УДК 681.3

Р. А. ЮЩЕНКО

ИЗМЕРЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ПАРАЛЛЕЛЬНЫХ КОМПЬЮТЕРОВ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Ключевые слова: *параллельные вычисления, производительность, оптимизация, коммуникационные затраты, кластеры, суперкомпьютеры, высокопроизводительные вычисления, обработка данных, MIMD, HPC.*

ВВЕДЕНИЕ

Суперкомпьютеры используют для решения задач в такой постановке, объемах и детализации, с которыми персональный компьютер не может справиться в принципе из-за ограничений памяти и производительности. Скорость суперкомпьютеров существенно возросла за последнее время и продолжает расти быстрыми темпами. Хотя тактовая частота процессоров достигла физического предела, распараллеливание алгоритмов открыло возможности для дальнейшего увеличения производительности. Такая тенденция в равной степени затронула суперкомпьютеры с массовым параллелизмом, кластерные комплексы и персональные компьютеры. Появились суперкомпьютеры, производительность которых превышает 1 ПФлопс.

Признанный авторитет в области линейной алгебры и высокопроизводительных вычислений Дж. Донгарра привел ряд интересных фактов. В среднем компьютер, лидирующий по производительности в списке «Top-500», за 6–8 лет занимает в нем последнее место. Суперкомпьютер, занимавший последнее место в этом списке, через 8–10 лет по мощности сравнивается с персональным компьютером [1].

Поскольку потребляемая мощность процессора пропорциональна кубу тактовой частоты, а хорошее распараллеливание дает линейное увеличение скорости, использование большого количества процессоров с пониженной тактовой частотой позволяет значительно увеличить производительность при малых энергетических затратах. Так, компания Intel имеет экспериментальный образец процессора, содержащего 80 ядер, пиковой производительностью в 1 ТФлопс и потребляющего всего 67 Вт электроэнергии. Но мир программного обеспечения оказался не готов эксплуатировать параллелизм, поскольку за время использования обычных компьютеров накоплен огромный багаж готовых последовательных программ. Формальные методы распараллеливания последовательных программ на практике не позволяют получить существенного ускорения. Усложняет ситуацию и частая смена сложившихся технологий параллельного программирования для суперкомпьютеров, поэтому программы часто приходится переписывать. Однако компьютерный мир уже стал параллельным и иного пути достижения максимальной производительности нет.

Цель статьи — рассмотреть вопросы, связанные с производительностью параллельных компьютеров, влияющие на нее факторы, используемые для ее измерения инструменты.

ОПРЕДЕЛЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ

Под производительностью компьютера (в том числе параллельного) понимается количество (арифметических) операций в секунду. Под операциями принято понимать как сложение, так и умножение. Тип операндов определяет единицу измерения: мипсы — для целых чисел, флопсы — для чисел с плавающей запятой[1]. Способ измерения количества операций определяет тип производительнос-

[1]Далее, если не указано специально, подразумеваются операнды двойной точности (double precision).

ти. Под пиковой (идеальной) производительностью понимают максимальную теоретически достижимую производительность, исходя из спецификации процессора при условии максимальной загруженности, моментальной скорости доступа к памяти и отсутствии любых накладных расходов. Пиковая производительность одного процессора вычисляется как $R_{\text{peak}} = F_c n_c k$, где F_c — тактовая частота ядра, n_c — количество ядер, k — фиксированная константа, зависящая от архитектуры процессора. Пиковая производительность параллельного компьютера вычисляется как сумма производительности его процессоров. Далее для простоты процессор — любое устройство, генерирующее поток инструкций (узел кластера, процессор, ядро).

Сравнивать пиковые производительности некорректно, если архитектуры сравниваемых процессоров различаются (в английском жаргоне бытует слово machoflops, иронизирующее над завышенными показателями пиковой производительности). Но для позиционирования компьютеров на рынке необходим унифицированный индикатор производительности. Стандартом де-факто при сравнении производительности компьютеров считают результаты пакета LINPACK Benchmark (далее — тест LINPACK). В оперативной памяти пакет генерирует систему линейных алгебраических уравнений (СЛАУ) с плотной матрицей коэффициентов, заполненную псевдослучайными числами, и решает ее методом LU-разложения. Этот алгоритм интенсивно использует процессор, оперативную память и коммуникационные каналы, оставаясь весьма эффективным, поскольку нагрузка на процессор преобладает над остальными.

Отметим, что говорить о применении результатов любых тестовых пакетов (в том числе LINPACK), предназначенных для измерения «реальной» производительности, для изучения конкретной параллельной программы можно лишь после дополнительного исследования корреляционной зависимости между результатами тестового пакета и производительностью программы. Это показано далее для отношения изоэффективности.

Достичь высокой производительности компьютера можно только в искусственных условиях[2], поэтому для параллельных программ обычно говорят не о производительности, а об эффективности. Достижение высокой эффективности параллельной программы начинается с оптимизации последовательной программы, включающей эксплуатацию кэш-памяти и векторных операций процессора. Обычно эту работу перекладывают на компилятор. По результатам исследований, проведенных в МГУ, использование компилятора, оптимизирующего программу под конкретную архитектуру, может увеличить скорость вычислений на порядок, позволяя, таким образом, лучше эксплуатировать имеющееся «железо».

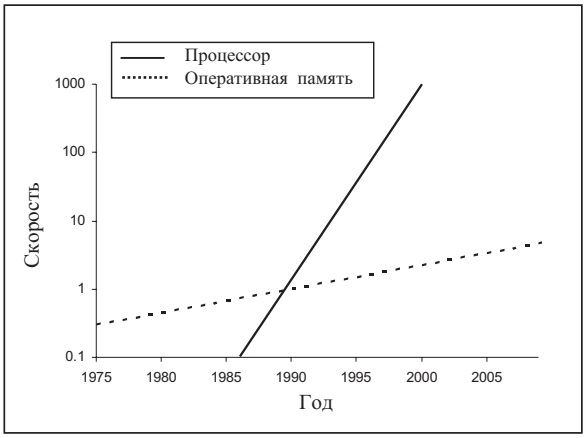

Рис. 1

Исследования также показали, что наибольшая эффективность достигается на задачах, в которых количество арифметических операций значительно превышает доступ к оперативной памяти. На рис. 1 условно показаны кривые скорости арифметических операций процессора по сравнению со скоростью доступа к памяти, и очевидно, что последняя растет

[2] Например, решая задачу $x \leftarrow a + b$, можно увеличить производительность, переписав выражение: $x \leftarrow 1a + 1b$. Тогда будет загружен не только сумматор, но и умножитель, а процессор сможет распараллелить вычисление.

медленнее. Однако многие алгоритмы можно оптимизировать для эффективного использования кэш-памяти процессора, на порядок увеличив скорость вычислений. Для многих математических задач существуют готовые библиотеки. Например, для операций с векторами и матрицами принят стандарт де-факто BLAS со множеством реализаций. Одна из реализаций, ATLAS, при сборке автоматически оптимизируется для конкретной платформы.

ОЦЕНКА ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Эффективность параллельной программы часто оценивают, сравнивая ее скорость с последовательной либо параллельной программой, но выполняющейся на одном ядре. Таким образом, можно оценить перспективы распараллеливания, а также оптимальное количество ядер.

Пусть параллельная программа выполняется на N ядрах за время $T(N) = T_N$. В случае идеального параллелизма время решения равно $T_N = T_1 / N$. Но такого на практике нельзя добиться даже тогда, когда решаемая задача не имеет обменов между ядрами (в литературе декомпозицию, не имеющую обменов между ядрами, называют тривиальным параллелизмом). Например, каждый отдельный прогон совершенно независим при моделировании методом Монте-Карло. Если выполнять каждый прогон на отдельном ядре, можно получить хорошее время распараллеливания, но оно все равно будет не идеальным, поскольку разные прогоны могут выполняться за разное время.

Для оценивания эффекта распараллеливания рассматривают понятия ускорения ($S(N)$) и эффективности ($E(N)$):

$$S(N) = \frac{T_1}{T_N}, \tag{1}$$

$$E(N) = \frac{S(N)}{N} = \frac{T_1}{N T_N}. \tag{2}$$

Эффективность может значительно ограничить последовательная часть программы, т.е. остающаяся последовательной часть, поскольку считают, что время ее выполнения несущественно либо распараллеливать ее «неинтересно». Например, это случается при решении конечно-разностных и конечно-элементных задач, если вычислительная решетка генерируется последовательно, а сам расчет выполняется параллельно. Если число ядер велико, то вычисления занимают намного меньше времени, чем генерация вычислительной решетки.

Серия отношений и теорем, предложенных Амдалом в 1967 г. и впоследствии ставших известными как законы Амдала, устанавливает теоретическую верхнюю границу распараллеливания при фиксированной доле последовательных вычислений:

$$S(N) = \frac{1}{\sigma + \dfrac{1-\sigma}{N}}, \tag{3}$$

где σ — доля последовательных вычислений.

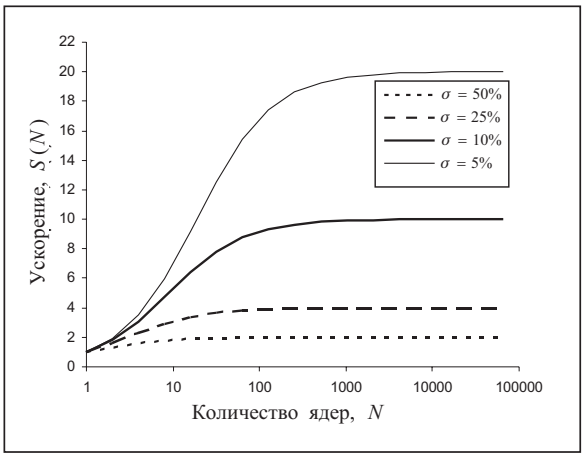

Рис. 2

На рис. 2 приведена зависимость ускорения (S) от количества ядер (N) при различных значениях фиксированной

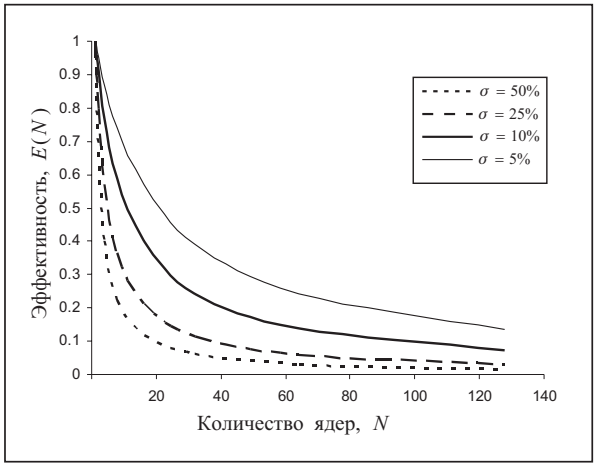

Рис. 3

доли последовательных вычислений (σ), а на рис. 3 — эффективности (E) от количества ядер (N). Весьма пессимистичным выглядит то, что, если последовательная доля вычислений составляет всего 5%, уже на 120 ядрах имеем эффективность менее 15%.

На практике доля последовательных вычислений в программе уменьшается с ростом размеров задачи (n). Для получения требуемой эффективности можно увеличить n, например, сделав модель более детальной. Этот принцип положен в основу закона Густавсона–Барсиса, предложенного в 1988 г. Используя этот закон, можно подбирать количество ядер или размер задачи для достижения нужной эффективности.

Пусть s — доля времени выполнения последовательной части параллельной программы,

$$s(n) = \frac{\sigma(n)}{\sigma(n) + \dfrac{\varphi(n)}{N}},$$

где $\varphi(n)$ — доля параллельных вычислений, $\sigma(n)$ — доля последовательных вычислений. Тогда оценка масштабируемого ускорения Густавсона–Барсиса имеет вид

$$S = \frac{T_1}{T_N} \le s + N(1-s) = N + (1-N)s. \qquad (4)$$

Закон Густавсона–Барсиса исходит из того, что для любого числа ядер можно подобрать задачу достаточно большого размера, чтобы она выполнялась эффективно. Однако на практике оценки часто чересчур оптимистичны.

Альтернативной концепцией параллелизма считают принцип макроконвейерных вычислений, предложенный В.М. Глушковым в 1978 г. и исследуемый сотрудниками Института кибернетики в 1980-х гг. Макроконвейер — модель параллельного компьютера с распределенной памятью, в которой процессоры соединены коммуникационными каналами. Каждое вычислительное устройство обладает своей локальной памятью, а данные между устройствами передаются по каналам. Кроме того, концепция подразумевает наличие входного канала для исходных данных и выходного — для результатов.

Модель макроконвейерных вычислений весьма точно отражает причины низкой производительности таких систем — накладные коммуникационные затраты. Под коммуникационными затратами $\kappa(n,N)$ подразумевают задержки, связанные с пересылкой данных по каналам и синхронизацией процессов. В зависимости от способа распараллеливания и топологии связей между вычислителями рассматривают итеративные, волновые и смешанные макроконвейеры с синхронным либо асинхронным обменом данными. Для каждого из них найдены оценки верхней границы эффективности [2].

По сути, макроконвейер воплощает модель первого в СССР компьютера с распределенной памятью, архитектурно аналогичного современным кластерным комплексам. Еще в 1980-х гг. В.М. Глушков видел перспективы таких компьютеров, способных масштабировать вычисления на сотнях и тысячах процессоров при практически линейном росте стоимости. Он также предвидел проблемы, возникаю-

щие при разработке параллельных алгоритмов и программ. В настоящее время, когда появились параллельные компьютеры с сотнями тысяч процессоров, оптимизация макроконвейерных сетей является особо актуальной задачей, и к ней часто возвращаются в публикациях, но используя новую терминологию [3].

Отношение изоэффективности позволяет определить диапазон ядер, на котором достигается необходимая эффективность [4]. Для этого ускорение рассматривается как функция $S(n,N)$, где n — размер задачи, а N — количество процессоров.

Вычислительной системой назовем параллельную программу, выполняемую на параллельном компьютере. Под масштабируемостью вычислительной системы понимают ее способность сохранять эффективность вычислений при наращивании N.

Определим накладные затраты как $T_0(n,N)=(N-1)\sigma(n)+N\kappa(n,N)$. Подставим их значение в формулу ускорения $S(n,N)=\dfrac{T(n,1)}{T(n,N)}\le\dfrac{N(\sigma(n)+\varphi(n))}{\sigma(n)+\varphi(n)+T_0(n,N)}$.

Предположим, эффективность (2) постоянна. Тогда выполняется неравенство

$$T(n,1)\ge CT_0(n,N), \tag{5}$$

называемое неравенством изоэффективности. Отношение изоэффективности используют для определения степени масштабируемости вычислительной системы. Для этого неравенство изоэффективности (5) рассматривают в виде $n\ge f(N)$, где $f(N)$ — некоторая функция.

Функция $M(n)$ задает объем памяти, необходимый для решения задачи размерности n, а функция масштабируемости $\mu(N)=\dfrac{M(f(N))}{N}$ — объем памяти, необходимый одному ядру для сохранения эффективности. Чтобы сохранить эффективность при увеличении N, нужно наращивать размер задачи в соответствии с неравенством изоэффективности.

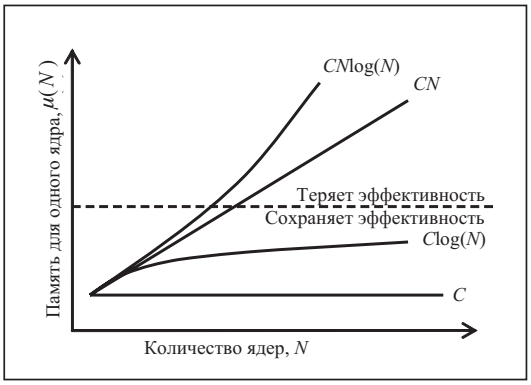

Рис. 4

На рис. 4 функция $\mu(N)$ проиллюстрирована на нескольких примерах. Штрихом условно показан объем памяти компьютера. Как видно из графика, при линейном и выше росте $\mu(N)$ вычислительная система теряет эффективность [4].

Функцию масштабируемости задают для вычислительной системы, исходя из вычислительной сложности последовательного и параллельного алгоритмов, сложности затрат на коммуникацию.

Хорошей мерой оценки достигнутой эффективности для готовой параллельной программы при ее работе на конкретном параллельном компьютере считается метрика Карпа–Флатта, предложенная в 1990 г. Она позволяет экспериментально оценить последовательную часть в программе и долю затрат на коммуникацию:

$$e(n,N)=\frac{\sigma(n)+\kappa(n,N)}{\sigma(n)+\varphi(n)}=\frac{\dfrac{1}{S(n,N)}-\dfrac{1}{N}}{1-\dfrac{1}{N}}=\frac{\dfrac{T_1(n)}{T_N(n)}-\dfrac{1}{N}}{1-\dfrac{1}{N}}. \tag{6}$$

Результаты (1)–(6) дают представление о параллелизме и оценке накладных затрат. Но для некоторых прикладных задач обработки данных больших объемов

критичным ресурсом является система дискового хранения данных. Такие задачи обладают низкой макроконвейерностью, поэтому увеличение числа ядер может привести к тому, что система хранения данных станет узким местом вычислений, а использование бóльшего числа процессоров для расчета может не только не увеличить, но даже уменьшить время вычислений.

Методика оценивания эффективности параллельных программ с однородной обработкой данных [5] позволяет оценить время выполнения параллельной программы на основе небольшого числа экспериментов. Идея методики состоит в том, что на компьютере, называемом базовым, проводят серию экспериментов с последовательной или параллельной программой. При выполнении этой программы собирают разные профильные данные, например время выполнения, доля времени доступа к дисковой памяти и пр. Для измерения встраивают счетчики непосредственно в текст программы либо используют средство профилирования. Для прогноза времени выполнения на произвольном кластере в формулы оценок эффективности подставляют коэффициенты характеристик базового компьютера и узла кластера, а также профильные данные. В результате получим оценочное время программы как функцию от числа процессоров. Для нее можно подобрать оптимальное N. Однако методика [5] имеет принципиальный недостаток, поскольку не учитывает динамических характеристик параллельного алгоритма, а значит, результаты нельзя применить, если размер решаемой задачи является переменным. Поэтому данная методика несколько расширена и для большей точности результатов разработаны оценки для разных паттернов параллельного программирования [6].

Термин «паттерн» впервые предложил архитектор К. Александр в 1977 г.; он означает готовые шаблоны решений для конкретных строительных объектов. Каждый «александровский» паттерн обладает тремя обязательными элементами: название, описание типичной проблемы и шаблон типового решения. Между описанием проблемы и ее решением К. Александр писал заглавными буквами слово «ПОЭТОМУ».

Использование паттернов в программировании обусловлено тем, что современные языки программирования насыщены абстракциями, которыми необходимо оперировать для эффективного написания надежных программ. Особенно это относится к языкам высокого уровня вроде C#. Обычное представление программы в виде блок-схемы удобно для детального описания сложных алгоритмов задач, но далеко недостаточно для описания комплексов программных систем. Современные языки предоставляют мощные средства, используя которые можно создавать изящные и удобные способы решения задач. В работах К. Бека и У. Каннингема, а также в знаменитой работе Е. Гамма, Р. Хельма, Р. Джонсона [7] аккумулирован программистский опыт в виде каркасов для решения задач на основе объектно-ориентированного подхода.

Термин «паттерн» (шаблон) использован в объектно-ориентированном проектировании для описания шаблонов классов и взаимосвязей между ними при решении типичных задач. Такой шаблон задает описание класса или объекта, его отношение с другими классами либо объектами (агрегирование, суперпозиция, ассоциация, наследование и др.) и сценарии их взаимодействия для решения классов задач. Так, паттерн «адаптер» — это класс, задача которого — обеспечение единого интерфейса для множества классов со сходными функциями, но различными интерфейсами. Названия паттернов, описанных в работах Е. Гамма, вошли в лексикон объектно-ориентированного проектирования, а такие, как «синглетон», — в грамматику языков программирования. В поддержку развития паттернов программирования в 1993 г. основана некоммерческая организация Hillside Group, проводящая ежегодные конференции Pattern Language of Programs (PLoP).

Область применения паттернов Гамма — объектно-ориентированное проектирование. В параллельном программировании возможного разнообразия решений и инструментов намного больше, поэтому работа Т. Мэттсона [8] стала крайне популярна и цитируема. Предложенная в ней каскадная технология проектирования параллельных программ, основанная на языке паттернов, подразумевает четыре стадии проектирования: поиск параллельности, схема алгоритма, вспомогательные структуры и механизмы реализации. Однако процедура проектирования на основе паттернов не гарантирует получение наилучшего проекта. Например, алгоритм распараллеливания, идеально подходящий к задаче, может оказаться неэффективным

при реализации на конкретном компьютере. Для получения оценки эффективности параллельной программы на кластере нужно рассмотреть паттерны, исходя, прежде всего, из средств реализации. Для задач однородной обработки данных результаты исследований показали, что наиболее целесообразны паттерны «мастер-рабочий», «одна программа — много данных» и «разделяй и властвуй».

По результатам моделирования методом Монте-Карло в среде GPSS для паттернов стадии проектирования структур программ «мастер-рабочий» (МР), «одна программа — много данных» (ОПМД), «разделяй и властвуй» (РВ) сформулированы рекомендации, приведенные ниже. Паттерны, обладающие преимуществом, показаны первыми.

Преимущество паттернов с разных точек зрения			
По требованиям к дисковой памяти	По балансировке загрузки	По затратам на коммуникацию	По масштабируемости
РВ	МР	ОПМД	МР
МР	ОПМД	МР	ОПМД
ОПМД	РВ	РВ	РВ

Э. Коеинг предложил концепцию «антипаттернов», которые помимо сути проблемы и типового решения содержат описание типовых неверных решений. В русле параллельного программирования имеем антипаттерны лишь для многопотокового программирования.

ПРОГРАММНЫЕ ПАКЕТЫ ИЗМЕРЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ

Существует разнообразие программного обечпечения, позволяющего оценить производительность параллельных компьютеров. Как правило, такое измерение проводят в два этапа. Первый — измерение производительности отдельных узлов и компонентов, включая процессоры, оперативную память, сеть, дисковые массивы. Задача этого этапа — выявить сбои в компонентах комплекса, не обязательно приводящие к потере работоспособности: часто компьютер продолжает работать со сниженной скоростью. Второй этап — тестирование производительности в целом.

Для тестирования процессора используют однопроцессорную и SMP-версии теста LINPACK, а оперативной памяти — утилиту MEMTEST86, STREAM или RandomAccess. Цель экспериментов — убедиться, что все компоненты имеют одинаковые показатели скорости.

Скорость сети критична для многих параллельных вычислений на компьютерах с распределенной памятью, поскольку она фактически определяет коммуникационные затраты. Обычно измеряют латентность — задержку при передаче сообщения минимальной длины и пропускную способность — максимальное количество байтов, передаваемых за единицу времени. Если параллельная среда подразумевает обмен данными в рамках стека протоколов TCP/IP, измерить латентность и пропускную способность можно с помощью утилиты «beff». Но часто для достижения максимальной скорости вычислительная среда использует сеть, минуя стек TCP/IP. Тогда используют утилиты совместно с сетевым оборудованием, например программу «bw», входящую в состав комплекта программ OFA[3].

Поскольку параллельные программы выполняются в вычислительной среде, она как фактор влияет на производительность. Рассмотрим MPI (Message Passing Interface) как стандарт де-факто для распараллеливания на компьютерах с распределенной памятью. В настоящее время сосуществуют две спецификации MPI: MPI 1.2, насчитывающая порядка 128 функций для обмена сообщениями, и MPI 2.0 (более 500 функций). Интенсивность их использования зависит, прежде всего, от логики (модели распараллеливания) параллельной программы. Исследование скорости выполнения важно, во-первых, для выбора наиболее подходящей модели распараллеливания, а во-вторых, чтобы убедиться, что все необходимые для реализации параллельной программы функции работают должным образом и с ожидае-

[3] Open Fabric Alliance (OFA) — международная некоммерческая организация, финансируемая разработчиками высокоскоростных технологий. Ее задача — унификация протоколов высокоскоростного обмена данными для различных сетевых технологий.

мой эффективностью для заданных объемов данных. Автор сталкивался со случаями, когда из-за ошибок в реализации MPI операции коллективного обмена (MPI_Bcast) выполнялись намного медленнее последовательной рассылки операциями типа «точка-точка» (MPI sub Send/MPI sub Recv).

Пакет SKaMPI [9] предназначен для углубленного тестирования функций MPI. Версия 4 от 2001 г. включает полностью автоматизированный процесс тестирования с генерацией отчета в формате PostScript (.ps). Последняя, пятая, версия обладает сложным механизмом настроек для описания задачи тестирования, что позволяет сфокусировать исследования на определенную функциональность.

Проведенные на комплексе Инпарком-64 эксперименты свидетельствуют, что эффективность разных реализаций MPI может весьма отличаться. На рис. 5 приведены результаты сравнения скорости передачи сообщения для разных реализаций MPI, поддерживающих сети InfiniBand. Кривыми проиллюстрирована зависимость времени передачи сообщения от его объема в байтах. В табл. 1 перечислены реализации MPI, применяя которые удалось достичь максимальной скорости отдельных операций.

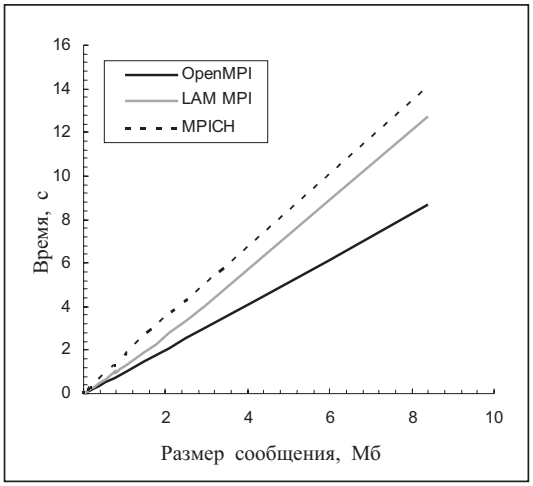

Рис. 5

Для некоторых задач, особенно связанных с однородной обработкой данных большого объема, крайне важна скорость доступа к дисковой памяти. Одна из простых утилит измерения скорости дисковой памяти — Bonnie.

Таблица 1. Реализации MPI, наиболее быстрые при выполнении операций

Операции MPI	Общая память		Распределенная память	
	Латентность	Пропускная способность	Латентность	Пропускная способность
MPI_Send/MPI_Recv	LAM MPI	OpenMPI	LAM MPI	OpenMPI
MPI_Bsendrecv	LAM MPI	MPICH	LAM MPI	MPICH
MPI_Barrier	OpenMPI	–	OpenMPI	–
MPI_Bcast	LAM MPI	OpenMPI	LAM MPI	OpenMPI
MPI_Reduce	LAM MPI	OpenMPI	LAM MPI	OpenMPI
MPI_Allreduce	LAM MPI	OpenMPI	LAM MPI	OpenMPI
MPI_Gather	LAM MPI	OpenMPI	LAM MPI	OpenMPI
MPI_AllToAll	OpenMPI	OpenMPI	MPICH	OpenMPI

Программный пакет IoMeter используют для изучения доступа к дисковой памяти, минуя файловую систему. Также пакет позволяет тестировать доступ к дисковой памяти согласно паттерну, определяемому долей чтения/записи и уровнем «случайности» доступа.

Проведенное NERSC выявление типичных паттернов работы с дисковой памятью основано на практике собственного исследовательского центра в обслуживании научно-исследовательских расчетов для более 300 пользователей, из которых выбрано 50 наиболее интенсивно использующих дисковую память [9]. Результатом стал известный пакет IOR для измерения скорости дисковой памяти, ориентированный на высокопроизводительные вычисления.

После проверки каждого компонента каждого узла можно приступать к тестированию производительности в целом.

Тест LINPACK, в некотором смысле, является случайностью, поскольку изначально задуман как вспомогательная утилита для оценки времени решения систем линейных уравнений с использованием библиотеки LINPACK [11]. Руководство по библиотеке LINPACK за 1979 г. содержало результаты по СЛАУ 100-го порядка для распространенных в то время компьютеров. Зная эти результаты, пользователи могли выбрать подходящий компьютер для решения их задач и оценить время решения, экстраполируя данные из руководства. Со временем список «производительности на тесте LINPACK» расширялся на добровольных началах и достиг порядка 1300 компьютеров.

В настоящее время тест LINPACK[4] является стандартом де-факто для вычисления скорректированной пиковой производительности при сравнении разных компьютеров. Проект «Top-500», начатый в 1993 г., отслеживает 500 наиболее мощных суперкомпьютеров в мире и публикует результаты дважды в год. В 2008 г. начал функционировать проект Green500, участники которого публикуют рейтинг 500 суперкомпьютеров из списка «Top-500», в порядке производительности на ватт электроэнергии.

Тест LINPACK позволяет получить хорошую коррекцию пиковой производительности, приводя фактически верхнюю границу возможного распараллеливания прикладных задач. Утвердился миф, что, используя производительность, которой удалось достичь на тесте LINPACK и которую спекулятивно называют «реальной», можно адекватно спрогнозировать эффективность прикладного приложения. Это далеко не всегда так, даже если решаемая задача — нахождение решения СЛАУ. На практике удачными можно считать те прикладные программы, производительность которых превышает 10% производительности, показанной тестом LINPACK. Это происходит по ряду причин.

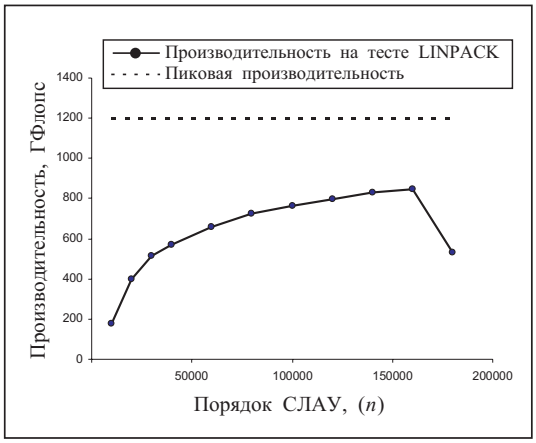

Рис. 6

1. Результаты, имеющие максимальную производительность, получают подбором размера задачи. На практике, если заданный размер задачи отличается от оптимального в большую или меньшую сторону, производительность оказывается ниже оптимальной.

На рис. 6 показаны результаты экспериментов на тесте LINPACK, проведенных на комплексе Инпарком-128 на всех ядрах. Из графика видно, что максимальная эффективность достигается, только если задача достаточно большая, чтобы занять всю оперативную память комплекса. Если задача настолько большая, что не помещается в оперативную память, время решения стремительно падает.

2. Результаты получены на тесте LINPACK перебором многочисленных параметров, влияющих на производительность (среди них: размер блока, конфигурация разбиения на блоки, топология обменов). Подбор параметров — длительный процесс (до недель) «искусного» получения максимальной производительности на тесте LINPACK. Для реальных задач заниматься перебором не хватает времени.

3. Производительность параллельной программы зависит от балансировки нагрузки разных ресурсов параллельного компьютера: ядер, оперативной памяти, сети, дисков. Тест LINPACK обладает фиксированным соотношением интенсивности использования этих ресурсов, которое редко совпадает с соотношениями прикладных задач.

[4] В зависимости от архитектуры параллельного компьютера существуют разные реализации «LINPACK Benchmark», называемые по-разному, например: «LINPACK 100», «LINPACK 1000», «LINPACK Parallel», «HPL».

4. Доступ к дисковой памяти не учитывается. Даже матрицу СЛАУ для теста LINPACK в оперативной памяти создает генератор псевдослучайных чисел, а на практике ее считывают с диска. При достаточно больших объемах (десятки и сотни Гб) чтение исходных данных и сохранение результатов составляет весомую долю вычислений.

5. Производительность по тесту LINPACK может дать только качественное представление о производительности прикладной программы, поскольку, во-первых, сложно измерить производительность прикладной программы (во флопсах), а иногда и невозможно; во-вторых, для прикладной программы важно время выполнения, нелинейно зависящее от производительности.

Производительность параллельного компьютера невозможно представить одним числом. Поскольку тест LINPACK часто используют не по назначению, ошибочно надеясь добиться эффективности, сравнимой с его показателями, разработчики LINPACK предложили новый пакет тестирования HPCC (HPC Challenge Benchmark Suite) [12]. Его задача — измерение производительности разных компонентов параллельного компьютера, способных ограничить выполнение прикладных программ. Этот пакет состоит из нескольких известных пакетов, измеряющих отдельные аспекты производительности, а также предоставляет среду для подключения дополнительных тестов. В частности, пакет HPCC включает такие известные программы измерения производительности, как STREAM (скорость оперативной памяти), HPL, DGEMM (умножение матриц), PTRANS (транспонирование матриц, FFT, RandomAccess (скорость произвольного доступа к памяти), b_eff (латентность и пропускная способность каналов). Результат измерения производительности в HPCC — не одно, а ряд значений производительности.

Коммерческий пакет SPEC CPU 2006 включает ряд параллельных расчетных модулей для известных прикладных программ. При исследовании компьютера этот пакет последовательно выполняет контрольные примеры классов задач гидродинамики, квантовой химии, биологии, конечно-разностных и конечно-элементных схем, линейного программирования, распознавания образов. Результаты исследований в этом пакете дают картину производительности компьютера, приобретаемого под решение класса задач. (На момент написания статьи указанный пакет поддерживает только компьютеры с общей памятью.)

ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ГОТОВЫХ ПРОГРАММ

Инпарком (Интеллектуальный параллельный компьютер) — семейство аппаратно-программных комплексов, разработанных Институтом кибернетики им. В.М. Глушкова НАН Украины совместно с ГНПП «Электронмаш». Интеллектуальное программное обеспечение комплексов Инпарком позволяет исследовать и решать ряд актуальных в науке и инженерии математических задач [13]. На данный момент комплексы Инпарком готовы к серийному производству (табл. 2).

Таблица 2. Линейка комплексов семейства Инпарком

Название комплекса	Количество узлов	Количество ядер	Дисковое хранилище	Производительность, ГФлопс		Год сдачи
				Пиковая	LINPACK	
Инпарком-8	1	8	–	75	63	2008
Инпарком-16	16	16	–	102	82	2005
Инпарком-32	16	32	–	205	161	2006
Инпарком-64	8	64	от 1 Тб	596	486	2007
Инпарком-128	16	128	от 1 Тб	1193	970	2008
Инпарком-256	32	256	от 1 Тб	2386	1854	2008

Для комплексов Инпарком реализована система управления кластером ACMS с централизованным управлением ресурсами и мониторингом [14]. В состав ACMS включен пользовательский интерфейс для изучения эффективности параллельной программы, прогноза эффективности, ускорения и времени выполнения параллельной программы на разном числе ядер.

Для запуска параллельной программы на комплексе Инпарком нужно создать так называемый профиль задания, в котором указать число ядер для выполнения программы. Если выполнен один профиль на разном числе ядер, автоматически собирается статистика и анализируется эффективность. На рис. 7 показано время выполнения теста LINPACK на разном числе ядер. По графику времени выполнения программы для разного числа процессоров сложно что-либо сказать об эффективности, поэтому в протоколе исследований на рис. 8 показаны также графики ускорения эффективности и накладных затрат.

На рис. 7, 8 приведены графики для теста LINPACK. Максимальное ускорение на комплексе Инпарком-256 не достигнуто, значит, увеличив число процессоров указанной архитектуры, получим лучшие результаты по тесту LINPACK, даже не наращивая размерность задачи.

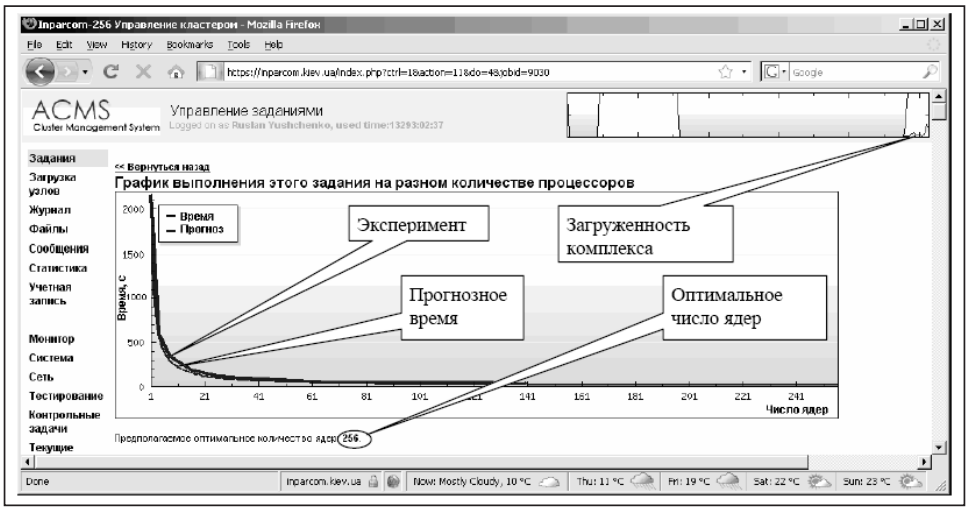

Рис. 7

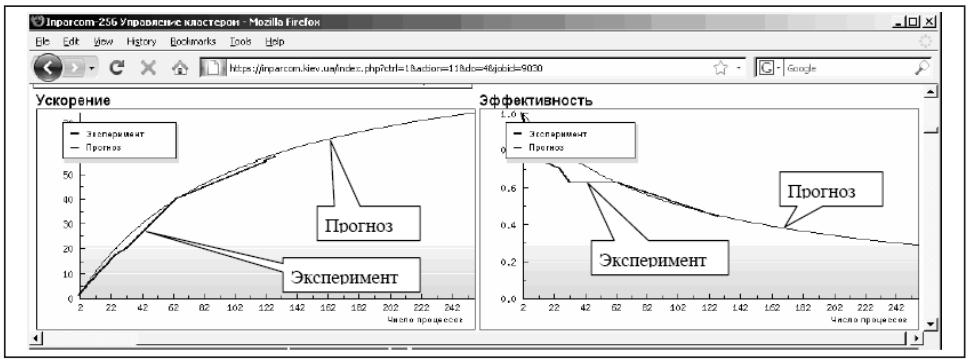

Рис. 8

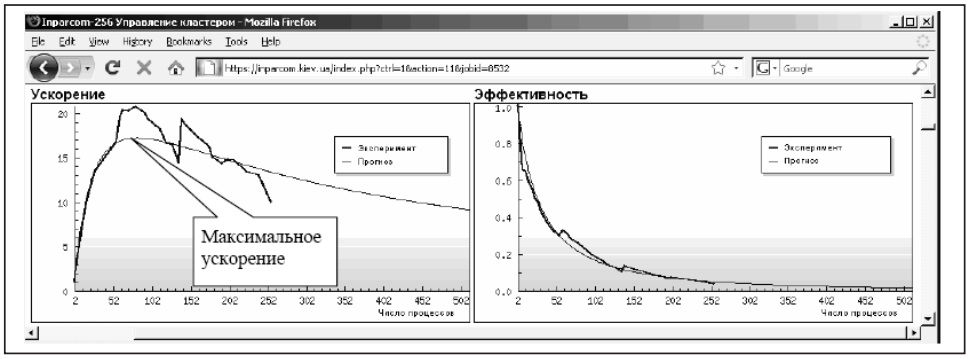

Рис. 9

На рис. 9 приведены графики ускорения и эффективности для одной из прикладных задач, решенной на комплексе. Оптимальное число ядер для этой задачи — примерно 80.

ЗАКЛЮЧЕНИЕ

В настоящее время происходит смена парадигм программирования, а эволюция оборудования опережает программные средства, компьютеры с разделенной памятью вышли за пределы производительности в 1 ПФлопс. Снова актуальны проблемы, которыми занимались отечественные ученые в 1970–1980 гг., среди них проектирование и оптимизация макроконвейерных сетей В.М. Глушкова для разработки эффективных параллельных алгоритмов с контрольными точками и возобновлением после сбоев на сверхбольшом числе процессоров [15]. Проводятся новые исследования достоверности результатов (в частности, анализ числа обусловленностей), а также эмулируемой и интервальной арифметики, арифметики со смешанной разрядностью [16].

Что касается производительности компьютеров, то не существует одного числа, достаточного для характеристики параллельного компьютера. Будущее за многокритериальным сравнением производительности под конкретный класс задач на основе результатов таких расширенных пакетов измерения производительности, как HPCC. Интеллектуальные средства исследования и прогнозирования производительности открывают новые возможности в выборе оптимального числа процессоров, планировании заданий и балансировке загрузки.

СПИСОК ЛИТЕРАТУРЫ

1. Performance of various computers using standard linear equations software: LINPACK benchmark report / J. Dongarra; Univ. of Tennessee. — 2008. — 89 p.
2. Глушков В.М., Капитонова Ю.В., Летичевский А.А. К теории проектирования схемного и программного оборудования многопроцессорных ЭВМ // Кибернетика. — 1978. — № 6. — С. 1–15.
3. Some issues in dense linear algebra for multicore and special purpose architectures: EECS Tech. Report LAWN #200 / M. Baboulin, J. Dongarra, S. Tomov. — 2008. — 615 p.
4. Grama A. Y., Gupta A., Kumar V. Isoefficiency: Measuring the scalability of parallel algorithms and architectures // EEE Computer Soc. Press. — 1993. — **1**, N 3. — P. 12–21.
5. Перевозчикова О.Л., Тульчинский В.Г., Ющенко Р.А. Построение и оптимизация параллельных компьютеров для обработки больших объемов данных // Кибернетика и системный анализ. — 2006. — № 4. — С. 117–129.
6. Ющенко Р.А. Оценка эффективности суперкомпьютерных архитектур для различных патернов параллельного программирования // Искусств. интеллект. — 2007. — № 3. — С. 447–453.
7. Gamma E., Helm R., Johnson R., et al. Design patterns: Elements on reusable object-oriented software. — New York: Addison–Wesley, 1995. — 385 p.
8. Mattson T., Sanders B., Massingil B.L. Patterns for parallel programming. — New York: Addison-Wesley, 2004. — 355 p.
9. Reussner R., Sanders P., Prechelt L., et al. SKaMPI: A detailed, accurate MPI benchmark // Proc. Int. Conf. “EuroPVM/MPI’98”. — Liverpool: Kluwer Acad. Pub., 1998. — P. 83–93.
10. Shan H., Shalf J. Using IOR to Analyze the I/O performance for HPC Platforms / Lawrence Berkeley Nat. Lab. — Paper LBNL-62647. — 2007. (http://repositories.cdlib.org/lbnl/LBNL-62647)
11. Dongarra J., Luszczek P., Petitet A. The LINPACK Benchmark: Past, present, and future // Concurrency and Computation: Practice and Experience. — 2002. — **15**, N 9. — P. 803–820.
12. Introduction to the HPC Challenge Benchmark Suite / P. Luszczek, J.J. Dongarra, D. Koester, et al. // http://icl.cs.utk.edu/projectsfiles/hpcc/pubs/hpcc-challenge-benchmark05.pdf
13. Численное программное обеспечение MIMD-компьютера Инпарком / А.Н. Химич, И.Н. Молчанов, В.И. Мова, О.Л. Перевозчикова и др. — Киев: Наук. думка, 2007. — 221 с.
14. Ющенко Р.А. Система управления кластером для комплексов Инпарком // Компьютерная математика. — 2007. — № 1. — С. 96–103.
15. Algorithmic based fault tolerance applied to high performance computing / G. Bosilca, R. Delmas, J. Dongarra, J. Langou // LAPACK Working Note. — 2008. — N 205. — P. 5–17.
16. Accelerating scientific computations with mixed precision algorithms / M. Baboulin, A. Buttari, J. Dongarra, et al. // Computational Physics. — 2008. — P. 35–49.

Поступила 07.07.2009