УДК 004.05:519.245

НОВЫЙ ПОДХОД ДЛЯ ОЦЕНКИ ТРУДОЕМКОСТИ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

***И.Б. Мееров*[1], *А.В. Русаков*[2]**

[1]Нижегородский госуниверситет им. Н.И. Лобачевского
[2]Швейцарский федеральный технологический институт, Цюрих

meerov@vmk.unn.ru

Поступила в редакцию 10.09.2012

Предлагается новый подход для оценки трудоемкости создания параллельных программ с использованием разных средств для параллельного программирования в системах с общей памятью. Вводится система метрик, методика их вычисления и применения. Предлагаемый подход иллюстрируется на примере одной из задач финансовой математики.

Ключевые слова: параллельное программирование, системы с общей памятью, трудоемкость разработки, средства для параллельного программирования.

Введение

Развитие и распространение параллельных вычислительных систем требует новых технологий, моделей, методов и программных средств системного уровня, ориентированных на их эффективное использование. Развиваются технологии (OpenMP, MPI, Cilk Plus и др.), фреймворки (OpenCL, CUDA и др.), библиотеки (TBB, ArBB, CCR и др.), языки (Co-Array Fortran, Erlang, APL, Sisal и др.) для параллельного программирования [1–3]. Многообразие указанных средств во многом связано с наличием нерешенных проблем, как с точки зрения эффективности использования возможностей аппаратуры, так и с точки зрения трудоемкости освоения и использования новых средств, ориентированных на создание параллельных программ.

Проблемы оптимизации параллельных программ с использованием разных технологий их разработки [4, 5] являются актуальными и активно освещаются в научной литературе. В то же время вопрос трудоемкости разработки программ с использованием этих технологий является мало изученным. Одними из немногих примеров могут послужить сравнительные анализы, подготовленные в Калифорнийском университете (UCSB) [6] и признанном центре разработок по программной инженерии – Высшей технической школе Цюриха (ETH) [7]. Оставляя за рамками вопрос о сложности создания новых параллельных алгоритмов, заметим, что даже в случае адаптации для параллельного выполнения уже существующих программ от прикладных программистов часто требуются серьезные навыки разработки, отладки и оптимизации, что существенно снижает производительность труда и зачастую нивелирует гипотетический выигрыш от перехода на новую программно-аппаратную платформу.

В статье предлагается новый подход для анализа трудоемкости создания параллельных программ для систем с общей памятью. Предлагаются набор метрик и методика их вычисления. Приводится пример использования указанного подхода для сравнения трудоемкости разработки программ с использованием TBB, Cilk Plus и OpenCL. Тестовая задача – оценивание опционов бермудского типа. Наряду с анализом трудоемкости разработки приводятся результаты анализа производительности (без каких-либо претензий на общность; основная цель – показать достаточное качество оптимизации кода при использовании разных технологий).

Указанный подход подготовлен к последующей *апробации* при чтении сотрудниками факультета ВМК ННГУ учебных курсов по средствам параллельного программирования [8–10]. Именно с этих позиций выполнено его описание.

1. Подход для оценки трудоемкости разработки

Задача выбора из разнообразия доступных средств и технологий для параллельного программирования является достаточно актуальной. Если требования к производительности параллельных программ могут быть выражены в таких показателях, как, например, время работы, ускорение и эффективность, то для анализа

удобства разработки и отладки сложно выделить универсальный механизм оценки.

Общим подходом к оценке трудоемкости разработки программ с использованием различных технологий параллельного программирования является анализ набора количественных данных, характеризующих код и сам процесс программирования. Подобные показатели называются метриками программного кода. Современная литература, посвященная вопросам управления качеством, содержит большое количество различных метрик, использующихся для оценки качества и сложности программного обеспечения и программного кода ([11, 12]). Авторами данной работы предлагается набор метрик, основанный на широко распространенных метриках, описанных в литературе, а также вытекающий из опыта написания параллельных программ на кафедре математического обеспечения ЭВМ факультета ВМК ННГУ.

Идея использования метрик для сравнения различных технологий основывается на сопоставлении полученных данных для каждого средства программирования с целью нахождения закономерностей на примере схожих задач. Для качественного сравнения важно определить и обеспечить общие правила подсчета и фиксирования данных.

1.1. Целевые группы. В рамках проведения исследования о сравнении различных технологий программирования с помощью анализа собранных метрик выделяются три определяющие роли: лектор, исследователь и разработчик. Задача лектора – представить теоретический материал о технологиях и оценить уровень полученных разработчиком знаний сразу после прочитанного курса. Цель исследователя – предоставить разработчику задачи для решения с использованием различных технологий программирования, а также получить и проанализировать результаты собранных разработчиками метрик. Роль разработчика состоит в подсчете интересующих метрик при решении своей прикладной задачи.

1.2. Лектор. Роль лектора заключается в прочтении курса по основам каждой из рассматриваемых технологий параллельного программирования. Для этого лектор должен обладать достаточной квалификацией в использовании этих технологий. После прочтения курса лектор оценивает уровень освоения технологий слушателями. Для этого используется тестирование на основе материала прочитанных лекций.

Тестирование

С целью получения объективных результатов необходимо обеспечить общие правила процедуры тестирования разработчиков для каждой технологии.

Лекторами подготавливается перечень вопросов для проведения тестирования. Каждый вопрос отвечает следующим требованиям:

– вопрос основывается на материале, освещенном в лекциях;

– вопрос относится к одной из введенных групп сложности: простой, средний или сложный;

– из предложенных вариантов ответа верным является ровно один вариант.

Тест должен содержать равное число вопросов для каждой группы сложности (авторы использовали 15 вопросов в сумме).

Лектор должен обеспечить следующие правила проведения тестирования:

– тестирование проводится сразу после прочтения последней лекции из курса;

– слушатели работают самостоятельно;

– на решение теста отводится время из расчета не более двух минут на вопрос.

После проведения процедуры тестирования лектор собирает и обрабатывает полученные данные для предоставления результатов исследователю.

Проверка результатов тестирования

Определение результатов тестирования в ходе проверки тестовых работ проводится по балльной системе. За каждый верный ответ на вопрос из группы «легкие» начисляется 1 балл, из группы «средние» – 2 балла, из группы «сложные» – 3 балла.

Для дальнейшего детального анализа лектор предоставляет результаты тестирования исследователю в виде таблицы – карты ответов. Таблица 1 содержит для каждого опрошенного отметку о результате (верно – 1/неверно – 0) для каждого вопроса. В строке «Итого» содержится

Таблица 1

Форма предоставления результатов тестирования

Фамилия	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Результат
Иванов	1	1	1	1	1	1	0	1	1	1	1	0	0	1	0	19
Петров	1	1	0	1	1	1	0	1	1	1	1	1	1	0	0	21
…																
Итого	2	2	1	2	2	2	0	2	2	2	2	1	1	1	0	

суммарное количество правильных ответов среди всех опрошенных для каждого из вопросов.

1.3. Исследователь. Исследователю в данной методике выделяется роль организатора процесса сбора метрик, а также аналитика в работе с полученными результатами. Именно перед исследователем ставится задача сравнения трудоемкости разработки программ с использованием различных средств параллельного программирования. Он должен ознакомить лекторов и разработчиков с данной методикой, разъяснить каждому его роль, обеспечить вспомогательными материалами и поставить задачу по сбору рассматриваемых метрик.

Исследователь обязан проконсультировать лекторов по вопросам составления заданий для тестирования и контролировать своевременность сбора метрик разработчиками, консультируя их по возникающим вопросам. После проведения тестирования и завершения процесса сбора метрик исследователь должен получить данные от лекторов и разработчиков для последующей обработки, анализа и обобщения результатов.

Выбор задач и средств для параллельного программирования

При выборе тестовых задач, решение которых будет реализовано разработчиками с использованием различных средств параллельного программирования, достижения объективных результатов сравнения необходимо обратить внимание на следующие моменты. Решаемые задачи должны быть из одной прикладной области или основываться на схожей математической базе. Задачи не должны существенно отличаться по уровню сложности. Разработчики должны быть знакомы с тестовыми задачами и их особенностями.

Исследователь выбирает для сравнения интересующие его технологии параллельного программирования для систем с общей памятью, исходя из возможностей лекторов предоставить теоретический материал по их основам, возможностей разработчиков освоить данные технологии и успешно применить к поставленным задачам, а также перспективности самих технологий и оригинальности подходов к распараллеливанию, реализованных в них.

Подход к анализу собранных метрик

Исследователем анализируются результаты, полученные лекторами в ходе проверки тестовых работ, а также метрики, собранные разработчиками в процессе решения поставленных перед ними задач. Идея состоит в составе метрик и наглядном сравнении их значений на сводных диаграммах с последующим описанием обнаруженных тенденций с их обоснованием при помощи как теоретических знаний о технологиях, так и комментариев разработчиков.

Для получения сравнительных диаграмм исследователь заполняет специально подготовленный шаблон таблицы собранных результатов (*Исследователь – Таблица для анализа метрик.xlsx*[1]). Исследователь вносит туда усредненные данные для всех разработчиков. В случае если разработчики решали разные задачи, требуется предварительная нормировка значений метрик (за 1.0 принимаются показатели метрик одной из рассмотренных технологий, метрики остальных технологий рассчитываются пропорционально ей).

Цель исследователя как аналитика – выявить зависимости в показателях определенных групп метрик. После определения причин выявленных зависимостей (при условии репрезентативности выбранного набора задач и разработчиков) делается вывод о применимости той или иной технологии к рассмотренной области задач с точки зрения трудоемкости разработки параллельных приложений.

1.4. Разработчик. Разработчиком в данном исследовании называется прикладной программист, непосредственно занимающийся написанием кода программы с использованием различных средств для параллельного программирования. Его роль в исследовании – отслеживание изменений интересующих метрик (таблица метрик) и своевременное фиксирование полученных данных о метриках в специально подготовленной таблице результатов (*Разработчик – Таблица для сбора метрик.xlsx*[2]).

Перед началом процесса программирования разработчик должен зафиксировать множество наблюдаемых метрик, подготовить таблицу результатов для внесения изменений, ознакомиться с инструкцией по сбору метрик, а также настроить и освоить необходимый инструментарий.

Инструкция по сбору метрик

Для качественного учета метрик важно обеспечить единообразный процесс их фиксирования. Для этого разработчик должен внимательно изучить таблицу собираемых метрик и в процессе решения задачи постоянно следить за их изменением. Другим определяющим моментом для достижения репрезентативности собранных данных является своевременность фиксирования метрик. Изменения должны вноситься в таблицу результатов сразу при возникнове-

Таблица 2

Метрики для сравнения технологий параллельного программирования

id	Метрика	Описание	Группа	Время сбора	Способ сбора
101	Время обучения технологии программирования	Время, затраченное на *теоретическое* изучение технологии программирования (лекции, самостоятельное изучение материалов)	Подготовка	Перед началом процесса программирования	Вручную
102	Общее количество ошибок в контрольном тесте	Число ошибок в итоговом тесте по окончании курса лекций по данной технологии программирования	Подготовка	Перед началом процесса программирования	Вручную
103	Время разворачивания и настройки среды программирования	Время, затраченное на подготовку рабочего места разработчика к началу практического использования той или иной технологии программирования (установка и настройка сред программирования, фреймворков и других инструментов)	Подготовка	Перед началом процесса программирования	Вручную
201	Количество структурных изменений программы	Общее число изменений структуры программы в процессе рефакторинга. (Объединение или разделение кода на различные функции, изменение используемых структур данных и т.д.)	Надежность	В процессе программирования	Вручную
202	Количество ошибок, выявленных в ходе просмотра кода	Число допущенных ошибок, которые были замечены путем просмотра кода *без использования средств тестирования и отладки*	Надежность	В процессе программирования	Вручную
203	Количество ошибок, выявленных при тестировании программы	Число допущенных ошибок, которые были замечены в процессе тестирования программы и при помощи средств отладки	Надежность	В процессе программирования	Вручную
204	Общее количество ошибок, выявленных при разработке	Общее число ошибок, допущенных и исправленных в исходном коде во всем процессе разработки	Надежность	В процессе программирования	Вручную
205	Среднее количество ошибок, выявленных при разработке, на строку кода	Отношение общего числа ошибок к числу строк исходного кода программы	Надежность	В процессе программирования	Автоматически в таблице
301	Количество сборок кода	Число компиляций исходного кода в процессе разработки	Процесс	В процессе программирования	Вручную
302	Среднее время сборки	Усредненное время сборки кода (в том числе на начальной стадии разработки, на промежуточных и по окончании)	Процесс	В процессе программирования	Вручную
303	Общее время разработки	Общее время, затраченное на программирование, при решении задачи	Процесс	После завершения процесса программирования	Вручную
304	Общие трудозатраты	Общее время разработки с учетом количества программистов, вовлеченных в решение задачи	Процесс	После завершения процесса программирования	Вручную

Таблица 2 (продолжение)

id	Метрика	Описание	Группа	Время сбора	Способ сбора
401	Количество строк исходного кода (с комментариями)	Число строк исходного кода, *включая* комментарии	Размер исходного кода	После завершения процесса программирования	Инструмент
402	Количество строк исходного кода (без комментариев)	Число строк исходного кода, *исключая* комментарии	Размер исходного кода	После завершения процесса программирования	Вручную
403	Количество функций	Число функций в исходном коде программы	Размер исходного кода	После завершения процесса программирования	Инструмент
404	Среднее количество функций на строку кода	Отношение числа общего числа функций к общему числу строк (исключая комментарии)	Размер исходного кода	После завершения процесса программирования	Автоматически в таблице
405	Количество операторов	Общее количество операторов языка программирования в исходном коде программы (например, if, else, for, +, – и т.д.)	Размер исходного кода	После завершения процесса программирования	Инструмент
406	Среднее количество операторов на строку кода	Отношение общего числа операторов к общему числу строк (исключая комментарии)	Размер исходного кода	После завершения процесса программирования	Автоматически в таблице
407	Количество операндов	Общее число операндов в исходном коде программы (например, константы, переменные и т.д.)	Размер исходного кода	После завершения процесса программирования	Инструмент
408	Среднее количество операндов на строку кода	Отношение общего числа операндов к общему числу строк (исключая комментарии)	Размер исходного кода	После завершения процесса программирования	Автоматически в таблице
409	Количество условных операторов (метрика Джилба)	Общее количество условных операторов языка программирования в исходном коде (if, else, switch и т.д.)	Размер исходного кода	После завершения процесса программирования	Вручную
410	Среднее количество условных операторов на строку кода (метрика Джилба)	Отношение общего количества условных операторов к общему числу строк (исключая комментарии)	Размер исходного кода	После завершения процесса программирования	Автоматически в таблице
501	Цикломатическая сложность по Мак-Кейбу	*Управляющим графом G* для программы называется ориентированный граф, содержащий лишь один вход и один выход; при этом вершины графа соотносят с теми участками кода программы, в которых имеются лишь последовательные вычисления и отсутствуют операторы ветвления и цикла, а дуги соотносят с переходами от блока к блоку и ветвями выполнения программы. Условие при построении данного графа: каждая вершина достижима из начальной и конечная вершина достижима из любой другой вершины. В этом случае цикломатической сложностью по Мак-Кейбу называется величина $V(G) = e - n + 2$, где e – количество дуг, n – количество вершин управляющего графа	Сложность потока управления	После завершения процесса программирования	Вручную

Таблица 2 (продолжение)

id	Метрика	Описание	Группа	Время сбора	Способ сбора
601	Количество классов	Количество классов в исходном коде программы	Объектно-ориентированные	После завершения процесса программирования	Инструмент
602	Насыщенность классов	Общее среднее число конструкторов и методов класса (не определенных в его родителях)	Объектно-ориентированные	После завершения процесса программирования	Вручную
603	Глубина наследования	Глубина дерева наследования (наибольший путь по иерархии классов к данному классу от класса-предка)	Объектно-ориентированные	После завершения процесса программирования	Вручную
604	Связность классов	Связность класса – количество классов, с которыми связан каждый конкретный класс. Общая связность равна отношению суммы связностей классов к общему числу классов	Объектно-ориентированные	После завершения процесса программирования	Вручную
605	Количество потомков	Количество непосредственных потомков среди всех классов в исходном коде программы	Объектно-ориентированные	После завершения процесса программирования	Вручную

нии того или иного характерного события (например, найдена очередная ошибка в программе или произведена компиляция).

Кроме того, важно учитывать, что метрики стоит собирать на соответствующих этапах решения задачи:

- *Метрики, подсчитываемые непосредственно перед началом решения задачи* (перед началом процесса программирования). К этой группе относятся метрики, связанные с обучением технологии программирования, а также с подготовительными операциями.
- *Метрики, непрерывно подсчитываемые в процессе программирования.* Данная группа метрик требует от программистов внимания и постоянного обновления данных об исследуемых метриках; данные обновляют таблицу результатов.
- *Метрики, подсчитываемые по окончании решения задачи* (после завершения процесса программирования). К этой группе относятся метрики, охватывающие весь процесс программирования в целом и полученный в результате программный код.

Для каждой конкретной метрики информация о моменте сбора указана в таблице метрик в столбце «*Время сбора*».

Общий алгоритм наблюдения за метриками можно описать следующим образом:

- определить множество метрик, относящееся к данному этапу решения задачи;
- в случае изменения какой-либо метрики из данного множества зафиксировать новое значение в таблице результатов;
- если возможно изменение какой-либо метрики из данного множества, повторить алгоритм, в противном случае – повторить алгоритм на следующем этапе решения задачи.

Таблица метрик

В таблице 2 приведён предлагаемый список метрик с их кратким описанием.

Столбец «*Группа*» отражает понятия, которые характеризует конкретная метрика.

Столбец «*Время сбора*» показывает, на каком этапе решения задачи следует вычислять ту или иную метрику и заносить ее в таблицу результатов.

Столбец «*Способ сбора*» отражает возможность подсчета некоторых метрик при помощи автоматизированных средств, описанных в разделе *Инструментарий*.

Инструментарий

Для подсчета части наблюдаемых метрик требуется лишь своевременно вносить данные в таблицу результатов. Однако выделяется множество метрик, использующих временны*е* показатели для подсчета. Поэтому в процессе исследования необходимо использовать средства измерения времени. Для вычисления отдельных метрик (например, цикломатической сложности по Мак-Кейбу) требуется дополнительный анализ написанной программы, построение графа и

вычисление некоторых его атрибутов. Использование автоматизированных средств для подсчета показателей исходного кода значительно упрощает вычисление соответствующих метрик (например, подсчет количества строк кода). Так, вычислять некоторые интересующие метрики из таблицы можно, к примеру, при помощи свободно распространяемой программы *SourceMonitor*[3].

Инструкция по заполнению таблицы результатов

Таблица результатов для заполнения разработчиком расположена в файле *Разработчик – Таблица для сбора метрик.xlsx*. Перед началом решения задачи с использованием определенной технологии программирования разработчику необходимо внести данные о себе и о задаче на листе «Данные о заполнителе» (рис. 1).

На листе «*Таблица собранных метрик*» (рис. 2) содержится список метрик для наблюдения и фиксирования. Этот список короче, чем приведенный в данном документе, поскольку удельные метрики (к примеру, количество ошибок на строку кода) вычисляются позже автоматически. В таблице отображены единицы измерения, в которых следует вносить значения собранных метрик. Графой для заполнения в этой таблице является графа «*Значение*».

После каждого сеанса внесения или обновления значений метрик следует сохранять файл результатов. Для различных технологий программирования следует использовать отдельные файлы результирующих таблиц. Перед сдачей таблицы результатов разработчиком исследователю следует проверить, что все значения наблюдавшихся метрик внесены в файл, корректны и приведены в требуемых единицах измерения.

2. Решение модельной задачи

Апробация предложенной методики, выполненная авторами статьи, заключается в сравнении технологий параллельного программирования для систем с общей памятью – OpenMP, Intel® Cilk Plus, Intel® TBB и OpenCL – на тестовой задаче. Сравнение выбранных технологий параллельного программирования производится на примере известной задачи финансовой математики – нахождение справедливой цены опциона бермудского типа. Модель финансового рынка в данной задаче описывается с использованием систем стохастических дифференциальных уравнений [13]. Для решения этой задачи был реализован основанный на методе Монте-Карло оптимизированный нерекурсивный алгоритм имитационных деревьев Broadie–Glasserman Random Trees, описанный в [14] и изна-

	A	B
1	**Фамилия**	Русаков
2	**Имя**	Андрей
3	**Отчество**	Валентинович
4		
5	**Курс**	6
6		
7	**Решаемая задача**	Определение справедливой цены опциона бермудского типа. Алгоритм BG97
8	**Используемая технология параллельного программирования**	OpenMP

Данные о заполнителе | Таблица собранных метрик

Рис. 1. Ввод данных о заполнителе

	A	B	C	D	E	F
1	Метрики					
2	**id**	**Группа**	**Название**	**Единицы измерения**	**Значение**	
3	101	*Подготовка*	Время обучения технологии программирования	Минуты	120	
4	102	*Подготовка*	Общее количество ошибок в контрольном тесте	Шт.	2	
5	103	*Подготовка*	Время разворачивания и настройки среды программирования	Минуты	20	
6	201	*Надежность*	Количество структурных изменений программы	Шт.	4	
7	202	*Надежность*	Количество ошибок, выявленных в ходе просмотра кода	Шт.	5	
8	203	*Надежность*	Количество ошибок, выявленных при тестировании программы	Шт.	7	

Данные о заполнителе | Таблица собранных метрик

Рис. 2. Ввод значений для собранных метрик

чально разработанный авторами [15]. Реализация данного алгоритма является распространенным тестом производительности среди финансовых прикладных задач и хорошо изучена авторами данной работы.

Анализ производительности

В этом разделе приведены результаты вычислительных экспериментов, которые были получены с использованием следующего программного и аппаратного обеспечения:

- Intel Xeon E5520 Gainestown (8 cores, 1.6 GHz), Mem 16 GB;
- 4x Intel(R) Xeon(R) E7-4860 (10 cores, 2.27 GHz), Mem 64 GB.
- Intel Composer-XE 2011.0.084.

Время работы программы на многоядерных системах для каждой технологии параллельного программирования приведено на рис. 3.

Обобщая результаты проведенных вычислительных экспериментов можно сказать, что на ряде технологий параллельного программирования, таких как OpenMP, Intel® Cilk Plus и Intel® TBB с использованием механизма Parallel for, при реализации приложения удалось добиться ожидаемой тенденции практически линейного ускорения на многоядерных архитектурах. Причем OpenMP опережает остальные технологии на данной модельной задаче с точки зрения производительности и масштабируемости.

Анализ трудоемкости разработки

Полученные результаты наглядно представлены на рис. 4, 5. Следует отметить основные тенденции, которые характеризуют применимость рассмотренных технологий к выбранной модельной задаче.

Так, процесс обучения технологии OpenCL оказался наиболее трудоемким среди всех технологий, рассматриваемых в работе. Разработчику, не имеющему достаточного опыта разработки приложений для гетерогенных систем, необходимо время для осознания моделей, лежащих в основе OpenCL. Обучение механизму логических задач при использовании библиотеки TBB также требует дополнительных усилий по сравнению с OpenMP и Cilk Plus.

Развертывание среды программирования для OpenMP и Cilk Plus состоит лишь в подключении соответствующих заголовочных файлов и проверке работоспособности. Это значит, что для начала программирования потребуется меньше времени (см. рис. 4А) по сравнению с TBB и OpenCL, где требуется установка дополнительных расширений.

Процесс написания параллельного кода для решения данной прикладной задачи на OpenCL потребовал наибольшего времени. Это объясняется сложностью изучения архитектуры стандарта, большим количеством ошибок в коде программы даже у опытных программистов

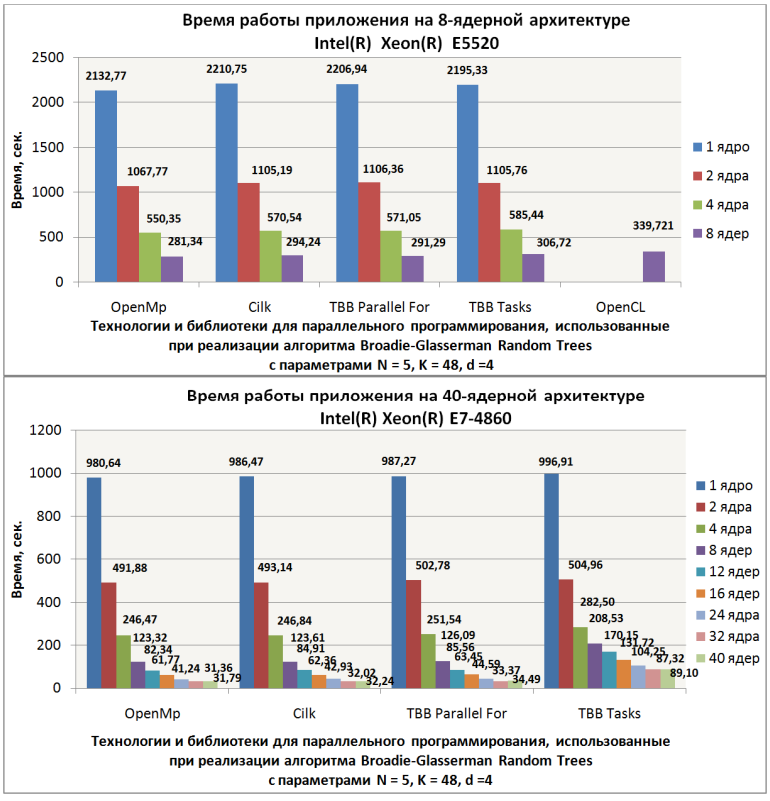

Рис. 3. Время работы на многоядерных системах

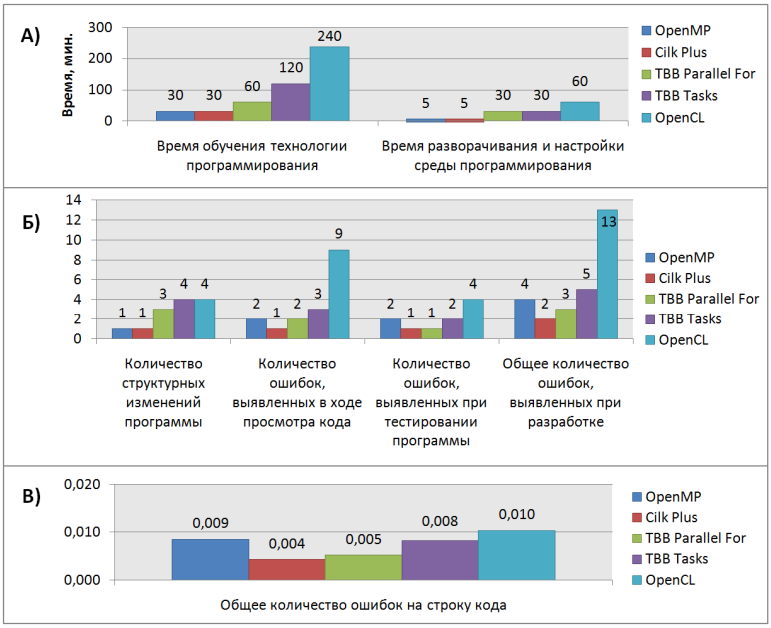

Рис. 4. Сравнение значений метрик групп «Подготовка» и «Надежность» для модельной задачи

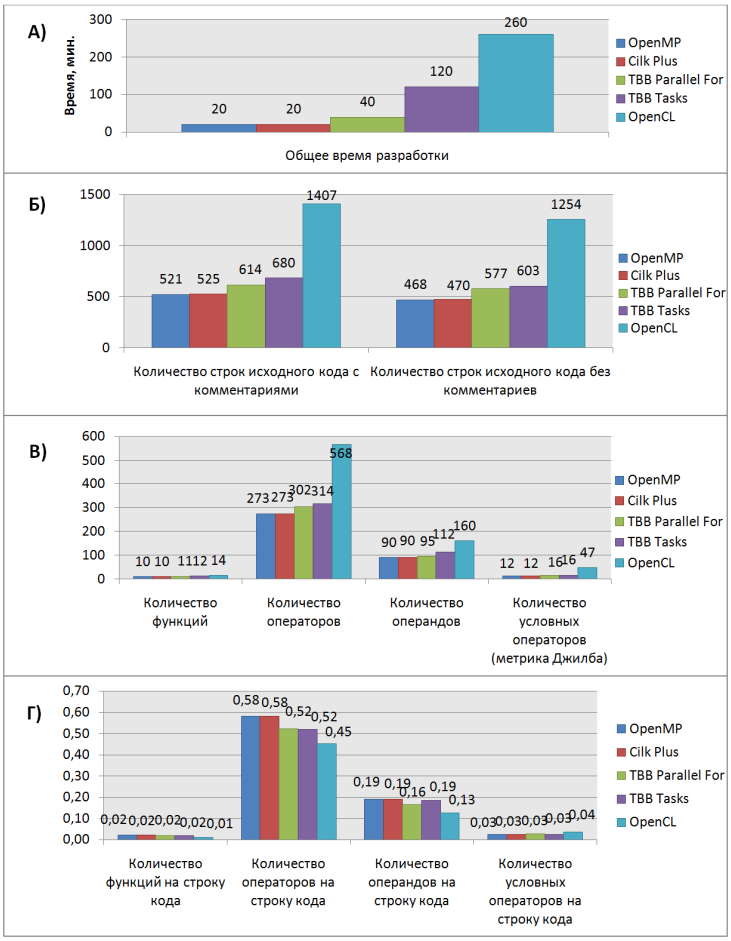

Рис. 5. Сравнение значений метрик групп «Процесс» и «Размер исходного кода» для модельной задачи

ввиду отсутствия отладчика ядер (см. рис. 4Б, 4В, 5А). Объектно-ориентированный подход, используемый в библиотеке Intel® TBB, требует реструктуризации последовательного кода программы при переходе к параллельному, это также является источником временных затрат и ошибок при разработке.

Количество строк исходного кода, представленное на рис. 5Б, для технологии OpenCL более чем вдвое превосходит аналогичный пока-

затель у остальных технологий. Это объясняется моделью построения OpenCL-программы и большим количеством вызовов низкоуровневых функций (например, функций передачи параметров объекту ядра). Все это делает код программы на OpenCL менее насыщенным функциями, операторами и операндами (см. рис. 5Г). Однако показатель среднего количества ошибок на строку кода у OpenCL является самым высоким из рассмотренных. Большое количество условных операторов в OpenCL (см. рис. 5В) также является следствием модели программирования. После каждой функции инициализации OpenCL программы для выбранных устройств проверяется возвращаемый код ошибки. Подобный подход помогает в отладке OpenCL-приложений.

В итоге, при использовании технологий OpenMP и Intel® Cilk Plus для решения рассмотренной прикладной задачи удалось создать приложения, которые наряду с лучшей производительностью требуют наименьших затрат на подготовку и реализацию, менее подвержены ошибкам со стороны программиста и обеспечивают более компактный код.

Заключение

В статье предлагается методика анализа трудоемкости разработки параллельных программ. Указанная методика включает систему метрик для проведения анализа, выбор ролевых групп участников процесса, подробные инструкции, регламентирующие процесс анализа, и вспомогательные материалы (шаблоны заполняемых документов в формате Microsoft Excel, ссылки на дополнительные инструменты подсчета метрик).

Авторами проведен сравнительный анализ некоторых широко распространенных технологий и библиотек для параллельного программирования в системах с общей памятью, включая гетерогенные системы с использованием графических процессоров. Анализ проводился на основе решения тестовой задачи – определения справедливой цены опционов бермудского типа методом имитационных деревьев (Broadie-Glasserman Random Trees). Для сравнения были выбраны следующие технологии и библиотеки: OpenMP, Intel TBB, OpenCL, Cilk Plus. Сравнительный анализ включал измерение производительности программного обеспечения, созданного с использованием указанных технологий и библиотек, а также количественный анализ трудоемкости разработки с использованием предложенной системы метрик. В целом наблюдаемая тенденция – усложнение работы в случае использования технологии программирования для графических процессоров по сравнению с аналогичными технологиями для центральных процессоров.

В статье авторы не претендуют на общность полученных результатов апробации, выполненной лишь на одной задаче, характерной особенностью которой является потенциал для идеального распараллеливания (используется метод Монте-Карло). В дальнейшем планируется представить результаты апробации предложенного подхода при чтении курсов по параллельному программированию на ВМК ННГУ с привлечением других разработчиков и расширением спектра решаемых задач.

Примечания

[1]Исследователь – Таблица для анализа метрик.xlsx – шаблон доступен для скачивания по адресу: http://itlab.unn.ru/?doc=1144

[2]Разработчик – Таблица для сбора метрик.xlsx – шаблон доступен для скачивания по адресу: http://itlab.unn.ru/?doc=1144

[3]Свободно распространяемое приложение SourceMonitor доступно для скачивания по адресу: http://www.campwoodsw.com/sourcemonitor.html

Список литературы

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.

2. Гергель В.П. Теория и практика параллельных вычислений. М.: Бином, 2007.

3. Корняков К.В., Кустикова В.Д., Мееров И.Б. и др. Инструменты параллельного программирования в системах с общей памятью: Учебник / Под ред. проф. В.П. Гергеля. Издание второе, исправленное и дополненное. М.: Изд-во МГУ, 2010. 272 с.

4. Горбунова А.С., Мееров И.Б., Никонов А.С. и др. Эффективное использование вычислительных ресурсов для определения справедливых цен опционов бермудского типа // Науч.-техн. вестник Санкт-Петербургского госуниверситета информационных технологий, механики и оптики. Вып. №54. СПб.: Изд-во СПбГУ ИТМО, 2008. С. 145–151.

5. Бастраков С.И., Донченко Р.В., Мееров И.Б., Половинкин А.Н. Особенности оптимизации вычислений в прикладных программах на языке С на примере оценивания опционов европейского типа // Научно-технический вестник Санкт-Петербургского госуниверситета информационных технологий, механики и оптики. Вып. №5(69). СПб.: Изд-во СПбГУ ИТМО, 2010. С. 95–100.

6. Hochstein L., Basili V.R., Vishkin U., Gilbert J. A pilot study to compare programming effort for two parallel programming models. 28 December, 2007 (Preprint submitted to Elsevier).

7. Nanz S., Torshizi F., Pedroni M., Meyer B. A

Comparative Study of the Usability of Two Object-oriented Concurrent Programming Languages. arXiv:1011.6047, 2010.

8. Баркалов К.А., Гергель В.П., Гергель А.В. и др. Организация и проведение всероссийской школы по суперкомпьютерным технологиям // Открытое и дистанционное образование. 2010. №2. С. 24–29.

9. Гергель В.П., Линев А.В., Мееров И.Б. и др. Об опыте проведения программ повышения квалификации профессорско-преподавательского состава по направлению «Высокопроизводительные вычисления» // Открытое и дистанционное образование. 2010. №3. С. 15–20.

10. Гергель В.П., Стронгин Р.Г. Опыт Нижегородского университета по подготовке специалистов в области суперкомпьютерных технологий // Вестник Нижегородского университета им. Н.И. Лобачевского. 2010. № 3(1). С. 191–199.

11. Новичков А. Метрики кода и их практическая реализация в IBM Rational ClearCase. URL: http://www.ibm.com/developerworks/ru/edu/0108novich/section2.html (дата обращения: 04.08.2008).

12. Петрухин В.А., Лаврищева Е.М. Метрики качества программного обеспечения. URL: http://www.intuit.ru/department/se/swebok/10/3.html (дата обращения: 24.09.2008).

13. Black F., Scholes M. The pricing options and corporate liabilities // Journal of Political Economy. 1973. Vol. 3. P. 637–659.

14. Broadie M., Glasserman P. Pricing American-style securities using simulation // Journal of Economic Dynamics and Control. June 1997. Vol. 21. P. 1323–1352.

15. Горбунова А.С., Козинов Е.А., Мееров И.Б. и др. Параллельная реализация одного алгоритма нахождения цены опционов бермудского типа // Высокопроизводительные параллельные вычисления на кластерных системах. Матер. Пятого Международ. науч.-практ. семинара / Под ред. проф. Р.Г. Стронгина. Н. Новгород: Изд-во Нижегородского госуниверситета, 2005. С. 60–67.

16. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем: Учебное пособие. Н. Новгород: Изд-во ННГУ, 2003. 184 с.

17. Горбунова А.С., Козинов Е.А., Мееров И.Б., Шишков А.В. Оптимизация вычислений в задаче определения справедливой цены опциона бермудского типа при некоторых значениях параметров // XI Нижегородская сессия молодых ученых. Математические науки. Материалы докладов. Н. Новгород, 2006. С. 22–23.

A NOVEL APPROACH TO ESTIMATE COMPLEXITY OF PARALLEL SOFTWARE DEVELOPMENT

I.B. Meyerov, A.V. Rusakov

We propose a novel approach to estimate the complexity of parallel software development and implementation using different parallel programming technologies and tools in shared memory systems. A system of metrics and the technique for their calculation and usage are presented. The proposed approach is illustrated by the example of a financial mathematics problem.

Keywords: parallel programming, shared memory systems, development complexity, parallel programming technologies and tools.