УДК 004.272

ИНСТРУМЕНТАРИЙ ОПТИМИЗАЦИИ ПАРАЛЛЕЛЬНОГО МОДЕЛИРОВАНИЯ НАНОСТРУКТУР С КВАНТОВЫМИ ТОЧКАМИ*

К. В. Павский[1,2], **М. Г. Курносов**[1,2], **А. Ю. Поляков**[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

Рассмотрен разработанный авторами инструментарий оптимизации выполнения параллельных программ на мультиархитектурных распределённых вычислительных системах. Описан метод оптимизации вложения параллельных MPI-программ в вычислительные кластеры с иерархической структурой коммуникационных сетей. Для организации эффективного отказоустойчивого моделирования на распределённых вычислительных системах предложен адаптивный подход к дельта-оптимизации контрольных точек восстановления.

Ключевые слова: вложение параллельных программ, отказоустойчивость, параллельное программирование, вычислительные системы.

Введение. Разработка перспективных электронных устройств и приборов, таких как твердотельные лазеры, светодиодные матрицы, квантовый компьютер, основывается на использовании квантовых точек, т. е. искусственных атомов, свойствами которых можно управлять. Существенные результаты в этой области могут быть получены путём математического моделирования гетероэпитаксиального роста атомов на структурированных подложках. Решение данной задачи связано с обработкой больших объёмов информации о структуре, свойствах и состоянии кристаллической решётки и её элементов на высокопроизводительных распределённых вычислительных системах (ВС) [1, 2].

В архитектурном плане распределённая ВС [3, 4] — это композиция множества вычислительных узлов и коммуникационной сети. Для современных ВС характерны большемасштабность, мультиархитектурная (гибридная) организация вычислительных узлов и иерархическая структура коммуникационной сети. Например, система "Tianhe-2" (первое место в 41-й редакции списка Top500, июнь 2013 г.) состоит из 16000 мультиархитектурных вычислительных узлов. Каждый узел включает два универсальных процессора Intel Xeon Ivy Bridge и три сопроцессора Intel Xeon Phi (всего 3120000 процессорных ядер). Время передачи информации между процессорными ядрами в таких ВС зависит от их размещения в системе. Так, процессорные ядра одного узла могут взаимодействовать через его оперативную память, а ядрам разных узлов необходима менее производительная коммуникационная сеть межузловых связей.

Эффективное параллельное моделирование наноструктур с квантовыми точками на распределённых ВС требует создания системного программного обеспечения, минимизирующего накладные расходы на межмашинные обмены и реализующего режимы отказоустойчивого выполнения параллельных программ. Цель данной работы — создание инструментария (модели, алгоритмы и программное обеспечение) оптимизации отказоустойчивого выполнения параллельных программ на распределённых ВС.

*Работа выполнена при поддержке Российского фонда фундаментальных исследований (гранты № 12-07-00145, № 13-07-00160, № 12-07-00019) и Сибирского отделения РАН (интеграционный проект № 39).

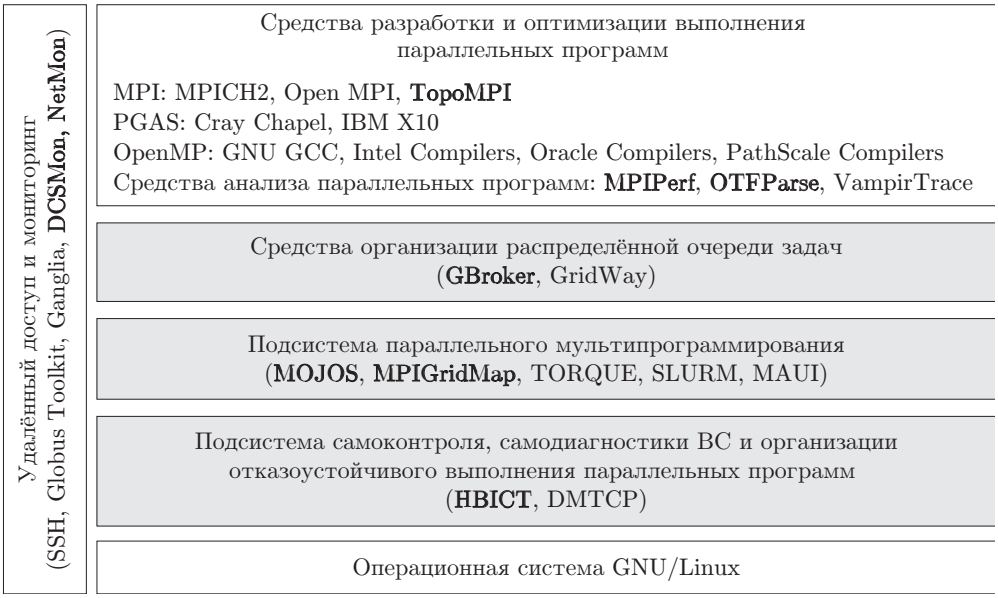

Рис. 1. Инструментарий параллельного мультипрограммирования (жирным шрифтом показаны компоненты, разработанные в ИФП СО РАН). Серым цветом обозначена подсистема параллельного мультипрограммирования

Инструментарий параллельного мультипрограммирования. Лабораторией вычислительных систем Института физики полупроводников (ИФП СО РАН) совместно с Центром параллельных вычислительных технологий Сибирского государственного университета телекоммуникаций и информатики (ЦПВТ ФГОБУ ВПО «СибГУТИ») создан инструментарий параллельного мультипрограммирования для распределённых ВС (рис. 1). Инструментарий включает: подсистему разработки и оптимизации выполнения параллельных программ; средства организации распределённой очереди задач и диспетчера пользовательских запросов; подсистему организации функционирования ВС в мультипрограммных режимах, состоящую из средств вложения параллельных программ и реализации эффективных групповых обменов между их ветвями; средства самоконтроля, самодиагностики и организации отказоустойчивого выполнения параллельных программ; средства мониторинга и организации удалённого доступа к ресурсам ВС.

Разработанные программные компоненты ориентированы на функционирование под управлением операционной системы GNU/Linux и поддерживают параллельные программы, созданные средствами MPI, Cray Chapel и IBM X10.

Оптимизация вложения параллельных программ в структуру ВС. Одной из важных проблем организации функционирования распределённых ВС является оптимизация вложения в них параллельных программ (task mapping, task allocation, task assignment). Под оптимальным вложением понимается такое распределение ветвей параллельной программы по процессорным ядрам ВС, при котором достигается минимум накладных расходов на межмашинные обмены информацией.

Существующие библиотеки стандарта MPI (MPICH2, Open MPI и др.) реализуют алгоритмы вложения параллельных программ с учётом только двух уровней коммуникационной сети — сети межузловых связей и общей памяти вычислительных узлов [5]. В работах [6, 7] предложены алгоритмы вложения программ в системы, имеющие фиксированную тороидальную структуру коммуникационной среды. Применение существующих методов вложения не обеспечивает предельной эффективности использования ресурсов ВС, так как не они учитывают все иерархические уровни коммуникационной среды.

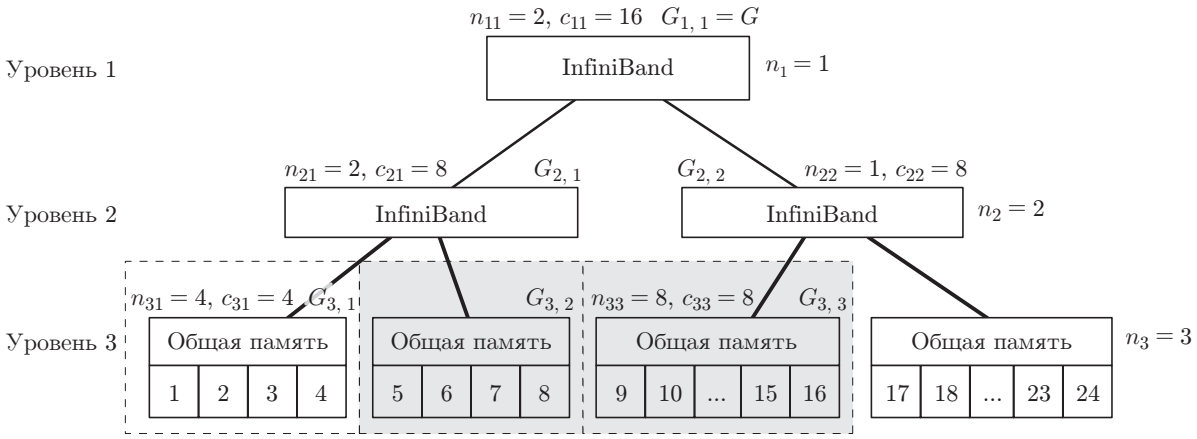

Рис. 2. Пример подсистемы из $N = 16$ процессорных ядер для решения параллельной задачи

Коммуникационная сеть ВС может быть представлена в виде дерева, содержащего L уровней. Для параллельной программы выделяется подсистема из N процессорных ядер (на рис. 2 отмечено серым цветом). Обозначим n_l число элементов на уровне $l \in \{1, 2, \ldots, L\}$; n_{lk} — количество прямых дочерних узлов элемента $k \in \{1, 2, \ldots, n_l\}$, находящегося на уровне l; c_{lk} — количество процессорных ядер, принадлежащих потомкам этого элемента.

Параллельная программа, созданная в модели передачи сообщений, представлена информационным графом $G = (V, E)$, где $V = \{1, 2, \ldots, N\}$ — множество ветвей параллельной программы, а $E \subseteq V \times V$ — множество информационно-логических связей между ветвями. Обозначим через d_{ij} вес ребра $(i, j) \in E$, характеризующий интенсивность обменов данными между ветвями i и j при выполнении программы.

Вложение параллельной программы в ВС задаётся значениями переменных $x_{ij} \in \in \{0, 1\}$: $x_{ij} = 1$, если ветвь $i \in V$ назначена на процессорное ядро $j \in \{1, 2, \ldots, N\}$, иначе $x_{ij} = 0$.

В качестве критерия оценки эффективности вложения используется время T выполнения информационных обменов. Оно определяется максимальным из времён выполнения обменов ветвями программы. Требуется построить вложение X, доставляющее минимум критерия T:

$$T(X) = \max_{i \in V} \left\{ \sum_{j=1}^{N} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip} x_{jq} t(i, j, p, q) \right\} \to \min_{(x_{ij})} \tag{1}$$

при ограничениях

$$\sum_{j=1}^{N} x_{ij} = 1, \quad i = 1, 2, \ldots, N, \tag{2}$$

$$\sum_{i=1}^{N} x_{ij} = 1, \quad j = 1, 2, \ldots, N. \tag{3}$$

Ограничения (2) гарантируют назначение каждой ветви параллельной программы на единственное ядро. Ограничения (3) обеспечивают назначение на процессорное ядро не более одной ветви. Задача (1)–(3) относится к дискретной оптимизации и является трудноразрешимой. В данной работе предложен метод HierarchicalMap её приближённого решения. Метод основан на разбиении информационного графа задачи на подмножества интенсивно обменивающихся параллельных ветвей и дальнейшем распределении их по

процессорным ядрам, связанным наиболее производительными каналами связи. Разбиение выполняется многократно для каждого уровня иерархии коммуникационной среды.

На вход метода подаётся взвешенный граф $G = (V, E)$ параллельной программы. В начале метода полагаем, что $G_{11} = G$, $l = 1$, $k = 1$.

Шаг 1. Если текущий уровень $l = L$, выполнение алгоритма завершается. В противном случае граф G_{lk} разбивается на n_{lk} частей $G_{l+1,1}, G_{l+1,2}, \ldots, G_{l+1,n_{lk}}$ по $c_{l+1,1}, c_{l+1,2}, \ldots$ $\ldots, c_{l+1,n_{lk}}$ вершин в каждой.

Шаг 2. Для каждого из подграфов $G_{l+1,1}, G_{l+1,2}, \ldots, G_{l+1,n_{lk}}$ выполняется процедура разбиения в соответствии с шагом 1.

Первый шаг алгоритма реализуется с помощью многоуровневых методов разбиения графов на непересекающиеся подмножества. Цель разбиения — минимизировать суммарный вес рёбер, инцидентных различным подмножествам разбиения. Для решения этой задачи существуют эффективные эвристические алгоритмы и пакеты программ (METIS, Scotch, gpart). Процедура разбиения графов останавливается, когда каждый подграф G_{lk} задачи может быть реализован одним вычислительным узлом (когда количество вершин в подграфе не превышает числа процессорных ядер узла).

Экспериментальное исследование предложенного метода проводилось на мультикластерной ВС ЦПВТ «СибГУТИ» и в Лаборатории вычислительных систем ИФП СО РАН. Использовались тестовые MPI-программы POP, Sweep3D, Graph 500 и тесты LU, SP, MG, BT из пакета NAS Parallel Benchmarks (NPB). Перечисленные тестовые пакеты реализуют основные схемы информационных обменов (one-to-all broadcast, all-to-all broadcast и др.), характерные для параллельных программ моделирования задач наноструктур с квантовыми точками, в частности для параллельных методов молекулярной динамики. Разбиение графов на k подмножеств (шаг 1) выполнялось с помощью библиотек Scotch, METIS и gpart. Заметим, что библиотека Scotch используется в пакете hwloc при вложении параллельных MPI-программ.

Время выполнения MPI-программ при вложении их алгоритмом, учитывающим все уровни коммуникационной среды ВС, на некоторых задачах от 1,1 до 5 раз ниже по сравнению с линейным вложением (рис. 3), которое применяется по умолчанию библиотеками MPI. Установлено, что предложенный метод эффективен для параллельных задач, имеющих разреженные информационные графы с преобладанием дифференцированных MPI-обменов (point-to-point). Время работы метода на одном процессоре не превышает 1 с для задач с $N \leq 65536$ (на процессоре Intel Xeon E5420).

Сравнение пакетов METIS, Scotch, gpart разбиения графов показало, что на большинстве тестовых программ все библиотеки дают сопоставимые результаты. Применение

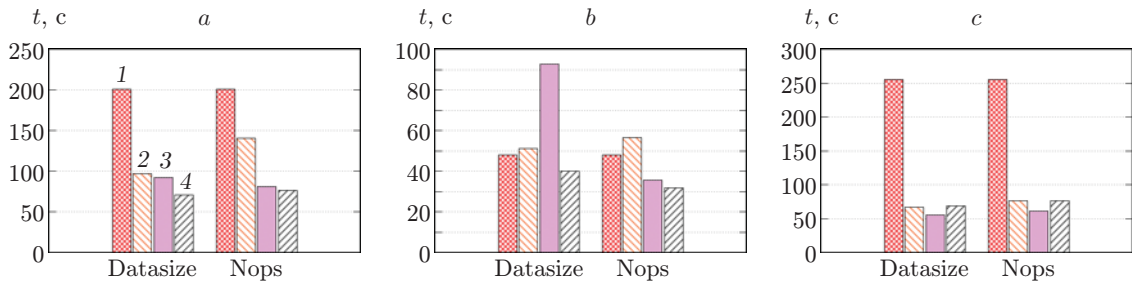

Рис. 3. Сравнение алгоритмов вложения параллельных программ: *a* — задача POP, $N = 120$; *b* — задача Sweep3D, $N = 120$; *c* — задача NPB MG, $N = 64$. Обозначения на диаграммах: *1* — линейное вложение, *2* — L1Map, *3* — L2Map, *4* — L12Map; datasize — вес ребра отражает суммарный размер сообщений, переданных между парой ветвей; nops — суммарное число сообщений, переданных по ребру

библиотеки Scotch в некоторых случаях (POP) обеспечивает незначительное сокращение времени выполнения программы. Разработанный метод реализован в пакете MPIGridMap.

Отказоустойчивое выполнение параллельных программ. Распределённые ВС являются сложными стохастическими объектами, отказы узлов в которых происходят в случайные моменты времени [8]. Для обеспечения способности ВС продолжать решение задач в условиях наличия отказов применяется периодическое сохранение состояний соответствующих параллельных программ в контрольных точках (КТ). В случае отказа в ВС КТ позволяют возобновить выполнение программ на исправных процессорных ядрах [9].

Рассмотрим программу, состоящую из N параллельных ветвей. Обозначим через $K = \{K_1, K_2, \ldots, K_T\}$ набор КТ, созданных в процессе выполнения программы в моменты времени $t = 1, 2, \ldots, T$. Контрольная точка K_t содержит состояние программы в момент времени t и физически представлена одним или более файлами. Под объёмом $S(K_t)$ ($[S(x)]$ = байт) КТ K_t будем понимать суммарный размер всех файлов, входящих в неё. Для записи K_t в хранилище данных необходимо время $W(S(K_t))$, где $W(x)$ — монотонно возрастающая функция (часто близкая к линейной). При записи КТ в хранилище данных вычислительный процесс приостанавливается, поэтому такие периоды выполнения программы относятся к накладным расходам (E).

Заметим, что КТ набора K описывают состояние одной и той же программы в различные моменты времени. Поэтому для программ, частично модифицирующих содержимое памяти, набор K будет содержать дублирующуюся информацию. Эта особенность позволяет производить дельта-оптимизацию (дельта-сжатие) КТ набора K [10]. Строится новый набор $C(b) = \{K_1, \Delta(K_2, K_{b_1}), \Delta(K_3, K_{b_2}), \ldots, \Delta(K_T, K_{b_{T-1}})\}$, где $\Delta(x, y)$ — функция дельта-сжатия КТ x относительно КТ y, а $b = \{b_i\colon 1 \le b_i \le i,\ i \in \{1, 2, \ldots, T-1\}\}$ — вектор индексов КТ, относительно которых производится сжатие. Требуется найти вектор b:

$$E(C(b)) = \sum_{t=1}^{T} W(S(C_t(b))) \to \min_b, \tag{4}$$

$$\sum_{i=1}^{T-1} b_i \to \min, \tag{5}$$

где соотношение (5) минимизирует количество изменений базовой КТ.

Инкрементное дельта-сжатие является основным видом дельта-оптимизации КТ, при этом $b = \{1, 2, \ldots, T-1\}$. Его существенный недостаток — необходимость наличия всех КТ из K для формирования K_T: $K_T = \Delta^{-1}(C_T, \Delta^{-1}(C_{T-1}, \Delta^{-1}(\ldots, \Delta^{-1}(C_2, C_1))\ldots))$.

Авторами предложен эвристический алгоритм дельта-оптимизации (Adaptive Delta-Compression Algorithm (ADCA)) набора K. Алгоритм осуществляет пошаговое построение вектора b, обеспечивающего (суб)оптимальное значение функции $E(C(b))$.

Шаг 1. Изначально полагаем, что $b_1 = 1$, $t = 2$, $\varepsilon = 0{,}2$.

Шаг 2. $C_t = \Delta(K_t, K_{b_{t-1}})$, $C'_t = \Delta(K_t, K_{t-1})$.

Шаг 3. Если $(S(C_t) - S(C'_t))/S(C_t) > \varepsilon$, то $b_t = t - 1$; иначе $b_t = b_{t-1}$. Перейти на шаг 2.

Представленный алгоритм реализован в инструментарии (HBICT) дельта-оптимизации КТ. В качестве $\Delta(x, y)$ применялся алгоритм FsCH [10], основанный на хешировании. Также выполнена интеграция HBICT с системой создания КТ DMTCP [9].

Экспериментальные исследования оптимизации КТ в программах моделирования гетероэпитаксиального роста квантовых точек [1, 2], разработанных в ИФП СО РАН, показали, что шаблон использования памяти в таких программах позволяет осуществлять дельта-сжатие относительно первой КТ ($b = \{b_i = 1,\ i \in \{1, \ldots, T-1\}\}$, $K_T = \Delta^{-1}(C_T, C_1)$). Аналогичные результаты были получены для программ NPB и LAMMPS.

Также в ходе экспериментов показана применимость выбранного алгоритма реализации функции $\Delta(x, y)$ путём сравнения с более точным, но менее гибким страничным подходом [11].

Заключение. Предложенный в данной работе инструментарий оптимизации отказоустойчивого выполнения параллельных программ — один из необходимых компонентов обеспечения живучести, надёжности и эффективности использования ресурсов распределённых ВС. Его применение позволяет сократить время параллельного отказоустойчивого моделирования наноструктур с квантовыми точками. В частности, предложенный метод вложения параллельных программ в распределённые ВС с иерархической структурой позволяет сократить время выполнения параллельных программ, имеющих разреженные информационные графы с преобладанием дифференцированных (point-to-point) обменов. Эффективность разработанных алгоритмов дельта-оптимизации контрольных точек параллельных программ подтверждена экспериментально.

СПИСОК ЛИТЕРАТУРЫ

1. **Novikov P., Smagina Zh., Vlasov D. et al.** Space arrangement of Ge nanoislands formed by growth of Ge on pit-patterned Si substrates // Journ. Crystal Growth. 2011. **323**, Is. 1. P. 198–200.
2. **Nenashev A. V., Dvurechenskii A. V.** Strain distribution in quantum dot of arbitrary polyhedral shape: Analytical solution // Journ. Appl. Phys. 2010. **107**, N 6. 064322.
3. **Хорошевский В. Г.** Распределённые вычислительные системы с программируемой структурой // Вестн. СибГУТИ. 2010. № 2(10). С. 3–41.
4. **Шидловский С. В., Сырямкин В. И., Шидловский В. С.** Перестраиваемые вычислительные среды в многосвязных системах автоматического управления // Телекоммуникации. 2010. № 10. С. 28–32.
5. **Broquedis F., Clet-Ortega J., Moreaud S. et al.** hwloc: A generic framework for managing hardware affinities in HPC applications // 18th Intern. Conf. on Parallel, Distributed and Network-Based Processing. Pisa, Italy, 17–19 Feb., 2010. P. 180–186.
6. **Hoefler T., Snir M.** Generic topology mapping strategies for large-scale parallel architectures // Proc. of the ACM Intern. Conf. on Supercomputing. Tucson, USA, June, 2011. P. 75–85.
7. **Balaji P., Gupta R., Vishnu A., Beckman P.** Mapping communication layouts to network hardware characteristics on massive-scale blue gene systems // Journ. Comput. Sci. Res. Dev. 2011. **26**, N 3–4. P. 247–256.
8. **Schroeder B., Gibson G. A.** A large-scale study of failures in high-performance computing systems // Proc. of the Intern. Conf. on Dependable Systems and Networks. Philadelphia, USA, 25–28 June, 2006. P. 249–258.
9. **Ansel J., Arya K., Cooperman G.** DMTCP: Transparent checkpointing for cluster computations and the desktop // Proc. of the IEEE Intern. Parallel and Distributed Processing Symposium. IEEE Press, 2009. P. 1–12.
10. **Kiswany S. A., Ripeanu M., Vazhkudai S. S., Gharaibeh A.** stdchk: A checkpoint storage system for desktop grid computing // Proc. of 28th Intern. Conf. on Distributed Computing Systems. Washington, USA: IEEE Computer Society, 2008. P. 613–624.
11. **Wang C., Mueller F., Engelmann C., Scott S. L.** Hybrid checkpointing for MPI jobs in HPC environments // Proc. of the 16th IEEE Intern. Conf. on Parallel and Distributed Systems. Washington, USA: IEEE, 2010. P. 524–533.

Поступила в редакцию 2 июля 2013 г.