - i]
[вычислить инварианты для для ряда i]
[произвести расчет - функция swater]
[записать результаты расчета - перевести инварианты в исходные значения]

[записать решение на соответствующем временном шаге]
```

Для расчета используются четыре матрицы: глубина ( $d$ ), высота волны ( $q$ ), скорость движения волны ( $u$ ) вдоль оси  $X$  и скорость движения волны ( $v$ ) вдоль оси  $Y$ . В целях оптимизации в последовательной программе вторая половина итерации по времени обрамлена функциями транспонирования матриц  $q$ ,  $u$  и  $v$ . Матрица  $d$  (глубина) остается неизменной и может быть транспонирована один раз перед основным циклом.

Один из стандартных подходов к адаптации программ под GPU – это последовательный перенос участков кода на GPU. После переноса кода на видеокарте NVIDIA GeForce 9800 GX2 время выполнения одной итерации составило в среднем 0,25 секунды. Оптимизация не выполнялась. При этом около 43 % времени отнимало ядро SWater\_ (отвечающее за вычисление параметров волны цунами на следующем шаге по времени), ядро transpose() занимало еще 26 %.

Функции вычисления инвариантов и обратных инвариантов являются тривиальными и не нуждаются в комментариях. Реализация функции `transpose()` была взята из CUDA SDK [5]. Наиболее сложной в плане реализации оказалась функция `SWater_`. Внутри нее требовалось проводить большое количество проверок на граничные условия (наличие берегов материков и островов), и вести интенсивное чтение данных из памяти. Для упрощения этой функции был заранее выполнен подсчет некоторых условий, что позволило исключить вложенные проверки. Кроме того, все необходимые для расчета данные копировались в разделяемую память, после чего работа велась именно с ней.

При моделировании выяснилось, что вычисление инвариантов происходит с некоторой неточностью, что определяется наличием в ядрах вызова функции вычисления квадратного корня, которая дает ошибку в 1–2 младших битах, что в конечном итоге приводило к самопроизвольному раскочиванию поверхности океана уже на 50 итерациях. Было рассмотрено несколько решений: перейти на вычисления инвариантов в `double`, использовать представление `double` с помощью двух `float` [6], и, как вариант, вообще избавиться от вычисления квадратного корня внутри основного цикла. Первые два варианта не приводили к большому провалу по производительности. На текущий момент времени каждый потоковый мультипроцессор содержит по 8 модулей для работы с числами с плавающей точкой с одинарной точностью и по 2 для работы с числами с двойной точностью. Последний вариант – отказ от вычисления квадратного корня – оказался наиболее простым и подходящим в плане реализации.

Для того, чтобы избавиться от постоянного вычисления квадратного корня было решено производить все вычисления в инвариантах и только на стадии сохранения данных моделирования производить пересчет инвариантов назад к значениям высоты волны и ее скоростям по направлениям. В результате код был модифицирован, поверхность океана перестала «раскочиваться», время вычисления одной итерации уменьшилось и составило в среднем 0,23 сек на одну итерацию по времени. Еще на 10 % удалось ускорить программу, заменив в ядре `SWater_` некоторые операции деления на умножение. В соответствии с документацией деление занимает в 9 раз больше тактов, чем умножение: 36 против 4 тактов. Тем самым время вычисления одной итерации составило 0,21 секунду.

Были проверены несколько путей оптимизации программы:

1. Сделать две функции `SWater_`, которые будут производить вычисления вдоль разных направлений, и тем самым избавиться от вызова функции `transpose()`;
2. Выровнять начала строк всех матриц в памяти GPU для того, чтобы получить последовательно чтение. Заменить вызовы функций `cudaMalloc()` на `cudaMallocPitch()`;
3. Перейти к работе с памятью через текстуры.

В результате время выполнения одного цикла по времени сократилось до 0,135 секунд. При этом упростилась и сама программа.

```

Inv1<<<dimGrid, dimBlock>>>(... , x_size, y_size);

for(int i=0; i<cfg.steps; i++){
    // calculate along X
    SWater_<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    Inv2<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    // calculate along Y
    SWater_r<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    // getting data from "sensors" and save result
    ...
    Inv3<<<dimGrid, dimBlock>>>(..., x_size, y_size);
} // for cfg.steps

```

Рис. 4. Основной цикл расчета после оптимизации

Для выявления направлений дальнейшей оптимизации использовался профилировщик (запуск программы с установленными переменными окружения `CUDA_PROFILE` и `CUDA_PROFILE_CONFIG` или через утилиту `cudaProf`). Анализ показал, что, несмотря на простой вид функций `Inv2`

и Inv3, непоследовательное чтение и запись происходят по 2,6 миллиона раз, в то время как последовательное – всего несколько тысяч.

Величина в 2,6 миллиона непоследовательных операций чтения – это приблизительно количество элементов в используемых массивах. Таким образом, все элементы являются невыравненными на границу блока. Чтобы это исправить, заменим выделение памяти с помощью функции `cudaMalloc()` на `cudaMallocPitch()`. При этом модификация программы будет не существенная.

В результате такой замены количество непоследовательных обращений к памяти сократилось до нуля и время обработки одной итерации по времени составило 0,037 секунды, что в 100 раз лучше, чем для последовательной программы и в 14 раз быстрее, чем на 8 ядрах версии программ на OpenMP. Стоит сказать, что для последовательной версии программы выравнивание начала строк давало ускорение лишь на доли процента.

Для программ на CPU работа с памятью происходит эффективно, если данные в памяти расположены рядом друг с другом и легко помещаются в кэш или CPU может определить шаблон доступа к памяти, тем опять же заранее подгружая данные в кэш. У GPU для помещения данных в кэш можно использовать текстуры. При этом в кэш текстуры помещаются те данные, которые локализованы в двухмерном пространстве относительно данных, к которым идет обращение.

Дальнейшая оптимизация свелась к объявлению семи двумерных текстур и двух одномерных, привязке текстур к областям данных в памяти GPU и небольшой модификации всех ядер. Выигрыш от такой оптимизации составил в среднем 0,03 секунды на итерацию – около 10 %.

Для финального тестирования программа была выполнена на NVIDIA Tesla C1060. Полученный результат – 0,02 секунды на итерацию, что составляет итоговое ускорение около 170 раз по сравнению с исходной последовательной программой. Несмотря на то, что у Tesla C1060 в два раза больше ядер, чем у GeForce 9800 GX2, ускорение в 2 раза получено не было по причине того, что тактовая частота видеокарты на 20 % выше.

### Заключение

В работе описаны результаты по адаптации программы расчета распространения волны цунами на современные вычислительные архитектуры. Было показано, что при использовании даже обыкновенного компьютера с графической картой можно получить ускорение больше, чем на кластерах и многопроцессорных системах с общей памятью, однако при этом нужна совместная работа специалиста по алгоритму и специалиста по архитектурам. Достигнуто ускорение 16 раз на системах с общей памятью, 50 раз – на игровой приставке Sony PS3 и более 100 раз на карте NVidia Tesla C1060. Время исходного последовательного исполнения программы сократилось с 7 часов (прохождение волны от побережья Чили до Японии) до 15 минут.

### Литература

1. Shuto N., Imamura F., Yalciner A. C., Ozyurt G.: TUNAMI N2: Tsunami Modeling Manual. [Electronic resource]. Mode of Access: <http://tunamin2.ce.metu.edu.tr/>
2. Titov V. V. Numerical Modeling of Tsunami Propagation by using Variable Grid // The IUGG/IOC International Tsunami Symposium. Computing Center Siberian Division USSR Academy of Sciences. Novosibirsk. USSR. 1989. P. 46–51.
3. Titov V., Gonzalez F. Implementation and Testing of the Method of Splitting Tsunami (MOST) // Technical Memorandum ERL PMEL-112, National Oceanic and Atmospheric Administration, Washington DC, 1997.
4. Borrero J. C., Sieh K., Chlieh M., Synolakis C. E. Tsunami Inundation Modeling for Western Sumatra // Proceedings of the National Academy of Sciences of the USA. 2006. Vol. 103 (52).
5. Боресков А. В., Харламов А. А. Основы работы с технологией CUDA. ДМК, Москва. 2010.
6. High-Precision Software Directory [Electronic resource]. Mode of Access: <http://crd.lbl.gov/~dhbailey/mpdist>.