ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 2(16), 2013, с. 21–42

УДК 004.4'4, 004.42

Н. А. Богословский, Ю. А. Климов, А. В. Савельев,
Д. К. Шалыга

Разработка экспериментального комплекса суперкомпьютерного моделирования на основе кода на языке Matlab

Аннотация. Авторами была выполнена разработка экспериментального образца комплекса программных средств суперкомпьютерного моделирования характеристик полупроводниковых наноструктурированных сред для сверхбыстрой модуляции света в системах передачи информации в волоконно-оптических линиях связи. В статье описывается процесс создания данного комплекса на языке C++ на основе разработанного авторами программного кода на языке Matlab. Приводится перечень основных проблем, с которыми пришлось столкнуться авторам, и предлагаются способы их решения. Во второй части статьи приводятся результаты экспериментального исследования, показывающие высокую эффективность и масштабируемость разработанного комплекса.

Ключевые слова и фразы: высокопроизводительные вычисления, суперкомпьютерное моделирование, параллельное программирование, преобразование программ, Matlab.

Введение

Во многих областях науки постоянно возникают потребности в различных вычислениях. Если еще в середине 20 века можно было обойтись силами одного человека или группы, то сейчас объемы необходимых вычислений таковы, что невозможно обойтись без использования вычислительных систем. В современном мире в качестве таких систем обычно используется компьютер в сочетании с удобной вычислительной средой. В настоящее время одной из самых популярных сред является Matlab [1], программирование в которой происходит на одноименном языке.

Работа выполнена при поддержке Министерства образования и науки Российской Федерации (госконтракт № 07.514.11.4148).

Система Matlab активно используется большим числом разработчиков аппаратуры, инженеров, математиков. Однако производительность данной среды, как будет показано далее, невысока по меркам высокопроизводительных вычислений. Эта система разрабатывалась достаточное время и в первую очередь ориентируется на удобное и эффективное использование, а не на высокую производительность (тем самым объясняются многие принятые при создании Matlab решения). Для примера отметим, что язык программирования Matlab является интерпретируемым, а не компилируемым языком. С одной стороны, это позволяет производить вычисления и сразу анализировать получаемые результаты, но с другой — заметно снижает производительность системы.

Возникает резонный вопрос: почему же до сих пор кто-то пользуется этой средой? Ответ в том, что она уже привычна и удобна многим людям, которые занимаются не высокопроизводительным параллельным программированием, а инженерными расчетами. И, что немаловажно, Matlab намного проще в освоении, нежели, например, C++.

Тем не менее, производительность системы тоже важна, и когда потеря производительности составляет порядки, требуется изменять ситуацию. По этой причине в мире ведутся работы по созданию автоматических утилит, переводящих код из языка Matlab в параллельный код на эффективных языках программирования.

Одной из таких утилит является продукт Matlab Complier [2] от MathWorks. Он переводит код на языке Matlab в код на языке C++, а также имеет удобный интерфейс, встроенный в родной интерфейс Matlab. Но есть ключевые проблемы, которые не позволили авторам статьи воспользоваться данным инструментом:

(1) Во-первых, система выдает код, содержащий множество обращений к статической библиотеке из-за существенных различий в семантике языков (некоторые из которых будут описаны далее). Код получается несопровождаемым и не предназначенным для дальнейших ручных и автоматических преобразований.
(2) Во-вторых, код, созданный таким образом, предназначен для последовательного исполнения (если программист самостоятельно не озаботился написанием параллельного кода), хотя в настоящее время даже на домашних компьютерах стоят процессоры с несколькими ядрами.

Отметим, что эти проблемы касаются кода, написанного пользователем. Встроенные функции зачастую имеют достаточно эффективную, но закрытую реализацию.

Также были найдены другие утилиты, позволяющие транслировать код в C, Python или Java. Увы, эти утилиты не заслуживают внимания, так как код, полученный с их помощью, даже не всегда мог компилироваться.

Перед авторами стояла задача создания параллельного комплекса суперкомпьютерного моделирования характеристик полупроводниковых наноструктурированных сред для сверхбыстрой модуляции света в системах передачи информации в волоконно-оптических линиях связи. Разработка данного комплекса была проведена в два этапа:

(1) На первом этапе была разработана программа на языке Matlab.
(2) На втором этапе эта программа была переписана в параллельную программу на языке C++.

Использование промежуточной программы позволило в одном проекте задействовать коллективы разработчиков, обладающих знаниями в совершенно различных областях.

На первом этапе специалисты в области физики и математического моделирования полупроводниковых наноструктурированных сред разработали код на языке Matlab. Использование данной среды позволило разработать корректную программу, проанализировать и проверить получаемые данные.

На втором этапе специалисты в параллельном программировании и суперкомпьютерных технологиях разработали эффективную параллельную реализацию, эквивалентную программе на языке Matlab. Второму этапу разработки и посвящена данная статья.

Дальнейшее изложение построено следующим образом. В первом разделе в общих чертах описан процесс перевода кода с языка Matlab в код на языке C++. Во втором разделе приведены основные проблемы, с которыми пришлось столкнуться авторам, и использованные пути решения данных проблем. В третьем разделе приведены результаты экспериментальных исследований полученного комплекса.

1. Разработка кода на языке C++, аналогичного коду на языке Matlab

В данном разделе приведены общие изменения структуры кода при переходе от программы на языке Matlab к программе на языке C++.

1.1. Структура программы на Matlab

Проект на языке Matlab состоит из 11 файлов, написанных на одноимённом языке. В каждом файле описана ровно одна функция. Корневым файлом является `matrix.m`, в котором содержится одноименная функция. Граф вызова функций изображен на рис. 1.

Для удобства сопровождения кода общая структура была сохранена: одному файлу на языке Matlab соответствует один файл на языке C++.

В системе Matlab пользователь в зависимости от текущих требований вызывает те или иные реализованные функции. Для использования на суперкомпьютере необходимо четко разделить весь программный код по отдельным модулям в зависимости от их использования. В данном проекте было принято решение выделить части, отвечающие за подготовку, ввод и вывод данных, в модуль ввода-вывода, а части, отвечающие только за математический расчет — в расчетный модуль. Такой подход позволяет один раз подготовить исходные данные для расчетов с помощью модуля ввода-вывода, а затем несколько раз использовать расчетный модуль.

Также отметим, что именно расчетный модуль нуждается в высокоэффективной параллельной реализации, а модуль ввода-вывода может быть реализован без использования систем параллельного программирования.

В данной программе функция `structure`, отвечающая за подготовку исходных данных для расчета, и все входящие в нее функции были помещены в модуль ввода-вывода, а функции `matrix` и `LaserOut`, производящие расчет, — в расчетный модуль. Взаимодействие между модулями было организовано через вспомогательные файлы: модуль ввода-вывода по описанию конфигурации вычисляет данные и записывает их в файл, содержащий начальные данные для расчета, а расчетный модуль читает данные из этого файла и использует их в качестве начальных данных для расчета.

1.2. Используемые макросы, функции и библиотеки

Для написания программы была использована библиотека STL: в основном использовались векторы и потоки. Помимо них использовались списки, отображения, общие алгоритмы.

Были введены следующие макросы, описывающие заголовки циклов:

```
#define FOR(i, a, b) for (int i(a), _b(b); i <= _b; ++i)
#define REP(i, n) for (int i(0), _n(n); i < _n; ++i)
```

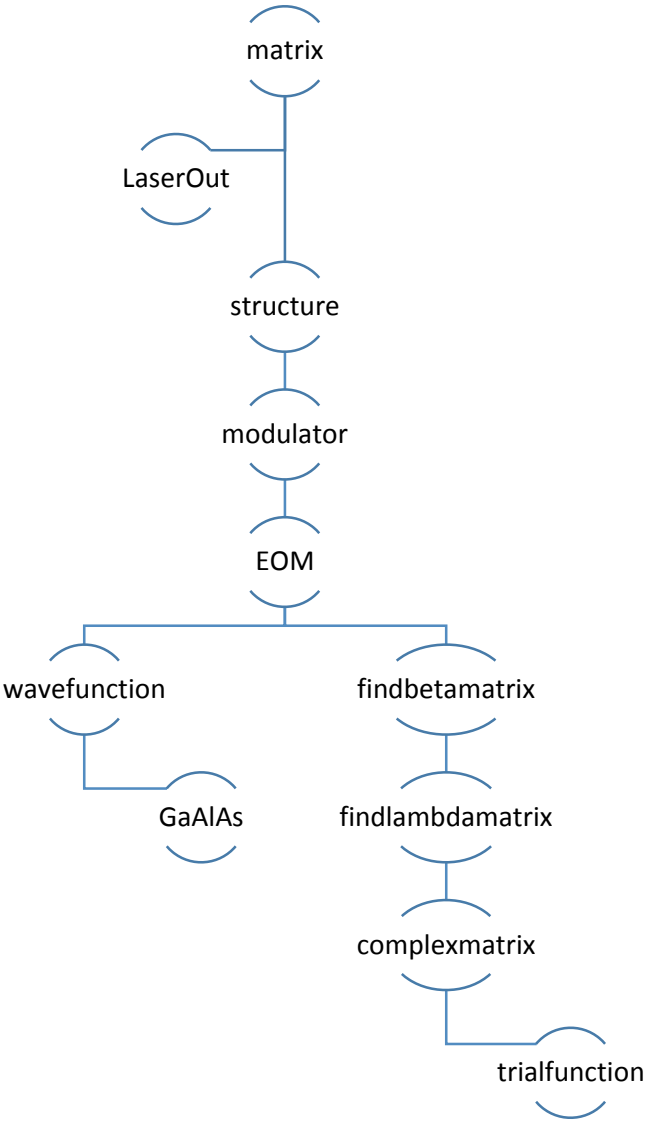


Рис. 1. Граф вызова функций в изначальной программе

Эти макросы заметно уменьшают громоздкость кода в программе и позволяют ей быть более понятной и прозрачной.

При трансляции использовались вспомогательные функции, которые являлись библиотечными функциями в Matlab. Например:

```
template<class T> inline T sqr(T x) {
    return x*x;
}
template<class T> string toString(T n) {
    ostringstream ost;
    ost<<n;
    ost.flush();
    return ost.str();
}
```

2. Основные проблемы при трансляции

При трансляции программ с языка Matlab на C++ были решены следующие основные проблемы:

- разная адресация массивов;
- разный синтаксис вызова функций;
- разная арифметика по умолчанию;
- глобальные переменные;
- динамическая типизация в языке Matlab;
- использование функций линейной алгебры.

2.1. Разная адресация массивов

Суть первой проблемы состоит в том, что нумерация массива в Matlab начинается с единицы, а в C++ с нуля. Для решения этой проблемы было решено менять индекс при обращении к ячейке массива на число, на единицу меньшее.

Чтобы код оставался внешне похожим, аналогичным образом изменен и индекс цикла. То есть если исходный цикл был от 1 по N, то после преобразования цикл на C++ будет от 0 по $N-1$. В результате код внутри многих циклов остался без изменений.

Например, в функции `EOM` встречается следующий фрагмент кода:

```
for l = 1:1:mSize
    Fz(l) = 0;
    for z = 1:1:n
        Fz(l) = Fz(l)+Fi(z, l)*1E7;
    end
```

```
end
```

При переводе он перешел в следующий код на C++:

```
REP(l, mSize) {
    Fz[l] = 0;
    REP(z, n) {
        Fz[l] += Fi[z][l]*1e7;
    }
}
```

Как видно, внутренний текст циклов почти не изменяется, что позволяет быть программе хорошо читаемой.

Тем не менее, этот подход имеет сложности, если индекс цикла используется не только как индекс массивов. Например, в функции `wavefunction` есть следующие строки кода:

```
for i = 1:1:(n-1)/2-30
    x((n+1)/2+i) = i*h;
    x((n+1)/2-i) = -i*h;
end
```

Для перевода на язык C++ можно было бы сделать следующее:

```
REP(i, (n-1)/2-30) {
    x[(n+1)/2+i] = i*h;
    x[(n+1)/2-i] = -i*h;
}
```

Однако это будет некорректно. Во-первых, во втором присваивании в индексе массива переменная `i` присутствует со знаком минус, и, соответственно, сдвиги индексов цикла и элементов массива не компенсируют друг-друга. Во-вторых, измененная переменная `i` присутствует в правых частях присваиваний.

Проблема в том, что мы сдвинули значение переменной `i` (являющейся индексом цикла) относительно того, что было в Matlab. Поэтому для правильной программы нам нужно скорректировать значение в правых частях присваиваний: `i` заменить на `i+1`.

Аналогично нужно поправить и индексы массива с учетом того, что у нас элементы массива сдвинуты на 1. Таким образом, во втором присваивании нужно сдвинуть индекс на 2 в «обратную» сторону, чтобы элементы «встали на свои места».

В итоге правильный программный код выглядит следующим образом:

```
REP(i, (n-1)/2-30) {
    x[(n+1)/2+i] = (i+1)*h;
    x[(n+1)/2-i-2] = -(i+1)*h;
}
```

2.2. Разный синтаксис вызова функций

Стандартный вызов функции на языке Matlab выглядит следующим образом:

```
[<out1>, <out2>....] = <name>(<in1>, <in2>, ...);
```

где `inX` и `outX` означают входные параметры и результаты функции соответственно. При использовании функций без результата (то есть процедур) или функций, возвращающих одно значение, не возникает никаких проблем с переносом кода на язык C++. Однако при использовании функций, возвращающих несколько значений, возникает проблема для использования в программах на языке C++. Причина в том, что функции в языке C++ возвращают только не более одного значения, а вызов функции выглядит следующим образом:

```
<variable> = <name>(<in1>, <in2>, ....);
```

Одним из вариантов может стать передача значений через массив: создается массив указателей типа `void*` и по этим адресам записываются значения, выдаваемые аналогичной функцией Matlab в конце исполнения.

Такой подход был отвергнут ввиду загромождения и нечитабельности кода, а также дополнительного использования памяти.

В итоге решением стало использование передачи параметров по ссылке. Функции на языке Matlab вида:

```
[<out1>, <out2>, ...] = <name>(<in1>, <in2>, ...);
```

переходили в функции на C++ вида:

```
<name>(<in1>, <in2>, ..., <out1>, <out2>, ...);
```

Для примера можно рассмотреть следующий отрывок файла `complexmatrix.m`:

```
for i = 1:1:mSize
    for j = 1:1:mSize
        hl1 = floor((i-1)/ne)+1;
        el1 = i-(hl1-1)*ne;
        hl2 = floor((j-1)/ne)+1;
        el2 = j-(hl2-1)*ne;
        [energy, norm12] = trialfunction(lam, bet, el1, el2,
                                         hl1, hl2);
```

```
        M(i, j) = energy;
        N(i, j) = norm12;
    end
end
```

В данном примере используются двумерные массивы, и необходимо учитывать измененную адресацию. Для этого были изменены стартовые значения индексов циклов. В итоговом файле на языке C++ имеются следующие строки кода:

```
REP(i, mSize) {
    M[i].resize(mSize);
    N[i].resize(mSize);
    REP(j, mSize) {
        hl1 = i/ne+1;
        el1 = i+1-(hl1-1)*ne;
        hl2 = j/ne+1;
        el2 = j+1-(hl2-1)*ne;
        trialfunction(lam, bet, el1, el2, hl1, hl2, energy12,
                      norm12);
        M[i][j] = energy12;
        N[i][j] = norm12;
    }
}
```

При этом объявление функции `trialfunction` выглядит следующим образом:

```
void trialfunction(double lam, double bet, int ElectronLevel1,
                   int ElectronLevel2, int HoleLevel1,
                   int HoleLevel2, double &ExEnergy,
                   double &norm12);
```

Видно, что «выходные» параметры `ExEnergy` и `norm12` передаются по ссылке.

2.3. Разная арифметика по умолчанию

В языке Matlab все численные переменные имеют внутренний тип `double`. Таким образом, даже если результат вычислений имеет целое значение, он будет храниться как число с двойной точностью. Это становится важным при проведении промежуточных вычислений над числами, являющимися целыми по своему происхождению и по логике алгоритма. Так, например, следующая подпрограмма на языке Matlab:

```
a = 1;
b = 2;
c = 6;
d = a/b*c;
d
```

выдаст значение 3 на выходе. Тем не менее, при прямом переводе этой программы на язык C++:

```
int a, b, c;
a = 1;
b = 2;
c = 6;
d = a/b*c;
cout<<d;
```

на выходе будет результат 0. Причина этого состоит в том, что Matlab при промежуточных вычислениях всегда использует тип `double`, а программы, написанные на C++ используют тот тип, которым обладают переменные, входящие в выражение. В данном случае Matlab при вычислениях будет иметь следующие результаты:

```
a/b = 1/2 = 0.5
(a/b)*c = 0.5*6 = 3
```

Таким образом, все переменные изначально были целыми числами и результат вычисления тоже является целым числом.

В отличие от Matlab, на C++ будет использовано целочисленное деление, и исполнение такой программы будет выглядеть иначе:

```
a/b = 1/2 = 0
(a/b)*c = 0*6 = 0
```

Это значит, что несмотря на то, что результат является целым числом, он неверен (не совпадает с результатом на Matlab). Для решения такого рода проблем использован следующий метод: все операции умножения переносятся в начало выражения и все операции деления — в конец. Таким образом исправленная программа имеет следующий вид:

```
int a, b, c;
a = 1;
b = 2;
c = 6;
d = a*c/b;
cout<<d;
```

В этом случае ответ был бы равен 3, как и должно быть по семантике, предполагаемой программистом на языке Matlab.

В функции `trialfunction` встречается следующий фрагмент кода:

```
for j = 1:1:(2*k-1)
    ze = (n-1)/2/k*i+1;
    zh = (n-1)/2/k*j+1;
    if or((mod(i, 2)==1), (mod(j, 2)==1))
        norm = norm + fe2(ze)*fe2(ze)*fh2(zh)*fh2(zh) *
               exp(-2*sqrt(bet)*abs(x(ze)-x(zh))/lam) *
               (1+2*sqrt(bet)*abs(x(ze)-x(zh))/lam);
    end
end
```

В нем используется умножение после операций сложения, и все переменные, участвующие в выражении, целочисленные. Таким образом, при переводе на язык C++ возникает упомянутая ошибка. Для правильного перевода важно не забыть про особенности арифметики на языке C++. В этом случае итоговый код будет выглядеть так:

```
REP(j, 2*k-1) {
    ze = (n-1)*(i+1)/2/k;
    zh = (n-1)*(j+1)/2/k;
    if (((i%2)==0) || ((j%2)==0)) {
        norm += fe1[ze]*fe1[ze]*fh1[zh]*fh1[zh] *
                exp(-2*sqrt(bet)*fabs(x[ze]-x[zh])/lam) *
                (1.0+2.0*sqrt(bet)*fabs(x[ze]-x[zh])/lam);
    }
}
```

В данном фрагменте кода есть еще одна возможность неправильного перевода:

```
mod(i, 2)==1
```

Дело в том, что при переводе кода на язык C++ все индексы мы сместили на единицу, а это значит, что и остаток от деления на 2 тоже изменится. Таким образом, используется другое сравнение:

```
(i%2)==0
```

2.4. Глобальные переменные

Все переменные, указанные без ключевого слова `global`, интерпретатор языка Matlab считает локальными. Для указания переменных, которые должны сохранять свое значение между всеми функциями и всеми файлами, используется ключевое слово `global`. Таким образом, написанная в файле `trialfunction.m` строка:

```
global m0 e hp;
```

скажет интерпретатору о том, что указанные переменные нужно считать глобальными, видимыми во всех функциях. В отличие от языка C++, такого рода объявления могут находиться в любом файле до использования указанных переменных.

Для того, чтобы такие переменные можно было использовать без особых трудозатрат и без сильных изменений кода на языке C++, был создан файл `globals.h`, в котором хранятся все переменные, глобальные по логике исполнения программы. Этот файл подключается в файле, содержащем функцию `main()`, до подключения остальных файлов, которые были в оригинале написаны на Matlab. Таким образом, эти переменные становятся доступными во всех функциях и исполняют ту же роль, что исполняли глобальные переменные в оригинальной программе. Так, например, указанная выше строка кода выглядит в файле `globals.h`:

```
const double m0 = 9.10938188e-28;
const double e = 4.8032e-10;
const double hp = 1.05457e-27;
```

Значения указаны непосредственно в правых частях деклараций этих переменных, так как нет возможности указывать значения переменных с ключевым словом const после их создания. В программе на Matlab эти значения указывались позже.

2.5. Динамическая типизация в языке Matlab

Язык Matlab является интерпретируемым, что позволяет использовать некоторые преимущества, такие, например, как динамическая типизация. Это позволяет никогда не указывать тип переменной при ее создании — интерпретатор вычисляет тип для результата выражения и приписывает переменной именно этот тип. Таким образом, тип переменной может меняться по ходу программы, что невозможно в программах на языке C++.

В функции complexmatrix можно найти следующие строки кода:

```
[energy, norm12] = trialfunction(lam, bet, el1, el2, hl1,
                                 hl2);
M(i, j) = energy;
```

Зная, что функция `trialfunction` возвращает два значения типа `double`, при выполнении присваивания интерпретатор припишет переменной `energy` тип `double`, и в ячейки матрицы `M` будут записаны значения типа `double`. Сотней строк ниже в той же функции встречается следующий код:

```
[vectMMM, energyMMM] = eig(MMM);
for i = 1:1:mSize
    D1(i) = energyMMM(i, i);
end
energy = sort(D1);
```

Интерпретатор знает, что функция `eig` возвращает вектор и матрицу. Переменной `D1` приписывается тип вектор, когда идет обращение по индексу. После чего идет сортировка вектора и значение отсортированного вектора записывается в упомянутую выше переменную `energy`. Таким образом интерпретатор изменяет тип переменной `energy` с `double` на `vector<double>`.

В языке C++ это невозможно. Для того, чтобы обойти такие ситуации, было решено создавать временные переменные с похожим на оригинал названием. После переработки исправленный код на языке C++ выглядит следующим образом:

```
double energy12;
...
trialfunction(lam, bet, el1, el2, hl1, hl2, energy12, norm12);
M[i][j] = energy12;
N[i][j] = norm12;
...
vector<vector<double>> vectMMM, vectM;
vector<double> energyMMM;
eig(MMM, vectMMM, energyMMM);
energy = energyMMM;
sort(energy.begin(), energy.end());
```

В исправленном коде появилась новая переменная `energy12`, которой не было в изначальном коде. Это вынужденная мера, так как невозможно одной переменной в языке C++ иметь два разных типа.

2.6. Использование функций линейной алгебры

Самой большой сложностью при переходе с языка Matlab на язык C++ является то, что требуется реализация функций линейной алгебры. В языке Matlab очень многие функции уже реализованы и встроены в язык или содержатся в библиотеке. Некоторыми из таких функций являются умножение матрицы на вектор и сложение векторов. Так, в функции `matrix` встречается использование обеих упомянутых операций:

```
right = -2*T*E2+T*E1-2*dt*R*E2+1/2*dt*R*E1;
```

В данном примере `T` и `R` являются матрицами, а `right`, `E1`, `E2` - векторами; `dt` является скаляром. Для перевода этого отрывка кода изначально применялась библиотека BLAS [3], но ввиду чрезмерного количества перемещений информации в памяти ради вызова требуемых функций было принято решение заново реализовать эти математические операции. В итоге полученные строки кода на C++ выглядят следующим образом:

```
REP(i, N) {
    right[i] = (-2*E2[i]+E1[i])*T[i][i] +
               (-2*dt*E2[i]+0.5*dt*E1[i])*R[i][i];
}
```

При таком подходе важно отметить, что было учтено несколько факторов из решаемой задачи. Например, тот факт, что матрицы `T` и `R` всегда будут диагональными, и нет нужды умножать на элементы, стоящие вне диагонали.

Помимо матричной арифметики в данной программе используются еще две операции из линейной алгебры: решение уравнения $Ax = B$, где A — матрица, x и B — вектора; нахождение собственных векторов для матрицы.

Каждая из этих задач решалась отдельно.

2.7. Нахождение собственных векторов

Для нахождения собственных векторов матрицы можно использовать несколько стандартных алгоритмов. В случае данной задачи очень важна точность и не сильно важно время, так как вычисление собственных векторов происходит фиксированное число раз независимо от входных данных. По этой причине было принято решение использовать библиотеку SLEPc [4], позволяющую варьировать точность нахождения собственных векторов.

Если отбросить все процессы выделения памяти и обозначения переменных, то функция, аналогичная функции `eig` из Matlab, на C++ выглядит так:

```
void eig (const vector<vector<PetscScalar>> &A,
          vector<vector<double>> &VV, vector<double> &D) {
    ...
    EPSSetTolerances(eps, 1e-14, 10000);
    EPSSetType(eps, EPSLAPACK);
    EPSSolve(eps);
    REP(i, Asize) {
        EPSGetEigenvector(eps, i, pVec, cVec);
        VecGetArray(pVec, &ptr);
        VV[i].resize(Asize);
        REP(j, Asize) {
            VV[i][j] = ptr[j];
        }
        EPSGetEigenvalue(eps, i, &pptr1, &pptr2);
        D[i] = pptr1;
    }
    ...
}
```

Здесь используется функция `EPSSolve`, позволяющая найти все собственные вектора для заданной матрицы. Далее поочередно для каждого вектора находится собственное значение с помощью функции `EPSGetEigenvalue` и записывается в соответствующие переменные.

В функции `complexmatrix` на языке Matlab нахождение собственных векторов встречается, например, в таком виде:

```
[vectN, EigValN] = eig(N);
```

что на языке C++ выглядит следующим образом:

```
vector<vector<double>> vectN;
vector<double> EigValN;
eig(N, vectN, EigValN);
```

2.8. Решение уравнения $Ax = B$

Для решения уравнения с матрицей и вектором используется библиотека PETSc [5], а в ней — метод BiCGStab с предварительным неполным LU-разложением. Это позволяет на малых матрицах получать абсолютно точный результат за счет неполного LU-разложения, а на больших матрицах — не тратить слишком много времени на

разложение, доводя решение до нужной точности итерационным методом. В функции `matrix` содержится следующий кусок кода:

```
E3 = A\right;
```

В этом куске кода `A` является матрицей, `right` и `E3` являются векторами. По логике программы подразумевается, что все нужные размеры и параметры согласованы. По итогам выполнения этой строки в `E3` будет лежать вектор, удовлетворяющий условию `A*E3 = right`.

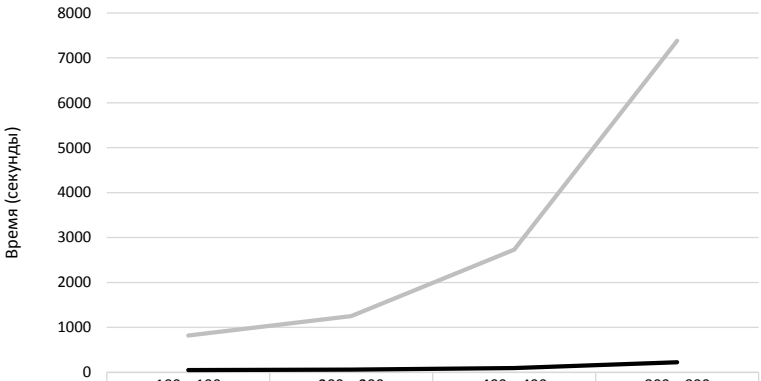
Для того, чтобы реализовать этот алгоритм, на языке C++ была написана функция `solve`, использующая функционал библиотеки PETSc. Если опустить объявления переменных и вызовы выделения памяти, то функция будет выглядеть следующим образом:

```
vector<PetscScalar> solve (
        const vector<vector<PetscScalar>> &A,
        const vector<double> &rhs) {
    ...
    PCSetType(pc, PCILU);
    KSPSetTolerances(ksp, 1e-16, 1e-16, PETSC_DEFAULT,
                     PETSC_DEFAULT);
    KSPSetType(ksp, KSPBCGS);
    ...
    KSPSolve(ksp, pVec, result);
    VecGetArray(result, &ptr);
    REP(i, Asize) {
        ret[i] = ptr[i];
    }
    ...
    return ret;
}
```

Эта функция получает на вход матрицу и вектор, на выходе выдается вектор. Все это делается в рамках классов из библиотеки STL [6], таким образом гарантируется высокая производительность.

С использованием данной функции производится обращение матрицы для последующего использования в вычислениях. Так как изначально в коде Matlab этого не было, то привести оригинал невозможно. На C++ код выглядит следующим образом:

```
REP(i, N) {
    if (i%size == myid) {
        REP(q, N) {
            right[q] = 0.0;
        }
        right[i] = 1.0;
```



	100 x 100	200 x 200	400 x 400	800 x 800
C++	52	59	95	225
Matlab	820	1250	2730	7380

Рис. 2. Сравнение времени исполнения программы на разных языках

```
        aT[i] = solve(A, right);
    }
}
```

В данном случае учитывается, что программа запускается в параллельном режиме, и нахождение векторов будет распределено по узлам вычислительной системы.

3. Результаты экспериментальных исследований

3.1. Сравнение последовательных реализаций

Исходный код на языке Matlab сравнили с полученным кодом на языке C++ на рабочем компьютере со следующими характеристиками:

- x64-совместимый 64-битный процессор Intel Core i5-3570 (4 ядра, 3,4ГГц, 6Мбайт кэш L3).
- Оперативная память 32Гбайт.
- Дисковое пространство 500Гбайт.
- Операционная система: Debian GNU/Linux 8.0 (testing), ядро версии 3.2.0.

Сравнение последовательных версий кода на языке Matlab и кода на языке C++ с одинаковыми размерами исходных данных приведено на рис. 2.

Результаты тестирования показывают, что по сравнению с оригинальным кодом на языке Matlab выигрыш во времени составляет больше, чем в 30 раз.

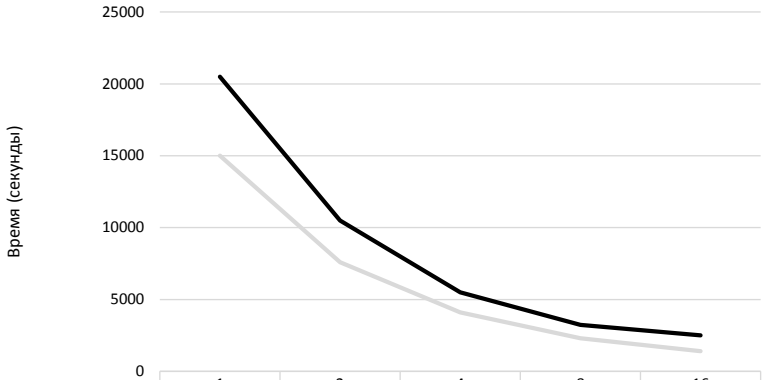
3.2. Исследование эффективности параллельной реализации

Экспериментальные исследования эффективности параллельной реализации проводились на суперкомпьютере К-100, установленном в ИПМ им. М.В. Келдыша РАН. Данная вычислительная система обладает следующими характеристиками:

- Общая пиковая производительность — 100 Тфлопс.
- 64 вычислительных узла:
  - два x64-совместимых 64-битных процессора Intel Xeon X5670 (на каждом процессоре по 6 ядер, 2,93ГГц, 12Мбайт кэш L3), 12 ядер на узел;
  - оперативная память 96Гбайт;
  - дисковое пространство 500Гбайт;
  - коммуникационный адаптер системной сети InfiniBand QDR;
  - коммуникационный адаптер коммуникационной сети «МВС-экспресс»;
  - три графических ускорителя nVidia Fermi C2050 (на каждом ускорителе по 448 CUDA-ядер и 2,5ГБайт памяти).
- Система хранения данных — общий объем 16Tb.
- Сеть:
  - системная сеть Infiniband QDR;
  - коммуникационная сеть «МВС-экспресс»;
  - внутренняя сеть Gigabit Ethernet.
- Программное обеспечение:
  - операционная система: Linux (SLES 11 SP1/CentOS 5.5), ядро версии 2.6.32;
  - параллельные средства: MPI, OpenMP, SHMEM, DVM, CUDA;
  - языки программирования: C/C++, Fortran, Java.

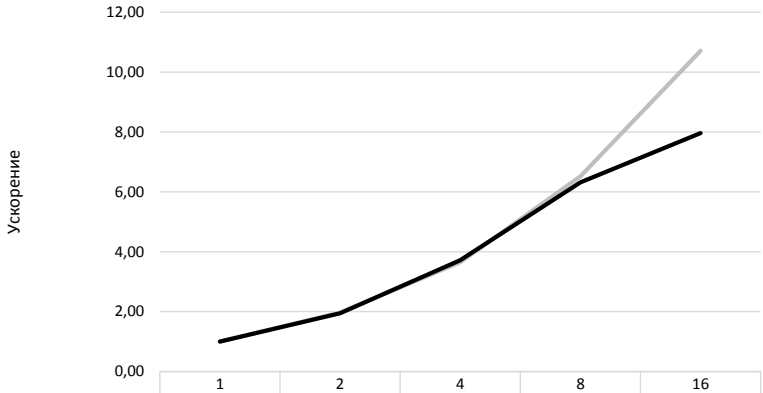
В качестве библиотек BLAS [3] и LAPACK [7] использовалась открытая реализация f2cblaslapack [8]. В качестве компилятора использовался Intel Composer 11.

С целью тестирования масштабируемости разработанного кода были использованы два варианта тестовых файлов с разным размером исходных данных. График зависимости времени исполнения от количества узлов предоставлен на рис. 3, а график зависимости ускорения исполнения от количества узлов предоставлен на рис. 4.



	1	2	4	8	16
13000 x 13000	15000	7600	4100	2300	1400
6500 x 6500	20500	10500	5500	3240	2510

Рис. 3. График зависимости времени исполнения от количества узлов



	1	2	4	8	16
13000 x 13000	1,00	1,97	3,65	6,52	10,71
6500 x 6500	1,00	1,95	3,72	6,32	7,96

Рис. 4. График зависимости ускорения исполнения от количества узлов

Результаты показывают высокий коэффициент масштабируемости — более 50%, что является отличным результатом для систем такого уровня.

Заключение

Широкий круг специалистов использует Matlab для быстрой и качественной разработки прикладных программ. Однако эффективность Matlab невысока и этот код невозможно исполнить на высокопроизводительных вычислительных системах.

Авторами был разработан экспериментальный образец комплекса программных средств суперкомпьютерного моделирования характеристик полупроводниковых наноструктурированных сред для сверхбыстрой модуляции света в системах передачи информации в волоконно-оптических линиях связи на основе прототипа на языке Matlab. В процессе разработки были решены проблемы преобразования кода на языке Matlab в код на языке C++.

Результаты экспериментальных исследований показали, что производительность разрабатываемой системы выросла на порядок по сравнению с оригинальным способом вычислений на Matlab. По графикам, приведенным выше, можно увидеть, что высокая эффективность достигается как используемым языком программирования, так и эффективным распараллеливанием программы.

По итогам проведенных работ можно сделать вывод, что существует возможность создать утилиту, позволяющую автоматизированно генерировать код на языке C++ по коду на Matlab, который будет возможно с незначительными затратами доводить до оптимального. Подобных средств, насколько известно авторам, в открытом доступе нет.

Список литературы

[1] Matlab, The Language of Technical Computing, http://www.mathworks.com/products/matlab/. ↑[]

[2] Matlab Compiler, http://www.mathworks.com/products/compiler/. ↑[]

[3] BLAS, Basic Linear Algebra Subprograms, http://www.netlib.org/blas/. ↑2.6, 3.2

[4] SLEPc, The Scalable Library for Eigenvalue Problem Computations, http://www.grycap.upv.es/slepc/. ↑2.7

[5] PETSc, Portable, Extensible Toolkit for Scientific Computation, http://www.mcs.anl.gov/petsc/. ↑2.8

[6] STL, Standard Template Library, http://www.sgi.com/tech/stl/. ↑2.8

[7] LAPACK, Linear Algebra PACKage, http://www.netlib.org/lapack/. ↑3.2

[8] CLAPACK, f2c'ed version of LAPACK, http://www.netlib.org/clapack/. ↑3.2

Рекомендовал к публикации *чл.-корр. РАН С. М. Абрамов*

Об авторах:

Никита Александрович Богословский

Сотрудник СПб АУ НОЦНТ РАН и ФТИ им. А. Ф. Иоффе, к.ф.-м.н. Принимал активное участие в разработке численных методов расчета характеристик электрооптических модуляторов. Область научных интересов: неупорядоченные полупроводники, высокочастотные электрооптические модуляторы.

e-mail: nikitabogoslovskiy@gmail.com

Юрий Андреевич Климов

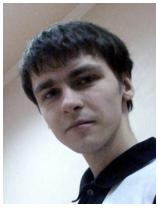
Ведущий инженер-программист ИПС им. А. К. Айламазяна РАН, старший научный сотрудник ИПМ им. М. В. Келдыша РАН, к.ф.-м.н. Разработчик метода специализации на основе частичных вычислений для программ на объектно-ориентированных языках, принимал активное участие в разработке коммуникационного программного обеспечения для сетей SCI, 3D-тор суперкомпьютера СКИФ-Аврора и «МВС-Экспресс» суперкомпьютера К-100. Область научных интересов: суперкомпьютеры, оптимизация и преобразование программ, функциональное программирование.

e-mail: yuklimov@botik.ru

Артем Владимирович Савельев

Старший научный сотрудник СПб АУ НОЦНТ РАН, к.ф.-м.н. Специалист в области численного моделирования физических процессов в полупроводниковых оптоэлектронных приборах. Область научных интересов: численное моделирование, моделирование квантовых явлений, разработка и оптимизация оптоэлектронных приборов.

e-mail: savelev@mail.ioffe.ru



Дмитрий Константинович Шалыга

Сотрудник ИПС им. А. К. Айламазяна РАН, студент механико-математического факультета МГУ им. М. В. Ломоносова. Занимается исследованием характеристик кривых Пеано для трехмерного случая и вопросами автоматического распараллеливания.

e-mail: dmitriy.shalyga@gmail.com

Образец ссылки на эту публикацию:

Н. А. Богословский, Ю. А. Климов, А. В. Савельев, Д. К. Шалыга.

Разработка экспериментального комплекса суперкомпьютерного моделирования на основе кода на языке Matlab // Программные системы: теория и приложения : электрон. научн. журн. 2013. Т. 4, № 2(16), с. 21–42.

URL: http://psta.psiras.ru/read/psta2013_2_21-42.pdf

N. A. Bogoslovskiy, Yu. A. Klimov, A. V. Savelyev, D. K. Shalyga. *Development of supercomputer simulation software based on Matlab source code.*

Abstract. Development of experimental supercomputer software for simulation of semiconductor nanostructure media characteristics for ultrafast light modulation in communication systems based on fiber-optic communication lines has been performed by the authors. The process of creation of the software in the C++ programming language from source code in the Matlab programming language is described. The problems that were uncounted are listed and solutions are suggested. In the second part of the article the results of computer experiments are presented. The results show that the developed software is more that 30 times faster than the original code in Matlab on a single-core computer and is highly scalable on a supercomputer.

Key Words and Phrases: high performance computing, supercomputer simulation, parallel programming, program transformation, Matlab.