ЧИСЛЕННОЕ МОДЕЛИРОВАНИЕ В ФИЗИКО-ТЕХНИЧЕСКИХ ИССЛЕДОВАНИЯХ

УДК 004.272

ПАРАЛЛЕЛЬНЫЕ МЕТОДЫ МОДЕЛИРОВАНИЯ ГЕТЕРОЭПИТАКСИАЛЬНОГО РОСТА НА МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ С ОБЩЕЙ ПАМЯТЬЮ

© **К. В. Павский**[1, 2], **А. Л. Ревун**[1, 2], **С. А. Рудин**[1], **Е. Н. Перышкова**[1, 2], **М. Г. Курносов**[1, 2]

[1] *Институт физики полупроводников им. А. В. Ржанова СО РАН, 630090, г. Новосибирск, просп. Академика Лаврентьева, 13*
[2] *Сибирский государственный университет телекоммуникаций и информатики, 630102, г. Новосибирск, ул. Кирова, 86*
E-mail: pkv@isp.nsc.ru

Предложены решения, позволяющие повысить эффективность исполнения параллельных программ на высокопроизводительных вычислительных системах, при моделировании гетероэпитаксиального роста на многопроцессорных системах с общей памятью. Разработанные алгоритмы ориентированы на выполнение программной реализации моделирования гетероэпитаксиального роста на многопроцессорных NUMA-узлах с общей памятью. Основное требование к эффективному выполнению параллельных программ на ресурсах многопроцессорного узла заключается в учёте архитектурно-ориентированного подхода к реализации алгоритмов передачи данных через разделяемую память NUMA-узлов. Предложенные алгоритмы оптимизации синхронизации в системах с общей памятью повышают эффективность доступа к общей памяти многопроцессорного узла и позволяют сократить время выполнения барьерной синхронизации. Разработанные методы и алгоритмы реализованы в виде программного обеспечения для многопроцессорных NUMA-узлов с общей памятью.

Ключевые слова: высокопроизводительные системы, параллельные программы, барьерная синхронизация, OpenMP.

DOI: 10.15372/AUT20240413
EDN: ZDNHKW

Введение. Квантовые точки (КТ) GeSi привлекают значительное внимание из-за возможности их практического применения в электронных и электронно-оптических приборах нового поколения [1–3], создаваемых на базе существующей кремниевой технологии. Наиболее распространённый способ синтеза подобных структур — эпитаксия, представляющая собой рост одного кристаллического материала на другом с сохранением кристаллической ориентации. Несоответствие кристаллических решёток осаждаемого материала и материала подложки приводит к росту напряжённых слоёв и накоплению в них деформации. При достижении критической толщины плёнки формируются трёхмерные наноостровки. Такой процесс называется ростом по механизму Странского — Крастанова. Однако при осаждении Ge на гладкую подложку Si места зарождения наноостровков Ge располагаются хаотично и не поддаются точному контролю, что сильно ограничивает область возможных применений. Упорядоченные ансамбли КТ обладают более однородным химическим составом и энергетическим спектром, что критически важно для их электронно-оптических свойств, а точный контроль расположения КТ в пространстве открывает возможность адресного обращения к каждой из них. При синтезе подобных

структур необходимо учитывать множество параметров, таких как рельеф поверхности, температура, скорость осаждения, количество осаждаемого материала. Одним из способов исследования гетероэпитаксиального роста является компьютерное моделирование, которое позволяет подробно изучать процессы, происходящие на поверхности и в объёме структуры, и отслеживать состояние каждого атома структуры в любой момент времени. Ввиду сложности вычислений актуальной является разработка параллельных алгоритмов для моделирования процессов гетероэпитаксии на вычислительных системах.

Цель представленной работы — разработка параллельных методов моделирования гетероэпитаксиального роста на многопроцессорных системах с общей памятью.

Распараллеливание вычислений при моделировании гетероэпитаксиального роста Ge на Si методом Монте-Карло. Для изучения роста структур с КТ разработана физическая модель гетероэпитаксиального роста [4]. В основе модели лежит алмазоподобная кристаллическая решётка. Взаимодействие между атомами описывается потенциалом Китинга [5]. Процесс моделирования роста производится методом Монте-Карло (МК) и состоит из последовательности элементарных событий, выбираемых случайным образом в соответствии с их вероятностями. Возможны события двух типов: осаждение и диффузионный прыжок атома по поверхности. Для учёта изменения деформации в пространстве с течением времени добавлен третий тип событий — тепловые колебания атомов вокруг их равновесных положений согласно распределению Больцмана. Периодический перерасчёт координат одного процента атомов (термический отжиг), выбираемых случайным образом, позволил избежать необходимости вычисления минимума внутренней энергии кристалла после каждого элементарного события, поскольку при достаточно большом их числе энергия системы автоматически будет стремиться к минимуму своей свободной энергии.

Данная модель успешно использована для изучения процессов, происходящих при гетероэпитаксии Ge на Si(100). В том числе были выявлены механизмы роста Ge на подложках Si(100) с созданными в условиях ионного облучения траншеями [6], массивами ямок, образующих фотонный кристалл [7], и обнаружена зависимость положения КТ от формы дна ямок [8]. Результаты моделирования согласуются с экспериментальными данными.

Пакет моделирования гетероэпитаксиального роста реализован на языке программирования C/C++ [4]. На рис. 1 представлена блок-схема его последовательной реализации.

Чтобы повысить эффективность исследования при последовательном моделировании, формируется множество одноранговых задач с разными исходными данными с дальнейшим исполнением на ресурсах вычислительной системы. Под рангом задачи понимается число её параллельных ветвей, т. е. количество требуемых элементарных машин (ЭМ) для решения этой задачи [9]. К примеру, запускается несколько последовательных задач для моделирования процесса осаждения Ge на Si(100) с различной температурой от 400 до 500 °C (рис. 2). Результатом моделирования является набор координат расположения атомов и их тип (например, Si/Ge), записанных в файл. Полученные структуры, представленные наборами атомов, анализируются в других программных комплексах, таких как Ovito [10], RasMol [11].

Процесс моделирования крайне трудоёмкий и сильно зависит от размера структур и температуры. К примеру, время, затраченное на последовательное моделирование осаждения четырёх монослоёв Ge на подложку Si(100) размерами $5 \times 5 \times 14$ нм со скоростью 0,00136 нм/с при температуре 500 °C, составляет 8 мин. При изменении только размера структуры получим: при $7 \times 7 \times 14$ нм — 24 мин, при $11 \times 11 \times 14$ нм — 200 мин, при $27 \times 27 \times 14$ нм — 11 дней. Исследования требуют моделирования структур, в десятки раз превышающих размеры структур, представленных в примере. Также необходимо изменение и других входных параметров: температуры, количества осаждаемых слоёв, рельефа поверхности, что повышает сложность процесса вычисления.

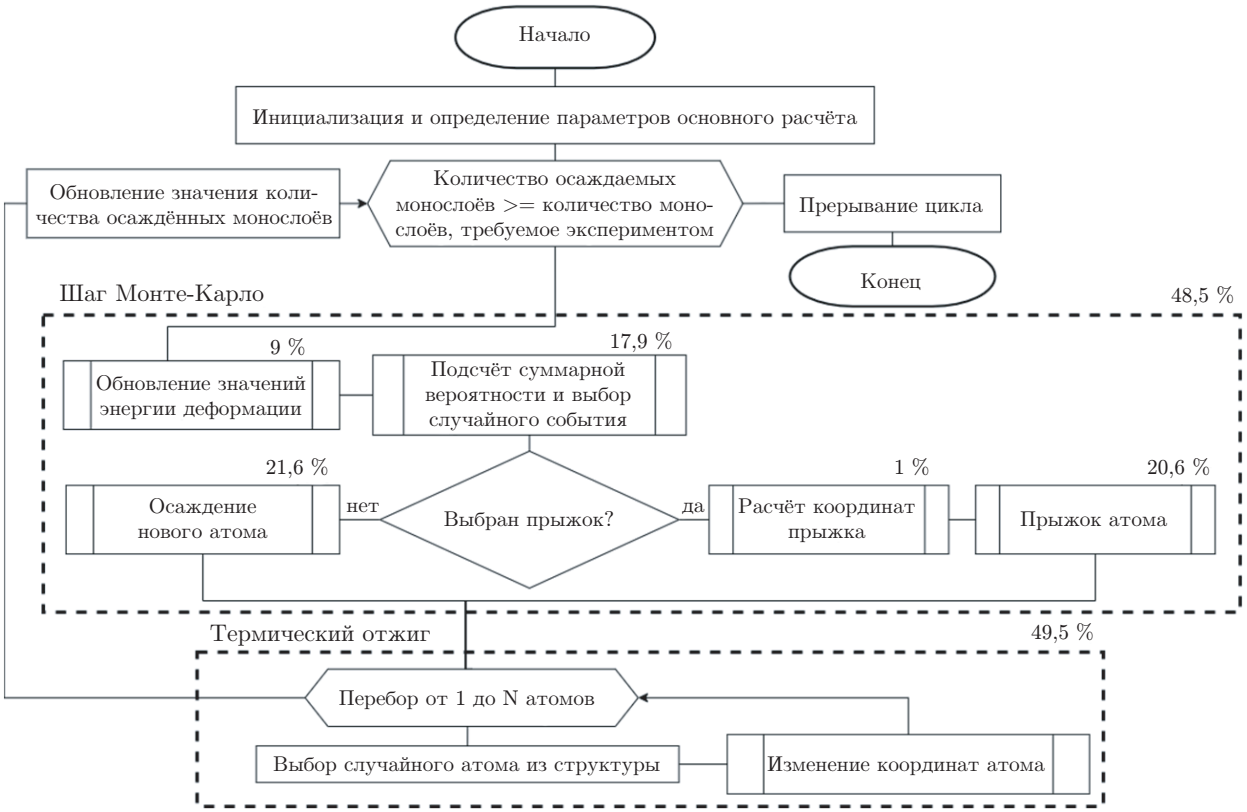

Рис. 1. Блок-схема однопоточной программной реализации моделирования гетероэпитаксиального роста методом МК (над блоками показаны результаты профилирования — процент от общего времени моделирования структуры размерами $11 \times 11 \times 14$ нм при температуре 500 °C)

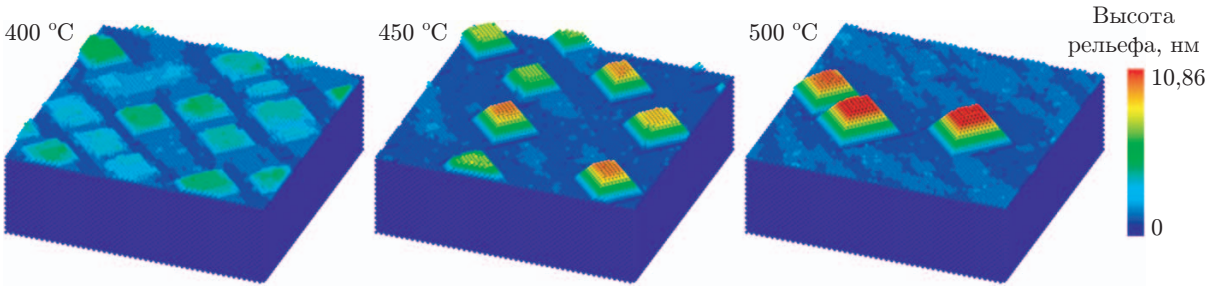

Рис. 2. Результаты моделирования роста Ge на подложке Si(100) при различной температуре. Размеры подложки $27 \times 27 \times 8$ нм, скорость осаждения Ge 0,00136 нм/с

Чтобы сократить время моделирования, авторами разработана многопоточная реализация алгоритма гетероэпитаксиального роста с использованием стандарта OpenMP.

Листинг 1. Псевдокод функции `updateEdef`

```
function updateEdef()
   #pragma omp parallel for private (x, y, z)
   for each atom in queueEdef do
   x, y, z = atom.x, atom.y, atom.z
   #pragma omp critical(first)
   calcEdef(x, y, z) // вычисление значения энергии деформации
   #pragma omp critical(second)
   calcJumpProbability(x, y, z) // вычисление вероятности прыжка атома
   queueEdef.pop()
   end for
end function
```

Листинг 2. Псевдокод функции `calcProbJumpSum`

Входные данные: `jumpProbability` — вероятность прыжка атома
`Lx, Ly, Lz` — размеры моделируемой структуры
Выходные данные: `sum` — суммарная вероятность прыжков атомов

```
function calcProbJumpSum()
   #pragma omp parallel for private(k,j) reduction(+:sum)
   for each i in [2...Lz-2] do
      for each j in [i%2...Ly] do
         j += 2
         for each k in [(i%2)+2*((i/2+j/2)%2)...Lx] do
            k += 4
            sum += jumpProbability(k, j, i)
         end for
      end for
   end for
   return sum
end function
```

При выполнении блока «Шаг Монте-Карло» происходят обновление энергии деформации атомов, напрямую или косвенно участвующих в элементарных событиях (прыжок, осаждение, колебание атома вблизи его равновесного положения), и, как следствие, изменение количества соседей. Функция `updateEdef`, представленная в листинге 1, перебирает каждый атом из очереди `queueEdef` для обновления энергии деформации и вероятности его прыжка в соответствии с текущим положением. Для распараллеливания циклов применяется директива `pragma omp parallel for`, а для ограничения области видимости переменных `x, y, z` — условие `private`. Обеспечить безопасное выполнение функций вычисления значения энергии деформации `calcEdef()` и вероятности прыжка атома `calcJumpProbability` в параллельном регионе помогает директива `pragma omp critical` с присвоенными различными именами для одновременного выполнения. Проверено, что такая реализация не нарушает целостности данных, а эффективность на именных секциях `critical` гораздо ниже. Без объявления критической секции нарушается целостность данных, что приводит к критическим ошибкам при моделированиии.

Листинг 3. Псевдокод функции `Jump`

Входные данные: `x, y, z` — рассчитанное направление прыжка
`atoms` — структура для хранения каждого атома кристаллической решётки
Выходные данные: `Nx, Ny, Nz` — новое положение атома в кристаллической решётке
`result` — оценка прыжка `(True, False)`

```
function jump (x, y, z)
... инициализация переменных ...
... сбор соседей атома в массив nbs ...
   #pragma omp parallel for shared (+:result)
   for each i in nbs do
      if result = 1 then continue
         Nx = nbs[i][0]
         Ny = nbs[i][1]
         Nz = nbs[i][2]
      if atoms(Nx, Ny, Nz).type > 0 && !atoms(Nx, Ny, Nz).config then
         result = 0
      end if
   end for
   queueEdef.push(atoms(Nx, Ny, Nz))
   return result
end function
```

Далее в блоке «Шаг Монте-Карло» выполняется подсчёт суммарной вероятности прыжка каждого атома и после происходит выбор элементарного события. Псевдокод функции расчёта суммарной вероятности всех прыжков `calcProbJumpSum` представлен в листинге 2. В зависимости от выпавшего случайного числа, сравниваемого с достигнутой суммарной вероятностью, происходит выбор события: осаждение или прыжок атома. Здесь директива `pragma omp parallel for` и условие `private` используются аналогично листингу 1. Редукцию `sum` обеспечивает директива `reduction(+:sum)`.

Прыжок атома представлен функцией `Jump` в листинге 3. В отличие от листингов 1 и 2, здесь применяется условие `shared` для корректного одновременного выхода из цикла всех потоков.

Затем процесс переходит к выполнению блока «Термический отжиг». Блок представляет собой цикл, вблизи которого осуществляется выбор случайного атома, а далее происходит его смещение вокруг своего равновесного положения. Рассмотрим функцию `doManyAXYZ`, реализующую вышеописанную логику, представленную в листинге 4. Количество итераций внешнего цикла зависит от размера структуры и заданного процента атомов, подверженных смещению. Внешний цикл распараллелен директивой `pragma omp parallel for` с правилом `private` для итератора `i`. В результате все изменённые атомы попадают в список на обновление своих коэффициентов вероятности прыжка и энергии деформации, что произойдёт уже в начале следующей итерации «Шаг Монте-Карло». Для сохранения целостности очереди `queueEdef` применяется `pragma omp critical`.

Оптимизация синхронизации в системах с общей памятью. Многопоточные алгоритмы явно используют барьерную синхронизацию (`#pragma omp barrier`) для обеспечения подготовки данных и согласованного перехода к следующей фазе вычислений или неявно после секций распределения вычислений (например, `#pragma omp for`).

Листинг 4. Псевдокод функции `doManyAXYZ`

Входные данные: `moves_percent` — процент атомов, которые участвуют в процессе «Термический отжиг»
`atoms` — структура для хранения каждого атома кристаллической решётки
`queueEdef` — список атомов для обновления энергии деформации и вероятностей прыжков

```
function doManyAXYZ (x, y, z)
   I= atoms.size() * moves_percent
   #pragma omp parallel for
   for each i in [0...I] do
... выбор случайного атома ...
... инициализация переменных, связанных с атомом n ...
... определение массива соседей атома nbs ...
... расчет смещения атома вокруг своего равновесного положения ...
      #pragma omp critical
      queueEdef.push(atoms(nbs.x[j],nbs.y[j],nbs.z[j]))
   end for
... сохранение изменений в структуру atoms ...
end function
```

Операция барьерной синхронизации блокирует выполнение потока до тех пор, пока каждый поток группы (команды, team) не осуществит её вызов. Поток выходит из операции барьерной синхронизации после того, как все члены группы начали её выполнение. Известные алгоритмы барьерной синхронизации для многопроцессорных систем с общей памятью: глобальный счётчик (central counter) [12, 13], плоское дерево (flat tree), плоское дерево с фазами gather/release [12], объединяющее дерево (combining tree) [14], MCS-барьер [14], турнирный алгоритм (tournament) [15], рассеивающий алгоритм (dissemination) [16] — основаны на использовании счётчиков и флагов в разделяемой памяти, при помощи которых процессы уведомляют друг друга о достижении барьера. Время выполнения барьера зависит от времени доступа к разделяемым переменным, а это в свою очередь зависит и от распределения процессов по ядрам системы. Разработан метод барьерной синхронизации, который группирует процессы, совместно использующие ресурсы на каждом уровне иерархии памяти (разделяют кеш-память L2, L3, контроллер доступа к памяти NUMA-узла, принадлежащему одному процессору). Это позволяет локализовать синхронизацию в созданных группах и тем самым сократить её время.

На первом этапе предложенного метода выполняется анализ иерархии памяти (при инициализации подсистемы OpenMP). Поток определяет число ядер, разделяющих кеш-память уровней L2, L3, число NUMA-узлов и процессоров на вычислительном узле. Вычисляется количество L, и формируется список активных уровней иерархии памяти, которые совместно используются двумя и более ядрами: кеш-память L2, L3, NUMA-узел, процессор (socket, package).

На втором шаге каждый поток определяет своё размещение и номер ядра в пределах многопроцессорного узла. Номер ядра и информация о топологии с предыдущего шага позволяет идентифицировать функциональный модуль `loc[l][i]`, который используется каждым потоком `i` на уровне `l` для коммуникации с другими ядрами. Например, уровень 0 — номера модулей кеш-памяти уровня 3, уровень 1 — номера NUMA-узлов (`loc[1][0]=2` — поток 0 размещён на ядре NUMA-узла 2).

На третьем шаге формируются группы потоков. Каждый поток для всех активных уровней иерархии памяти строит списки потоков, с которыми он разделяет один функциональный модуль (кеш-память L2, NUMA-узел, процессор). Построение начинается с низшего активного уровня. В каждой группе выбирается лидер — поток с наименьшим номером. Лидеры групп становятся членами групп следующего активного уровня. Цикл повторяется для всех активных уровней.

В момент вызова операции барьерной синхронизации каждому потоку известен свой исходный номер `id`, а также номер `group[l].groupid` в каждой группе уровня `l`. В сегменте разделяемой памяти процесс хранит счётчики `group[l].state[groupid]` числа обращений к барьеру на уровне иерархии `l` и имеет доступ к общему флагу `sense` уведомления о достижении барьера каждым потоком.

При входе в барьер каждый поток `id` меняет значение своего флага `sense_local[id]` на противоположное. Далее выполняется синхронизация процессов в группах уровней $0, 1, \ldots, L$. Синхронизация в группе выполняется алгоритмом плоского дерева — лидер

Листинг 5. Псевдокод функции `Barrier`

```
algorithm Barrier()
   id = get_thread_id()
   sense_local[id] = sense_local[id] xor 1
   BarrierLevel(0)
end algorithm
```

Листинг 6. Псевдокод функции `BarrierLevel`

Входные данные: `level` — уровень коммуникаций

```
algorithm BarrierLevel(level)
   grank = group[level].groupid // Индекс потока
   gleader = group[level].group_id_leader // Индекс лидера в группе
   group[level].state[grank]++ // Уведомление лидера о входе
   if grank = gleader then
      while true do // Лидер ожидает уведомления от потоков группы
         narrived = 0
         for child = 0 to group[level].size - 1 do
            if group[level].state[child] >= group[level].state[gleader] then
               narrived++
         end for
         if narrived = group[level].size then break
      end while
      if level + 1 < ngroups then // Лидер переходит на следующий уровень
         BarrierLevel(level + 1)
      else if rank = 0 then
         sense = sense_local[id] // Лидер уведомляет о выходе из барьера
      end if
   else
      while sense != sense_local[id] do // Ожидание уведомления от лидера 0
   end if
end algorithm
```

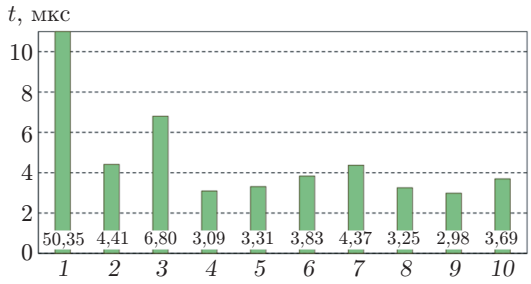

Рис. 3. Время выполнения барьерной синхронизации на 128 ядрах двух процессоров (четыре NUMA-узла по 32 ядра, ARMv8). Обозначения: *1* — sense-reversing, *2* — sense-reversing-counter, *3* — sense-reversing-gr, *4* — combining-tree, *5* — mcs, *6* — tournament, *7* — dissemination, *8* — topo-numa-sk, *9* — topo-numa, *10* — topo-sk

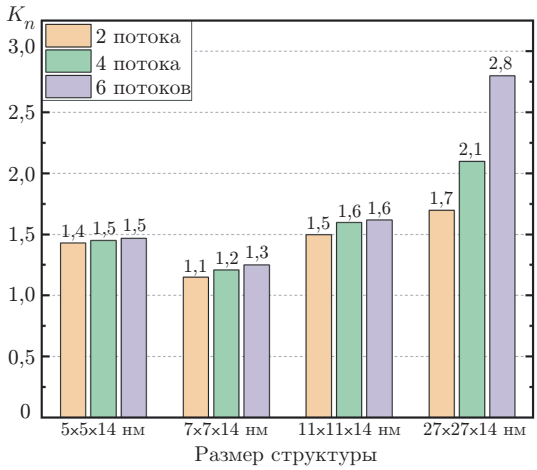

Рис. 4. Эффективность исполнения параллельного моделирования $K_n = T_1/T_n$, где K_n — коэффициент ускорения, T_n — время выполнения программы при n потоках, n — количество потоков

ожидает, пока члены группы не переведут счётчики состояния в необходимое значение. После этого лидер переходит к синхронизации на уровень выше (`BarrierLevel(level+1)`), а остальные потоки — в ожидание, пока значение локального флага `sense_local[rank]` не станет равным значению глобального флага `sense`. Инвертирование счётчиков `sense_local[rank]` и `sense` обеспечивает корректную работу многократных вызовов барьера (схема sense reversing). Начальные состояния: `sense = 1`, `sense_local[rank] = 1`. В листингах 5 и 6 приведён псевдокод предложенного алгоритма.

На рис. 3 представлены результаты для двух процессоров (128 ядер, четыре NUMA-узла, ARMv8, Linux 4.14.0-115). Наилучшие результаты демонстрирует иерархический алгоритм с группировкой процессов по NUMA-узлам (topo-numa), незначительно ему уступает алгоритм объединяющего дерева. Иерархический алгоритм с группировкой по NUMA-узлам и процессорам (topo-numa-sk) опережает на 14 % алгоритм с группировкой только по сокетам (topo-sk).

Параллельное моделирование гетероэпитаксиального роста Ge на Si методом Монте-Карло. На рис. 4 представлены результаты эффективности исполнения параллельного моделирования гетероэпитаксиального роста Ge на Si методом МК на

многоядерной системе (Intel Core i7-9700K, 32 Гб ОЗУ, операционная система Ubuntu 22.04.3 LTS, архитектура x86_64, компилятор gcc 11.4.0, версия OpenMP 4.5) с использованием предложенных решений.

Результаты экспериментов показывают, что эффективность распараллеливания увеличивается с ростом размера структуры. Это связано с тем, что доля времени на накладные расходы уменьшается относительно общего времени моделирования. Таким образом, многоранговая задача моделирования допускает решение на вычислительной системе с общей памятью, которое при формировании множества многоранговых задач производится в рамках нескольких узлов с максимально возможным количеством имеющихся потоков.

Заключение. В представленной работе предложены алгоритмы, позволяющие повысить эффективность моделирования гетероэпитаксиального роста Ge на Si методом Монте-Карло на вычислительных системах с общей памятью. Оптимизация и распараллеливание самых трудоёмких циклов выполнены с применением директив библиотеки, что позволило ускорить время моделирования гетероэпитаксии методом МК до 2,8 раз. Разработан метод барьерной синхронизации, который группирует процессы, совместно использующие ресурсы на каждом уровне иерархии памяти (разделяют кеш-память L2, L3, контроллер доступа к памяти NUMA-узла, принадлежащей одному процессору). Это позволяет локализовать синхронизацию в созданных группах и тем самым сократить время синхронизации. Разработанные методы и алгоритмы реализованы в виде программного обеспечения для многопроцессорных NUMA-узлов с общей памятью.

Финансирование. Работа выполнена в рамках государственного задания Министерства науки и высшего образования РФ (тема № FWGW-2022-0011).

СПИСОК ЛИТЕРАТУРЫ

1. **Schmidt O. G., Eberl K.** Self-assembled Ge/Si dots for faster field-effect transistors // IEEE Trans. Electron Devices. 2001. **48**, Iss. 6. P. 1175–1179.
2. **Tsybeskov L., Lockwood D. J.** Silicon-germanium nanostructures for light emitters and on-chip optical interconnects // Proceedings of the IEEE. 2009. **97**, Iss. 7. P. 1284–1303.
3. **Stangl J., Holý V., Bauer G.** Structural properties of self-organized semiconductor nanostructures // Rev. Mod. Phys. 2004. **76**, Iss. 3. 725. DOI: 10.1103/RevModPhys.76.725.
4. **Рудин С. А., Зиновьев В. А., Ненашев А. В.** Трёхмерная модель гетероэпитаксиального роста германия на кремнии // Автометрия. 2013. **49**, № 5. С. 50–56.
5. **Keating P. N.** Effect of invariance requirements on the elastic strain energy of crystals with application to the diamond structure // Phys. Rev. 1966. **145**, Iss. 2. P. 637–645.
6. **Dvurechenskii A., Smagina Z., Novikov P. et al.** Spatially arranged chains of Ge quantum dots grown on Si substrate prepatterned by ion-beam-assisted nanoimprint lithography // Phys. Stat. Solid. C. 2016. **13**, Iss. 10–12. P. 882–885.
7. **Rudin S. A., Zinovyev V. A., Smagina Zh. V. et al.** Groups of Ge nanoislands grown outside pits on pit-patterned Si substrates // Journ. Cryst. Growth. 2022. **593**. 126763. DOI: 10.1016/j.jcrysgro.2022.126763.
8. **Smagina Zh. V., Zinovyev V. A., Rudin S. A. et al.** Nucleation sites of Ge nanoislands grown on pit-patterned Si substrate prepared by electron-beam lithography // Journ. Appl. Phys. 2018. **123**, Iss. 16. 165302. DOI: 10.1063/1.5009154.
9. **Хорошевский В. Г.** Архитектура вычислительных систем: Учеб. пособие. 2-е изд., перераб. и доп. М.: Изд-во МГТУ им. Н. Э. Баумана, 1966. 520 с.
10. **Stukowski A.** Visualization and analysis of atomistic simulation data with OVITO-the Open Visualization Tool // Model. and Simul. Mater. Sci. and Eng. 2010. **18**, N 1. 015012. DOI: 10.1088/0965-0393/18/1/015012.

11. **Sayle R. A., Milner-White E. J.** RASMOL: Biomolecular graphics for all // Trends Biochem. Sci. 1995. **20**, Iss. 9. P. 374–376.
12. **Jain S., Kaleem R., Balmana M. G. et al.** Framework for Scalable Intra-Node Collective Operations using Shared Memory // Proc. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis (SC18). Dallas, USA, 11–16 Nov., 2018. P. 374–385.
13. **Yew P.-C., Tzeng N.-F., Lawrie D. H.** Distributing hot-spot addressing in large-scale multiprocessors // IEEE Trans. Comput. 1987. **C-36**, Iss. 4. P. 388–395.
14. **Mellor-Crummey J. M., Scott M. L.** Algorithms for scalable synchronization on shared-memory multiprocessors // ACM Trans. Comput. Syst. 1991. **9**, Iss. 1. P. 21–65.
15. **Hensgen D., Finkel R., Manber U.** Two algorithms for barrier synchronization // Int. Journ. Parallel Programming. 1988. **17**, Iss. 1. P. 1–17.
16. **Brooks E. D. III.** The butterfly barrier // Int. Journ. Parallel Programming. 1986. **15**, Iss. 4. P. 295–307.

Поступила в редакцию 22.05.2024
После доработки 03.06.2024
Принята к публикации 05.07.2024