Обзор некоторых пакетов измерения производительности кластерных систем

Кому это нужно...

Этот материал предназначен, прежде всего, тем, кто в своей работе использует или собирается использовать кластерные системы. В работе рассматривается процесс тестирования кластеров при помощи специальных наборов тестов. Уделено внимание всем уровням рассматриваемого вопроса, начиная от общих методологических аспектов и заканчивая практическими рекомендациями настройки тестовых пакетов.

Берутся во внимание кластерные системы, ориентированные преимущественно на научные исследования (science workload) и численные расчеты.

Статья не затрагивает общие вопросы построения кластерных систем и организации параллельных вычислений.

Статья написана в надежде, что поможет специалистам и интересующимся лицам ответить на, возможно, неизвестные им вопросы в области тестирования кластерных систем.

Перед прочтением рекомендую обратить внимание на терминологию и общие вопросы кластеринга.

Введение

В рамках работы под кластерной системой (кластером) будем понимать массивно-параллельную систему, состоящую из одинаковых узлов — гомогенная среда. Каждый узел обязательно имеет центральный процессор (или несколько), локальную оперативную память (без возможности прямого доступа к оперативной памяти других узлов), коммуникационное устройство. Все узлы системы связаны посредством некоторой коммуникационной среды (FastEthernet, Myrinet, GigabitEthernet, Scalable Coherent Interface — SCI) и образуют локальную вычислительную сеть (ЛВС). В ЛВС действует среда передачи информации между узлами. Эта среда предоставляет приложениям пользователя интерфейс к своим функциям, реализованным посредством взаимодействия с операционной системой.

Частным случаем такой системы является вычислительная ферма — кластер, все узлы которого размещены в одном корпусе.

Необходимо отметить, что рассматриваемое в статье программное обеспечение нацелено на использование в UNIX подобных операционных системах, которые являются наиболее распространенными в мире кластеров.

... и зачем?

Тестирование кластеров необходимо проводить по многим объективным причинам. На мой взгляд, основной причиной для тестирования должно быть естественное желание увеличить вычислительную мощность системы. Немаловажной задачей также является определение различных факторов, прямо или косвенно сказывающихся на

производительности. Мерой производительности принято считать MIPS (millions of operation per second) — миллион операций в секунду. При тестировании систем для вычислительных задач основным критерием производительности является MFLOPS и GFLOPS миллион и миллиард операций с плавающей точкой в секунду, соответственно. Для массивно-параллельных систем значимой величиной является латентность ЛВС, выражающаяся в среднем времени передачи по сети сообщения нулевой длинны.

Увеличения производительности можно добиться разнообразными способами (не используя изменения на аппаратном уровне): изменение размерности основных величин и/или алгоритма вычислений, определение оптимальной топологии сети. Большой прирост скорости может дать правильный выбор и настройка программного обеспечения (например, опции компилятора). Выбор средств для улучшения можно сделать только после всестороннего тестирования.

Для конкретного метода решения, при помощи соответствующих тестов можно определить масштабируемость данной системы, т.е. зависимость между увеличением числа узлов в кластере и увеличением скорости расчетов. В конечном счете, при росте числа узлов будет наблюдаться увеличение производительности до некоторого предела, после которого, возможно, производительность начнет падать. Это будет обусловлено латентностью ЛВС.

Если объем обрабатываемых данных велик для полного размещения в оперативной памяти системы, то необходимо произвести оценку производительности внешних накопителей и/или файлового сервера.

В процессе настройки, отладки и сдачи кластера имеет смысл сделать объективное сравнение кластера с уже работающими подобными системами. Этот шаг необходим для того, чтобы окончательно убедится, в том, что кластерная система работает и правильно настроена. Чтобы не получилось так, что система из 16 узлов Pentium IV 3000 работает медленнее, чем система из 8 Athlon 1000. Кроме того, заказчику может ни о чем не говорить то, что производительность его кластера составляет 24 GFLOPS.

Что для этого нужно?

Benchmarks, benchmarks и benchmarks. Под benchmark'ом (тестом) будем понимать алгоритм или метод, программу или программный комплекс, отвечающий ряду требований [7]:

- Полнота. Тест должен оценивать только те параметры, для оценки которых создавался. Выдаваемые результаты должны быть непротиворечивыми, лаконичными и легкими для понимания.
- Легкость в использовании.
- Масштабируемость. Тест должен быть доступен для большого числа разного по вычислительной мощности аппаратного обеспечения.
- Переносимость. Тест должен быть доступен для большого числа разного по архитектуре аппаратного обеспечения. Основной чертой переносимости является язык программирования, а, соответственно, наличие компилятора под данную платформу.
- Репрезентативность. Вне зависимости от платформы тест должен загружать систему аналогично используемыми пользователями приложениями. Сходные по структуре тесты должны коррелировать между собой.
- Доступность. Тест должен быть доступен, в том числе и его исходный код. Однако если тест распространяется вместе со своими исходными кодами, то при представлении результатов должна быть указанна версия и все внесенные изменения.
- Воспроизводимость. При необходимости должна быть возможность повторить тест с получением аналогичных результатов. Для этого при публикации результатов необходимо предоставлять исчерпывающую информацию о программном и аппаратном обеспечении.

На сегодняшний день существует большое количество программ, подпадающих под определение benchmark'a. Для удобства изложения тесты лучше классифицировать на несколько категорий:

- «игрушечные» (toy benchmarks) — маленькие, длиной в несколько сот строк исходного кода. Как правило, решающие одну очень известную задачу. Решето Эратосфена, пирамидальная сортировка, перемешивание и многие другие.
- Микротесты (microbenchmarks) — специализированные, направленные на определение одной из основных количественных характеристик аппаратного обеспечения:
  - производительность центрального процессора;
  - производительность и пропускную способности локальной оперативной памяти;
  - скорость базовых операции ввода/вывода;
  - производительность и пропускная способность ЛВС;

  В это группу входят тесты, оценивающие производительность операций, требующих синхронизации, и тесты операционной системы (переключение контекстов, системные вызовы и создание процессов). Часто микротесты объединяются в пакеты тестов.

- Ядра (kernels) — это фрагменты кода, взятые из реальных приложений. При исполнении приложения проводят большую часть времени именно в этих фрагментах. Ядра позволяют определить скорость исполнения реальной программы на разных платформах.
- Синтетические тесты (synthetic benchmarks) оценивают производительность на основе набора большого количества показателей и не привязаны к какому-либо отдельному приложению.
- Приложения (application benchmarks) — наиболее часто используемые программы для реализации тех или иных задач. Сюда же можно отнести и псевдо-приложения. Псевдо-приложения это программы, созданные на основе реальных приложений, но адаптированные по разным причинам специально для задач тестирования.
- Пакеты тестов (benchmarks suites) — коллекции различных типов тестов с преобладанием приложений.

Из всего множества тестов существуют и такие, которые специально направлены на тестирование массивно-параллельных систем. Наиболее широко они представлены в классе ядер (HPL, NPB), приложений (NPB) и тестовых пакетов (SPEC). Из микротестов большой интерес могут представлять программы, направленные на исследование сетевого окружения (например, Netperf).

Спецификой ядер и приложений является то, что они «привязаны» к одному из основных алгоритмов численных методов или базовых операции, т.е. к той части программы, выполнение которой занимает наибольшее время. В связи с этим можно заранее выбрать нужный тест, зная, какие задачи будут решаться на кластере. Если расчеты связаны с применением прямых методов решения систем линейных алгебраических уравнений (СЛАУ), то можно использовать Linpack), в случае применения итерационных методов решения СЛАУ, следует использовать, например, ядра SP, BP из NPB и Iterative Solver Benchmark.

Под базовыми операциями понимается, например, умножение матрицы на матрицу, матрицы на вектора, вектора на вектор, генерирование псевдослучайных чисел, сортировка…

В этой статье позволю себе остановиться на двух тестах: Linpack Benchmark в версии для массивно-параллельных систем — HPL (Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers) и NAS Parallel Benchmarks (NPB).

Почему именно они? Дело в том, что этим тестам присущи ВСЕ свойства, которые должны быть у тестов:

- Полнота. Тесты замеряют только один параметр в конкретной задаче и ничего более. Результатом теста является одна величина — количество операции в единицу времени. Обычно это операции с плавающей точкой или непосредственно затраченное на расчет время.
- Легкость в использовании. Для запуска тестов не требуется специальных знаний в области программирования и математики. Достаточно базовых знаний в области администрирования UNIX подобных ОС.
- Масштабируемость, переносимость и доступность, по большей части, обеспечиваются распространением тестов в исходных кодах на самых популярных языках программирования — C и Fortran. Более того, доступны Java версии программ. Это может быть полезно для тестирования гетерогенных систем. Тестами регламентируются все возможные изменения в исходном коде. А тесты NPB вообще являются «paper and pencil» (см. ниже). Тесты в качестве среды передачи информации используют стандарт передачи сообщений Message Passing Interface (MPI), что не может не сказываться на переносимости, потому что MPI является самой распространенной средой передачи сообщений. Подавляющее большинство кластеров использует именно этот стандарт. Реализации MPI присутствуют практически во всех дистрибутивах популярной ОС Linux.
- Репрезентативность. Тесты представляют собой части реально и очень часто используемых приложений и библиотек.
- Воспроизводимость. Для того чтобы убедиться в этом, достаточно взглянуть в колоссальную базу данных уже произведенных экспериментов.

Кроме объективных причин, есть и «необъективные». Одна из них это авторитетность обоих программ (и их авторов). Вероятно, не так сложно разработать тест, отвечающий нужным требованиям, намного сложнее — сделать так, чтобы тест заслужил доверия у многочисленной аудитории.

Linpack benchmark

Немного истории

Linpack benchmark (LB) появился на свет в далеком 1979 году как дополнение к библиотеке Linpack (набор подпрограмм для решения различных СЛАУ) [4]. Целью создания LB было получение возможности оценки времени решения той или иной системы уравнений при помощи пакета Linpack. Основным автором LB можно считать J. Jack Dongarra. С тех пор Linpack был заменен более новым и расширенным пакетом LAPACK, а LB остался средством для сравнения производительности компьютеров при работе с плавающей точкой. С тех пор суть теста не изменилась.

А суть этого теста очень простая — решение СЛАУ $Ax=f$ методом LU-факторизации (LU-разложения) с выбором ведущего элемента столбца. Где A — плотно заполненная матрица размерности N.

Первоначально программа использовалась для тестирования отдельных ЭВМ при помощи матрицы с $N=100$. С ростом мощностей ЭВМ размерность матрицы была увеличена до $N=1000$. Эти размерности стали классическими и до сих пор применяются для тестирования на FLOPS отдельных машин. Реализации этих тестов на разных языках можно найти здесь.

Для задач тестирования кластеров используется указанная выше версия теста HPL. В этой версии пользователь имеет возможность задать все значимые параметры алгоритма, подбирая их для наилучшей производительности.

Каждые полгода публикуется отчет о производительности различных компьютерных систем на основе теста LB — Linpack Benchmark report. Кроме того, на сайте www.top500.org представлены 500 самых быстрых компьютеров в мире по версии LB.

Результаты тестов, сделанных при помощи HPL, могут быть предоставлены для рассмотрения на www.top500.org [5].

Алгоритм

При параллельном процессе на вычислительном кластере исходная матрица разделяется на логические блоки размерностью N_B x N_B (обычно N_B x N_B при расчетах лежит в интервале от 32 — 256). Эти блоки в свою очередь разбиваются сеткой PxQ на более мелкие. Каждый из таких блоков «достанется» отдельному процессору системы. Коэффициенты P и Q берутся в зависимости от структуры кластера, а их произведение не может быть больше доступного числа узлов.

Если в кластере 8 узлов, то допустимыми значениями PxQ будут: 1x8, 2x4, 3x2, 2x2, 1x4… При этом в расчетах будут участвовать PxQ процессоров. Именно процессоров, а не узлов. Конкретные значения P и Q следует выбирать в зависимости от коммуникационной среды.

За одну итерацию главного цикла факторизации подвергаются N_B столбцов с последующим обновлением оставшейся части матрицы. Результаты разложения пересылаются всем узлам одним из шести алгоритмов распространения (broadcast algorithm).

После разложения последовательно решается две системы уравнений: Ly=f, Ux=y.

Тест считается выполненным, если $r_n=O(1)$, $r_1=O(1)$ и $r_\infty=O(1)$, где

$r_n=||Ax-f||_\infty /(||A||_1 N_\varepsilon)$;
$r_1 =||Ax-f||_\infty /(||A||_1 ||x||_{1\varepsilon})$;
$r_\infty =||Ax-f||_\infty /(||A||_\infty ||x||_{\infty\varepsilon})$.

Эти условия означают, что задача решена верно. Где ε — точность представления чисел с плавающей точкой.

Особенности

Особенностью теста является требование наличия в системе (кроме реализации MPI) любой из следующих библиотек: Basic Linear Algebra Subroutines (BLAS) или Vector Signal Image Processing Library (VSIPL).

NAS Parallel Benchmarks

Немного истории

Если LB появился в конце 70-х начале 80-х, то NPB на целый десяток лет позже — в начале 90-х. NPB появился как дочерний проект при решении агентством NASA задач вычислительной гидродинамики. Цель создания NPB была примерно той же, что и у LB — оценка необходимой мощности аппаратного обеспечения для решения задач гидродинамики. NPB изначально был нацелен на оценку производительности параллельных вычислений на кластерах [1]. На текущий момент стабильной версией является 2.4. Версия 3.0 пока является альфа версией.

Описание

По замыслу создателей NPB является «paper and pencil» тестом, т.е. официально NPB только набор правил и рекомендаций, доступных «на бумаге». Правила декларируют практически все вопросы, которые могут возникнуть в процессе разработки:

- допустимые языки программирования;
- исчерпывающее описание всех алгоритмов (ядер, приложений, генерации случайных чисел…);
- разрядности чисел с плавающей запятой;
- разрешения/запрещения распараллеливания некоторых видов алгоритмов (например, поиск минимума в векторе);
- ввод/вывод;
- моменты замеров времени и, следовательно, включаемые в тест операции;
- правила публикации результатов тестов;
- оценку правильности полученных результатов;

и многое, многое другое.

Помимо всего прочего на сервере NASA доступны готовые реализации теста. В терминах разработчиков — это пример кода (Sample Code), написанный на Fortran-77 и стандарте MPI. До версии 2.0 программа позиционировалась как пример реализации, чтобы упростить создание своей версии конечными пользователями. После версии 2.0 — это законченный продукт. Для скачивания необходимо пройти регистрацию, правда, бесплатную.

Состав

Тест состоит из ряда простых синтетических задач: ядер (kernel benchmarks) и псевдо-приложений (application benchmarks), эмулирующих вычисления на реальных задачах (в частности в области вычислительной гидродинамики). В терминологии NPB ядра и приложения могут производить вычисления в определенных классах задач (Problem Classes): «Sample code», «Class A», «Class B» [1], «Class C» [2], «Class D» [3]. В NPB под классом понимается размерность основных массивов данных, используемых в тесте. Другими словами, класс A — это маленькие матрицы, B — большие, C — очень большие, D — огромные. Например, для теста на LU-разложение это будет размерность исходной матрицы: 12^3, 64^3, 102^3, 408^3 для каждого из перечисленных выше классов соответственно. Значения на принадлежность к определенному классу остальных ядер см. ниже.

На текущий момент существует 5 ядер: EP, MG, CG, FT, IS.

Ядро EP — «Embarrassing Parallel» основано на порождении пар псевдослучайных, нормально распределенных чисел (Гауссово распределение). Получаются числа $r_i \in (0,1)$, где i=0…2n, а n, в свою очередь, определяется классом теста. По замыслу разработчиков этот тест позволяет оценить максимальную производительность кластера при операциях с плавающей точкой при сведении к минимуму межузловых взаимодействий. Эти взаимодействия сводятся к окончательному объединению результатов, рассчитанных на каждом узле независимо от всех остальных. Этот тест может быть полезен, если на кластере будут решаться задачи, связанные с применением метода Монте-Карло. В алгоритме также учитывается время на форматирование и вывод данных.

MG — simple 3D MultiGrid benchmark. Приближенное решение трехмерного уравнения Пуассона $\Delta u=f$ на сетке NxNxN с периодическими граничными условиями (функция на всейгранице равна 0 за исключением заданных 20 точек). Где N определяется классом теста. Функция f — кусочно-постоянная функция равная нулю по всей границе за исключением определенных 20 точек. Этот тест полезен для оценки межузловых соединений.

CG — solving an unstructured sparse linear system by the Conjugate Gradient method (решение неупорядоченной, разряженной СЛАУ методом сопряженных градиентов). Матрица СЛАУ является положительно определенной и симметричной. Метод сопряженных

градиентов используется для нахождения приближенного значения наименьшего собственного числа матрицы. В тесте используется обратный степенной метод для нахождения наибольшего собственного числа матрицы.

FT — A 3-D Fast-Fourier Transform partial differential equation benchmark — численное решение уравнения в частных производных $\delta u(x,t)/\delta t=\alpha\Delta u(x,t)$, $x\in R^3$ с использованием прямого и обратного быстрого преобразования Фурье. Этот тест включает большое количество действий, оказывающих большую нагрузку на сетевое окружение (перемещение массивов данных).

IS — Parallel Sort of small Integers. Параллельная сортировка N целых чисел. Тест не использует арифметические операции с плавающей точкой. На эффективность теста большое влияние оказывает первоначальное распределение чисел в памяти. Сортировка целых чисел является важной частью метода частиц (particle method).

Кроме ядер, пакет NPB предлагает ряд псевдо-приложений, которые эмулируют работу реальных программ по вычислительной гидродинамике. Отказаться от использования реальных приложений в пользу псевдо-приложений решено по нескольким причинам:

- сохранение настоящего исходного кода в тайне;
- облегчение работы с исходным кодом в вопросах портирования на другие архитектуры;
- легкость добавления новых компонентов;
- легкость масштабируемости кода для больших размерностей.

Алгоритмы приложений используют описанные выше ядра в том или ином виде и, в конечном итоге, сводятся к решению СЛАУ специального вида (впрочем, как и подавляющее большинство вычислительных задач). Основная масса машинного времени в таких задачах тратится именно на решение СЛАУ. Поэтому приложения можно охарактеризовать как итерационные методы решения СЛАУ. Таких приложений три: LU, SP, BT.

Алгоритмы SP (от Scalar Pentadiagonal) и BT (от Block Tridiagonal) подобны: решение трех несвязанных систем уравнений (в направлениях x, y и z) методом multi-partition scheme. Отличие приложений состоит в структуре матриц: для SP — это пятидиагональная матрица, а для BT — блочная трехдиагональная матрица с размером блока 5x5. Этот метод хорошо распараллеливается и обеспечивает оптимальную загрузку сети, но требует, чтобы количество узлов в кластере было квадратом целого числа. Тест SP более чувствителен к латентности сети.

Приложение LU решает систему уравнений с равномерной разряженной блочной структурой (5x5) методом симметричной последовательной верхней сверхрелаксации (symmetric successive over-relaxation — SSOR), к которой приводят трехмерные уравнения Навье-Стокса. Для распределения данных этому приложению требуется количество узлов, кратных степени двойки. Особенностью этого теста является его критичность ко времени передачи очень маленьких объемов данных между узлами (размер передаваемого сообщения в этом тесте составляет 40 байт).

Принадлежность к определенному классу каждого из приложений определяет размерность системы уравнений. Ниже представлена сводная таблица принадлежности ядер и приложений к определенному классу:

Тест	Класс A	Класс B	Класс C	Класс D
EP	2^{28}	2^{30}	2^{32}	2^{36}
MG	256^3	256^3	512^3	1024^3
CG	14000	75000	$1.5x10^5$	$1.5x10^6$

FT	256^2x128	256^2x512	512^3	1024^2x2048
IS	2^{23}	2^{25}	2^{27}	
LU	64^3	102^3	162^3	408^3
SP	64^3	102^3	162^3	408^3
BT	64^3	102^3	162^3	408^3

Особенности

Особенностью реализации теста является необходимость компиляции программы для каждого класса задач. Это вызвано тем, что стандарт языка fortran-77 не поддерживает динамическое распределение памяти.

Что в итоге?

В статье был дан обзор общих вопросов, связанных с тестированием массивно-параллельных вычислительных систем. Было представлено описание двух тестов: Linpack Benchmark и NAS Parallel Benchmarks. Первый из них не претендует ни на что, кроме как на определение скорости решения плотной СЛАУ прямым методом (LU факторизации). Этот тест позволяет варьировать многочисленными параметрами, с помощью которых можно настроить и «улучшить» кластер и сравнить с колоссальным количеством разнообразных систем. Второй тест, напротив, не позволяет столь вольно обращаться с параметрами расчета, а позволяет производить вычисление только в заранее определенных классах. Зато в своем составе NPB имеет большой выбор средств для выяснения узких мест в вычислительном процессе.

Литература

- [1] D. Bailey., E. Barszcz., J. Barton., D. Browning., R. Carter., L. Dagum., R. Fatoohi., S. Fineberg., P. Frederickson., T. Lasinski., R. Schreiber., H., Simon, V. Venkatakrishnan., S. Weeratunga. The NAS Parallel Benchmarks. RNR Technical Report RNR 94-007, March, 1994.
- [2] David Bailey, Tim Harris, William Saphir, Rob van der Wijngaart, Alex Woo, and Maurice Yarrow. The NAS Parallel Benchmarks 2.0. Report NAS-95-020, December, 1995.
- [3] Rob van der Wijngaart. The NAS Parallel Benchmarks 2.4. Report NAS-02-007, October, 2002.
- [4] Dongarra J. The Linpack Benchmark: An Explanation. 1988.
- [5] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitetz. The LINPACK Benchmark: Past, Present, and Future. December 2001.
- [6] J. Dongarra, V. Eijkhout, Henk van der Vost. Iterative Solver Benchmark. 14.01.2001.
- [7] J. Fernandez, J. M. Garcia. Representative benchmarks for commercial workload, September 1999.
- [8] Tau Leng, Rizwan Ali, Jenwei Hsieh, Victor Mashayekhi, Reza Rooholamini. An Empirical Study of Hyper-Threading in High Performance Computing Clusters, 2002
- [9] Jack. J. Dongarra, Performance of Various Computers Using Standart Linear Equation Software. 01.01.2004.

Приложение 1 или как это сделать?

В этом приложении постараюсь кратко описать процесс установки тестового пакета HPL. От скачивания до запуска приложения. Кажется, все просто. Но не все так очевидно, как кажется с первого взгляда.

Также представлен обзор основных параметров теста HPL и некоторые рекомендации по выбору значений параметров.

Инструментарий

Вам необходимо иметь рабочую станцию с установленной UNIX подобной операционной системой. Например, RedHat Linux 9.0. Желательно (но не обязательно), чтобы компьютер был подключен к сети, и Вы имели возможность доступа к другому компьютеру с подобной ОС. В системе должны быть установлены компиляторы языков C и Fortran.

Настройка MPI

Для компиляции и запуска HPL конкретная реализации MPI должна предоставлять три утилиты и поддерживать стандарт MPI 1.1:

mpicc/mpif77 — для компилирования MPI программ на языке c/fortran-77;

mpirun — для запуска исполняемого файла, получившегося при компиляции первыми двумя программами.

В RedHat 9.0 (и очень многих других дистрибутивах) есть реализация MPI под названием LAM — Local Area Multicomputer. Установите этот пакет. После установки станет доступна команда *lamboot*, запускающая среду LAM. Если команда *lamboot lamhost.conf* выполняется успешно, то переходите к следующему пункту. Здесь *lamhost.conf* имя файла, содержащее адреса узлов кластера. На этапе отладки можно один или несколько раз прописать *localhost*. Если команда завершилась с ошибкой, то Вам не повезло. В этой ситуации я не рекомендую использовать rpm пакеты, а зайти на официальный сайт, скачать последнюю доступную версию в исходных кодах и собрать ее у себя. Более того, как показал опыт библиотеки, поставляемые в rpm пакетах ведут себя довольно странно, и не так как хочется.

В качестве примера возьму еще одну свободно распространяемую реализацию MPI — MPICH. После скачивания и распаковки необходимо сконфигурировать пакет командой *configure --prefix=/usr/share/mpich*. Здесь параметр *prefix* указывает на каталог, в который будет установлен пакет. Процесс конфигурирования завершен. Последовательность команд *make; make install* скомпилирует пакет и установит его в указанное место. Для запуска на узлах кластера MPICH требует какой-нибудь удаленный shell. По умолчанию это *rsh* (Remote SHell). Если по каким-то причинам (безопасность, удобство) вы предпочитаете использовать *ssh* (Secure SHell), то вам стоит переконфигурировать пакет с параметром *-rsh=ssh*. В любом случае на вашей локальной машине должна работать одна из команд: *rsh localhost* или *ssh localhost*. Если ни одна из них не работает, то Вам следует настроить соответствующие сервисы. Для проверки работоспособности MPI скомпилируйте примеры в папке */examples/basic*. После этого из этой папки выполните команду *../../bin/mpirun -np 2 ./cpi*. Эта команда запустит на выполнение программу *cpi* на двух узлах кластера, которые указаны в файле *./util/machines/machines.LINUX*. Содержимое файла для отладки можно ограничить всего одной строкой — *localhost*. В дальнейшем содержимое этого файла должно определять список доступных узлов кластера.

Настройка BLAS

Как уже было указанно выше, для работы HPL в системе необходимо установить библиотеку BLAS. Так же как и в случае с MPI не стоит пользоваться версиями, поставляемыми в дистрибутиве. Можно скачать последнюю версию этой библиотеки, но лучше воспользоваться пакетом ATLAS (Automatically Tuned Linear Algebra Software). ATLAS оптимальным образом скомпилирует и настроит BLAS под вашу платформу. Если не вдаваться в детали, то настройка на удивление проста — *make*. После нескольких заданных мне вопросов, я получил код, оптимизированный под Pentium 4, с поддержкой

SSE2. Кроме Pentium 4 ATLAS знает большое количество архитектур: Opteron/Athlon 64, IA64, PIII, UltraSparc II & III, Athlon/Athlon XP.

Настройка HPL

Перед работой с тестовым пакетом HPL настоятельно рекомендуется изучить поставляемую с ним документацию.

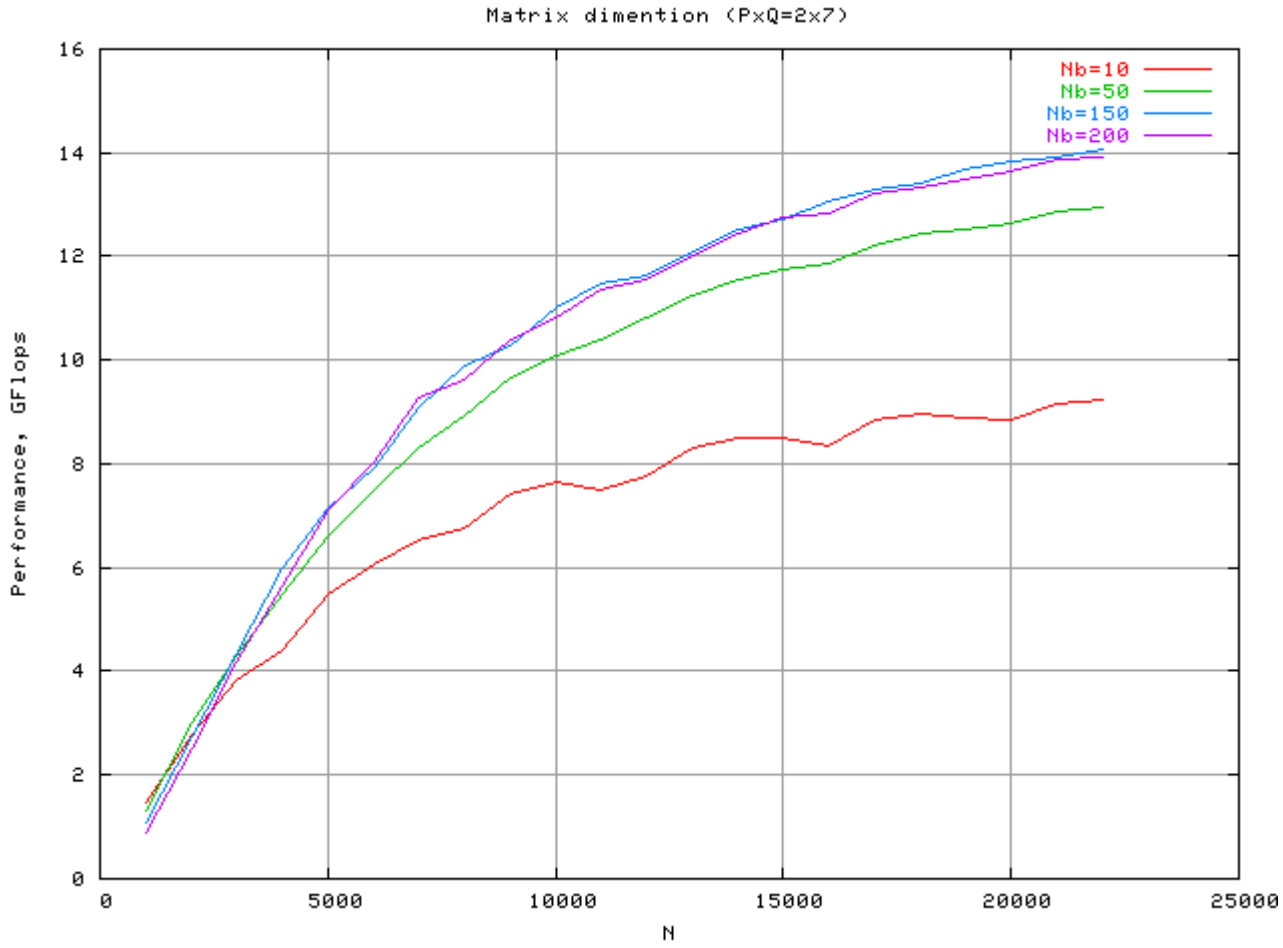
Компиляция. Для компиляции вы должны настроить make-файл. В папке setup есть несколько вариантов файла для различных архитектур (еще один пример — Make.test1 с комментариями). По их аналогии несложно составить свой файл *Make.my_arch* и скомпилировать программу: *make arch=my_arch*. Если все правильно, то в папке */bin/my_arch* появиться исполняемый файл *./xhpl* и конфигурационный файл *./HPL.dat*.

Для запуска на *N* машинах нужно выполнить: *mpirun -np N ./xhpl*. После запуска, программа читает конфигурационный файл, в котором определены все параметры алгоритма. Файл имеет очень аскетичный формат. Каждая строка определяет некоторый параметр алгоритма. Подбором соответствующих параметров можно добиться нужной производительности. Ниже дано описание основных параметров и их влияние на производительность. Основными параметрами считаются те, изменение которых сильно сказывается на производительности системы. Тестирование производилось на кластере из 7 узлов со следующими характеристиками:

- Pentium 4 3,2GHz, HT on;
- материнская плата Intel на i875P чипсете;
- 2x DDR SDRAM PC3200 512MB in dual mode;
- GigabitEthernet;
- жесткие диски отключены;
- ОС Linux 2.4.24 SMP, MPICH;
- Считается, что каждый узел является двухпроцессорной системой.

Размерность матрицы (*N*)

Наиболее сильно сказывающийся на производительности параметр. С ростом размерности наблюдается рост производительности системы, с последующим насыщением. После некоторого *N*, значительного прироста не наблюдается. Ниже приведен график зависимости мощности от размерности матрицы при разных параметрах N_b (см. ниже).



PxQ

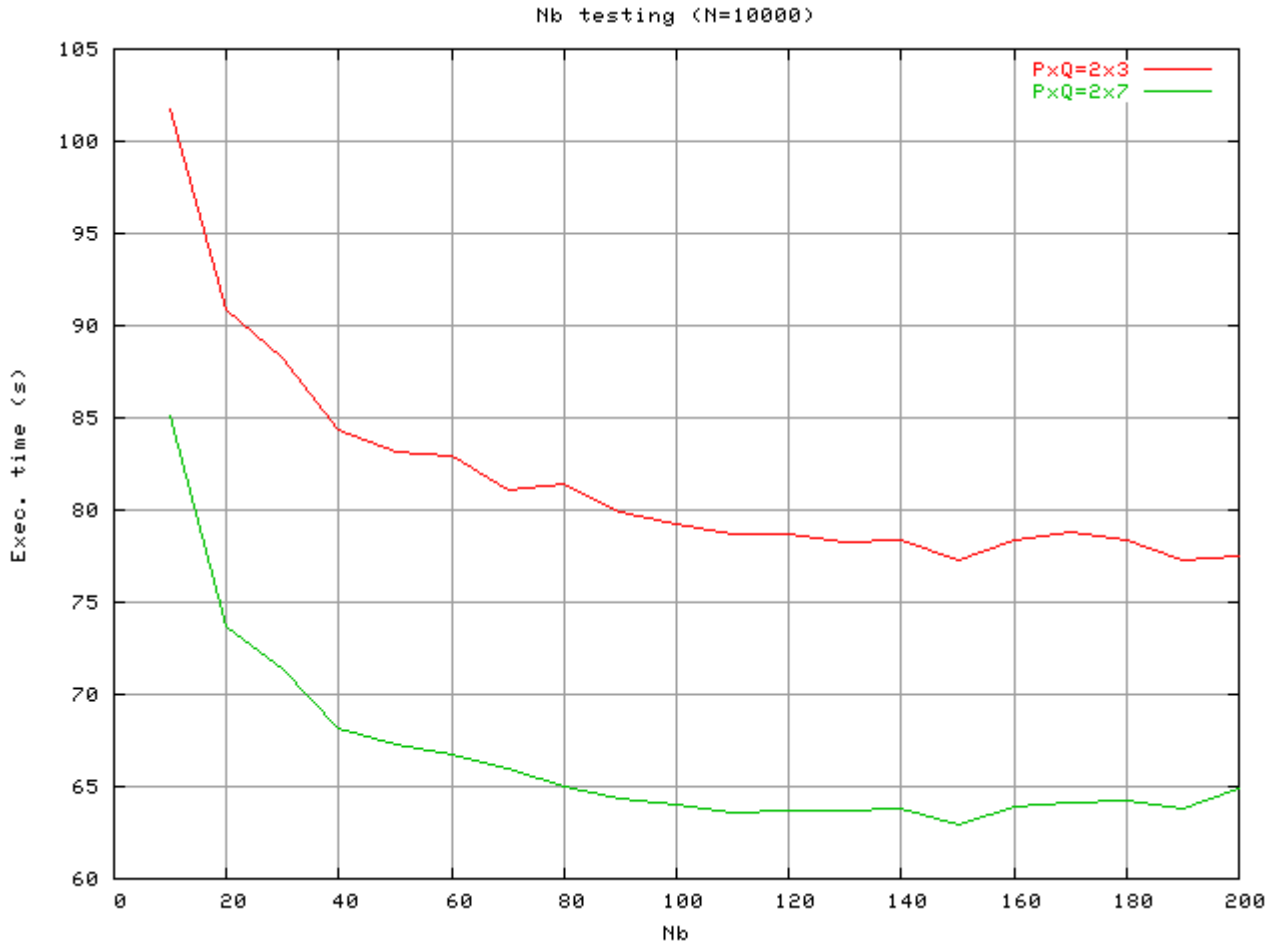
Выбор этих коэффициентов в большей степени зависит от характеристик сетевого окружения. Разработчики рекомендуют брать значение *P* несколько меньшее, чем *Q* (3x4, 4x5, 4x5 и т.п.). В случае Ethernet разница между *P* и *Q* должна быть более значительна (1x14, 2x14 и т.п.). Рекомендации подтверждаются и экспериментально.

PxQ	1x12	2x6	3x4	4x3	6x2	12x1
Время выполнения, с	12.8	14.05	14.75	15.3	18.17	28.85

Тест выполнялся на шести машинах при N=5000 и N_b=100.

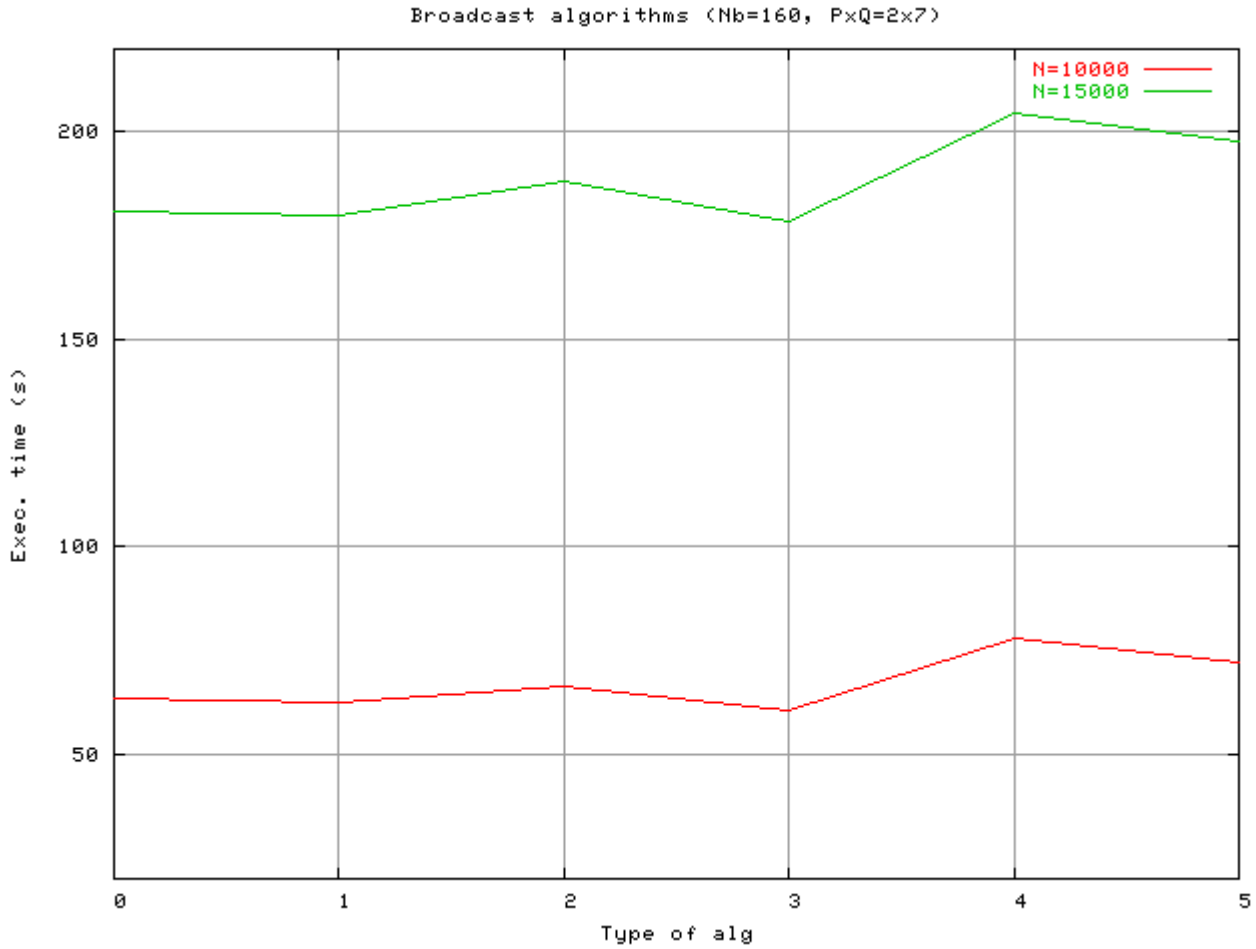
N_b

Параметр отвечает за степень распределения матрицы. Чем больше этот параметр, тем на большие куски разбивается исходная матрица. Количество пересылок данных по сети уменьшается, но при этом возрастает их интенсивность. В рассматриваемой конфигурации кластера производительность с увеличением N_b ведет себя аналогично увеличению N.



Алгоритмы распространения (broadcast algorithm)

HPL предоставляет на выбор шесть различных алгоритмов передачи данных. От простых, типа «кольцо», до достаточно сложных. Выбор конкретного алгоритма зависит от топологии и от типа сетевого окружения. На рассматриваемом кластере влияние каждого из них выглядит следующим образом (при разных значениях *N*):



Лучшим стал №3 — increasing2ring (modified).

Остальные параметры

Влияние других параметров на производительность при различных вариантах тестирования оставалась незначительной. В лучшем случаем изменения были в 0.1—0.2 GFlop.

Необходимые замечания и выводы

Представлены наиболее типичные результаты. Полученные результы не следует рассматривать как абсолютные показатели. Варианты поведения могут изменяться от системы к системе. Полученные значения кореллирут, как с рекомендациями разработчиков, так и с уже опубликованными результатами [8, 9].

Андрей Сапронов *(duan@bk.ru)*

Опубликовано — 22 марта 2004 г.