Нижегородский государственный университет им. Н.И.Лобачевского

Межфакультетская магистратура по системному и прикладному программированию для многоядерных компьютерных систем

Учебный курс «Технологии построения и использования кластерных систем»

Раздел «Оценка производительности кластерных систем»

Разработчики: А.Н. Свистунов

Нижний Новгород
2007

Содержание

Оценка производительности кластерных систем

1. Введение

В настоящий момент можно констатировать, что создание и использование многопроцессорных (параллельных) вычислительных систем является важнейшей составляющей развития компьютерной техники.

В свою очередь, кластерные вычислительные системы составляют заметную часть параллельных вычислительных систем и играют важнейшую роль в связи с некоторыми присущими им уникальными особенностями: относительно низкой стоимостью, сравнительной простотой развертывания, возможностью постепенного наращивания мощности при одновременном использовании «старого» оборудования, и т.д.

В связи с этим, все крупнейшие научные центры и ведущие промышленные предприятия либо обладают, либо имеют планы установки параллельных вычислительных систем кластерного типа.

Таким образом, вопросы, связанные с проектированием, построением и дальнейшим эффективным использованием кластерных систем являются крайне актуальными.

Данные раздел посвящен одной из важнейших проблем, которая возникает на всех этапах жизненного цикла кластерной системы – от проектирования, до поддержки продуктивной эксплуатации – проблеме тестирования кластерной системы.

1.1. Необходимость тестирования

Прежде чем перейти к рассмотрению конкретных методик тестирования и программных продуктов, эти методики реализующих, следует определиться, почему вообще возникает необходимость в тестировании кластерных систем. Представляется, что резонов для проведения тестов может быть два:

- определение работоспособности системы;
- выявление существенных характеристик (особенностей) кластерной системы.

Первый пункт в списке абсолютно понятен и вряд ли требует каких-то пояснений – кластер, как и всякая другая система, эксплуатируемая в промышленном режиме должен иметь средства самодиагностики. Такие средства как правило имеются и обычно опираются на средства диагностики, встроенные в операционную систему, сетевое оборудование и/или коммуникационное программное обеспечение. Следует отметить, что построение подобных систем является важной задачей и очень часто она решается в рамках специализированных программных комплексов, называемых *системами управления кластерами* (*Cluster Management System*, *CMS*).

В рамках этого раздела будут рассматриваться тесты второго рода – а именно выявляющие некоторые *существенные характеристики* тестируемых систем. В качестве существенных характеристик мы будем рассматривать такие характеристики, которые напрямую влияют на производительность рассматриваемых систем. Измерение этих характеристик для конкретного кластера позволяет решить сразу несколько важных задач:

- сравнить полученную вычислительную систему с другими аналогичными (что может дать некоторое усредненное понимание того, правильно настроена система или нет)

- выявить «узкие места» в тестируемой вычислительной системе, устранение которых потенциально способно повысить общую производительность;
- выбрать алгоритм решения задачи (или его готовую реализацию), позволяющий максимально эффективно «подстроиться» под имеющиеся характеристики вычислительной системы;
- определить, на этапе тестирования, достаточна мощность кластера для решения конкретной задачи в заданное время или нет, и т.д.

Кроме указанных, достаточно очевидных применений измерений, можно предложить и более сложные ситуации. Скажем, анализируя опубликованные результаты тестов различных систем и обладая информацией о показателях, которых необходимо достичь, можно на этапе проектирования попытаться определить необходимый программно-аппаратный состав будущей системы. Или, скажем, зная для разных классов задач необходимые требования к соответствующим характеристикам, заранее подобрать кластер, наиболее пригодный для них. Ну и, наконец, появляется возможность оценивать время выполнения той или иной задачи (при условии наличия оценки ее *сложности,* выраженной в единицах измеряемых тестом), что позволяет реализовывать различные варианты планирования выполнения задач на кластере.

Таким образом, результаты тестов позволяют:

- на стадии проектирования принимать решения о примерном составе будущей вычислительной системы;
- на стадии отладки и ввода в эксплуатацию путем сравнения с аналогами принимать решение о достижении необходимых характеристик;
- на стадии эксплуатации/модификации выявлять узкие места системы,

что делает тестирование необходимой составляющей всех этапов процесса работы с кластерной вычислительной системой.

1.2. Способы тестирования

В предыдущем разделе мы попытались доказать, что тестирование совершенно необходимо и его результаты явно или опосредованно используются на всех этапах жизненного цикла вычислительной кластерной системы. Ниже, приведены некоторые требования, которым должны удовлетворять тесты, для того чтобы они могли быть использованы в заявленных целях [1]:

- **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, и, соответственно, наличие компилятора под данную платформу.
- **Репрезентативность**. Вне зависимости от платформы тест должен загружать систему аналогично используемыми пользователями приложениями. Результаты сходных по структуре тестов должны быть сопоставимы.
- **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.

Все тестовые программы, удовлетворяющие указанным требованиям (а их в настоящий момент существует множество) можно классифицировать следующим образом [2]:

- «*Игрушечные*» *тесты* (*toy benchmarks*) — маленькие, длиной в несколько сот строк исходного кода. Как правило, такие тесты представляют собой решение какой-либо широко известной математической задачи – быстрая или пирамидальная сортировка, перемешивание и т.д.

- *Микротесты* (*microbenchmarks*) — специализированные, ориентированные на определение какой-то одной из основных количественных характеристик аппаратного обеспечения – среди тестируемых характеристик может быть:
  - производительность центрального процессора;
  - производительность и пропускную способности локальной оперативной памяти;
  - скорость базовых операции ввода/вывода;
  - производительность и пропускная способность коммуникационной среды.

  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. Поскольку при выполнении приложения проводят большую часть времени именно в этих фрагментах, ядра позволяют достаточно объективно 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 реальных задач. К этому же классу можно отнести и псевдоприложения. Псевдоприложения – 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.

Применительно к тестированию кластерных систем в настоящий момент традиционно преобладают тесты, относящиеся к **классу ядер** (Linpack, NAS Parallel Benchmarks), **приложений** (NAS Parallel Benchmarks) и некоторых **микротестов**, как правило, тестирующих коммуникационную составляющую (Netperf, тесты лаборатории параллельных информационных технологий НИВЦ МГУ). Поэтому, в силу своей широкой распространенности и полноты охвата, в дальнейших разделах будут рассматриваться именно эти тесты.

2. Стандартные тесты производительности

2.1. Linpack benchmark

Первым тестом, о котором пойдет речь в этом разделе, является тест Linpack. Этот тест в настоящее время является практически стандартом де-факто в тестировании вычислительных систем – по крайней мере, список 500 наиболее высокопроизводительных систем мира (www.top500.org) составляется именно по результатам этого теста.

2.1.1. История теста

Тест Linpack, впервые был опубликован в 1979 г. и первоначально являлся дополнением к одноименной библиотеке численных методов, содержащей набор процедур для решения систем линейных алгебраических уравнений (СЛАУ) и предназначался для оценки времени решения той или иной системы с помощью этой библиотеки. Linpack является классическим примером теста-ядра (причем, поскольку к решению тех или иных СЛАУ сводятся очень многие реальные расчетные задачи – измеренные им характеристики являются в высокой степени репрезентативными).

Автором теста является Джек Донгарра (J. Jack Dongarra) из Университета штата Теннесси (до этого он сотрудничал с Аргоннской национальной лабораторией, где и была сформирована концепция тестов Linpack).

Тест состоит в решении системы линейных арифметических уравнений вида

$$Ax = f$$

методом LU-факторизации с выбором ведущего элемента столбца, где A – плотно заполненная матрица размерности N (первоначальный, «классический» вариант Linpack решал задачу размерности 100). Производительность в тесте Linpack измеряется в количестве производимых операций с плавающей запятой в секунду. Единицей измерения является *1 флопс*, то есть одна такая операция в секунду. Поскольку количество операций, необходимое для решения задачи Linpack, известно с самого начала и зависит от ее размерности, измеряемая характеристика производительности получается как простое частное от деления этого известного числа операций на время, затраченное на решение задачи.

С течением времени и увеличении вычислительной мощности компьютеров, размерность теста Linpack была увеличена до 1000. Однако с появлением все более мощных вычислительных систем и эта размерность стала чересчур малой, более того, для тестирования кластерных систем была создана отдельная версия теста (доступная на сайте http://www.netlib.org/benchmark/hpl/) в которой размерность матрицы (и некоторые другие параметры) не являются фиксированными, а задаются пользователем теста.

Дело в том, что при увеличении размерности матрицы решаемой задачи, растет степень параллелизма, что может привести к увеличению производительности. Другим важным параметром, влияющим на производительность, является размер блока, с которым матрица распределяется между узлами кластерной системы. Особую значимость этот параметр имеет для компьютеров с векторной архитектурой (в этом случае он характеризует длину обрабатываемых векторов), и было бы большой ошибкой не считаться с ним при тестировании систем других классов. Дело в том, что практически все современные компьютеры широко используют средства параллельной обработки (конвейеризованная и/или суперскалярная арифметика, MPP-организация системы и т. д.), поэтому оценка производительности при разной глубине программного параллелизма весьма показательна для любой современной системы. Таким образом, в отличие от классического теста Linpack, Linpack в его «кластерном»

варианте требует *подбора* указанных (и ряда других) параметров с целью достижения максимальной производительности.

Возвращаясь к истории, следует отметить, что первоначально тест был написан на языке Fortran (и сейчас часто используется эта версия теста), однако для тестирования кластерных систем существует версия на языке C. Основное время теста (свыше 75% времени выполнения) занимает внутренний цикл, выполняющий типичную для действий с матрицами операцию

$$y(i) = y(i) + a \cdot x(i),$$

представленный тестовой процедурой SAXPY/DAXPY. Поскольку тест LINPACK достаточно хорошо векторизуется и распараллеливается на большинстве систем, имеет смысл компилировать его компиляторами поддерживающими векторизацию.

Как уже говорилось ранее, для тестирования кластерных систем используется версия теста, называемая HPL (*High-Performance Linpack Benchmark*, http://www.netlib.org/benchmark/hpl/). В этой версии пользователь имеет возможность задать все значимые параметры алгоритма, подбирая их для достижения наилучшей производительности.

2.1.2. Решаемая задача

Документация по алгоритму, используемому в Linpack, доступна по адресу http://www.netlib.org/benchmark/hpl/algorithm.html. Как уже упоминалось, тест состоит в решении системы линейных алгебраических уравнений методом LU-факторизации с выбором ведущего элемента столбца.

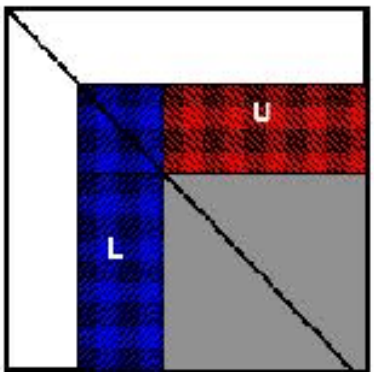


Рис. 1. Схема решения системы линейных уравнений в тесте Linpack

При параллельном процессе на вычислительном кластере исходная матрица разделяется на логические блоки размерностью *NB*×*NB* (*NB* – параметр алгоритма, задаваемый пользователем, обычно при расчетах лежит в интервале от 32 — 256). Эти блоки в свою очередь разбиваются сеткой *P*×*Q* на более мелкие. Каждый из таких блоков «достанется» отдельному процессору системы.

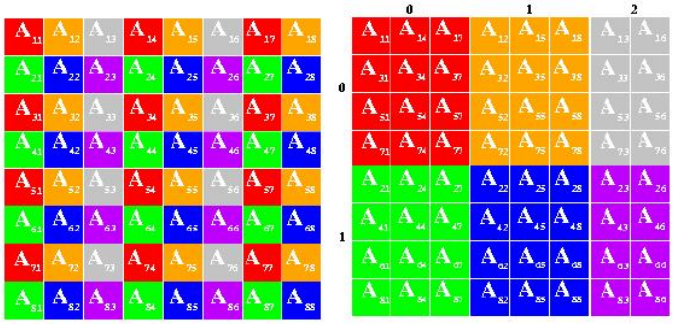
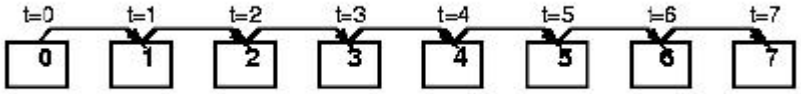


Рис. 2. Схема распределения данных при решении системы линейных уравнений в тесте Linpack

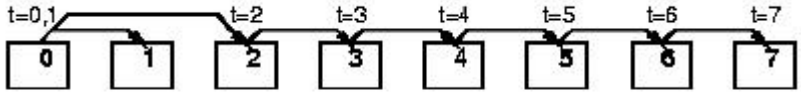
Коэффициенты P и Q берутся в зависимости от структуры кластера, а их произведение не может быть больше доступного числа узлов. Если в кластере 8 узлов, то допустимыми значениями $P \times Q$ будут: 1×8, 2×4, 3×2, 2×2, 1×4…. При этом в расчетах будут участвовать $P \times Q$ процессоров. Именно процессоров, а не узлов (что важно, при использовании в кластере SMP-узлов). Конкретные значения P и Q следует выбирать в зависимости от структуры кластера и коммуникационной среды.

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

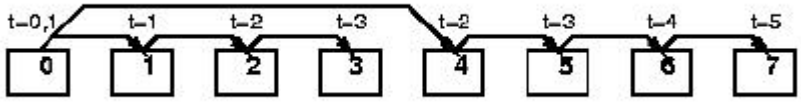
Increasing-ring: Данные пересылаются последовательно $0 \rightarrow 1$; $1 \rightarrow 2$; $2 \rightarrow 3$ и так далее. Этот алгоритм один из классических:



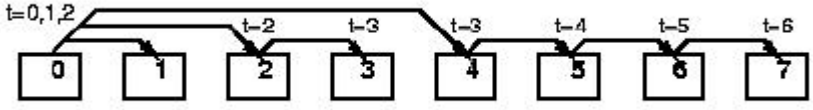
Increasing-ring (modified): Данные пересылаются в соответствии с правилом $0 \rightarrow 1$; $0 \rightarrow 2$; $2 \rightarrow 3$ и так далее. Процесс с номером 0 отправляет 2 сообщения и процесс 1 принимает только одно сообщение. Этот алгоритм лучше предыдущего, но не является самым лучшим:



Increasing-2-ring: Q процессов подразделяются на две части: $0 \rightarrow 1$ и $0 \rightarrow Q/2$. Далее 1 и $Q/2$ процессы действуют как источники двух сообщений $1 \rightarrow 2$, $Q/2 \rightarrow Q/2+1$; $2 \rightarrow 3$, $Q/2+1 \rightarrow Q/2+2$ и так далее:



Increasing-2-ring (modified): Данные пересылаются по схеме: $0 \rightarrow 1$; $0 \rightarrow 2$; $2 \rightarrow 3$ ($0 \rightarrow 4$); $2 \rightarrow 3$ ($0 \rightarrow 4$) и так далее:



Long (bandwidth reducing): В отличие от предыдущих вариантов, требует синхронизации всех процессов, участвующих в вычислении, для выполнения операции обмена. Передаваемый блок данных разделяется на Q частей, после чего «разбрасывается» по всем процессам, причем обмен разными частями сообщения происходит в разные моменты времени (фаза распределения). После этого, происходят попарные обмены полученными частями сообщений между процессорами (фаза обмена):

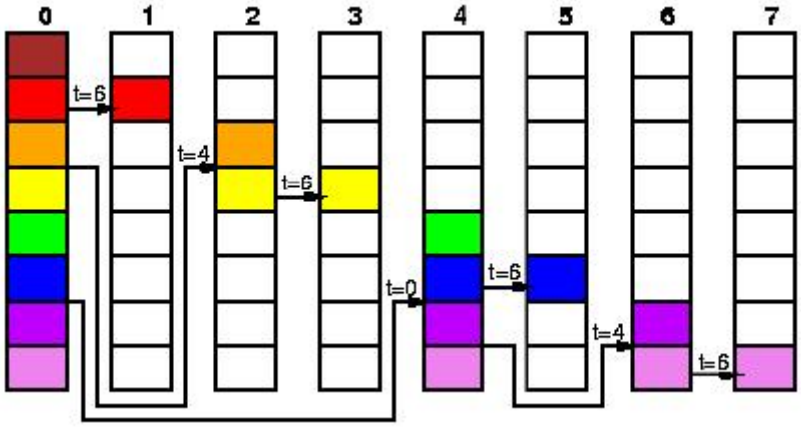


Рис. 3. Фаза распределения в схеме Long алгоритма рассылки данных

Фаза распределения использует бинарные деревья, фаза обмена– только взаимные обмены сообщениями:

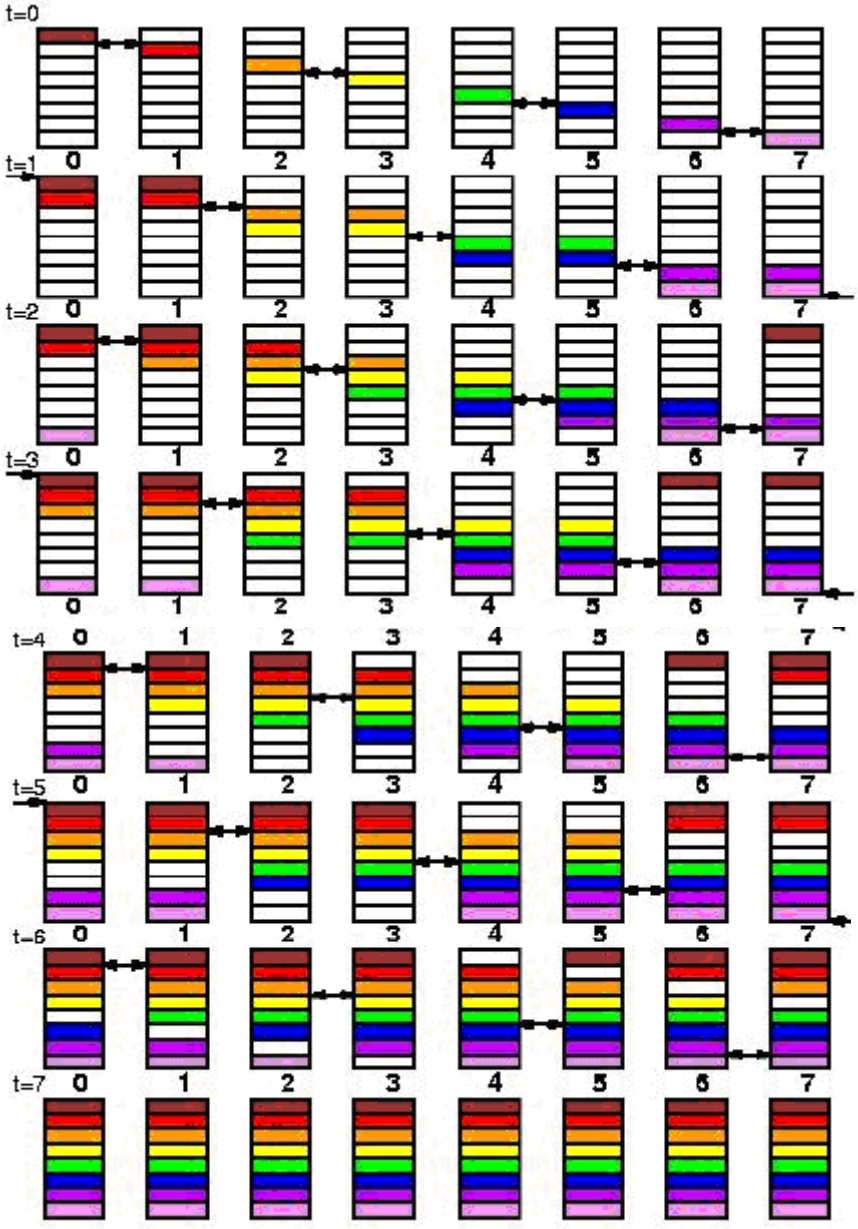


Рис. 4. Фаза обмена в схеме Long алгоритма рассылки данных

Long (bandwidth reducing modified): похож на выше изложенный вариант, исключая первый шаг от 0 → 1, затем используется предыдущий Long вариант 0, 2, 3, 4 … *Q*-1.

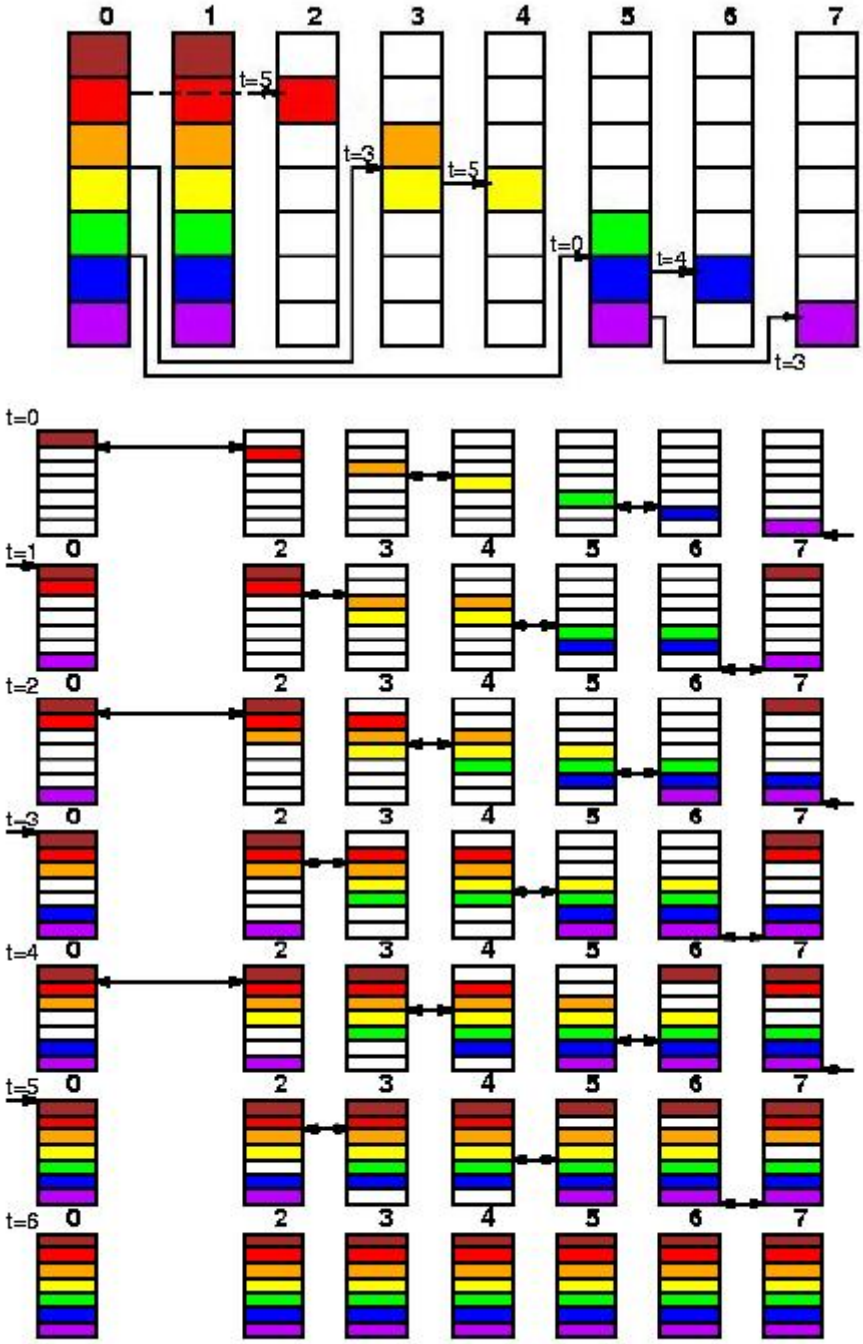


Рис. 5. Схема обменов данными для алгоритма Long (bandwidth reducing modified)

После того, как разложение закончено, последовательно решается две системы уравнений:

$$Ly = f, \quad Ux = y$$

Задача считается успешно решенной, а тест считается выполненным, если выполнены следующие условия:

$$\|Ax-b\|_\infty /(\varepsilon * \|A\|_1 * N) < \alpha$$

$$\|Ax-b\|_\infty /(\varepsilon * \|A\|_1 * \|x\|_1) < \alpha$$

$$\|Ax-b\|_\infty /(\varepsilon * \|A\|_\infty * \|x\|_\infty) < \alpha$$

где: - ε — точность представления чисел с плавающей точкой,

- α — константа, задаваемая в конфигурационном файле,
- $\|A\|_1 = \max_j \sum_i a_{ij}$,
- $\|A\|_\infty = \max_i \sum_j a_{ij}$,
- $\|x\|_1 = \sum_i x_i$,
- $\|x\|_\infty = \max_i x_i$.

2.1.3. Использование теста

Для того чтобы воспользоваться тестом необходимо загрузить его дистрибутив (включающий в себя исходный код и makefile-ы, предназначенные для компиляции под различные платформы). Для компиляции также потребуется наличие какой-либо реализации MPI, а также любая реализация библиотеки BLAS (Basic Linear Algebra Subroutines) или библиотеки VSIPL (Vector Signal Image Processing Library).

После сборки получившийся исполняемый модуль использует файл HPL.dat, в котором могут быть указаны существенные параметры алгоритма. Для облегчения проведения тестирования в файле HPL.dat может быть задана последовательность параметров, при этом будет выполнена серия тестов со всеми перечисленными значениями параметров.

Ниже приведен пример конфигурационного файла HPL.dat для теста Linpack, строки которого для удобства пронумерованы

```
1.      HPLinpack benchmark input file
2.      Innovative Computing Laboratory, University of Tennessee
3.      HPL.out      output file name (if any)
4.      0            device out (6=stdout,7=stderr,file)
5.      3             # of problems sizes (N)
6.      1000 2000 3000         Ns
7.      2            # of NBs
8.      112 120 128  NBs
9.      0            PMAP process mapping (0=Row-,1=Column-major)
10.     4            # of process grids (P x Q)
11.     1 2 1 4      Ps
12.     1 2 4 1      Qs
13.     16.0         threshold
14.     1            # of panel fact
15.     0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
16.     2            # of recursive stopping criterium
17.     4 2          NBMINs (>= 1)
18.     1            # of panels in recursion
19.     2            NDIVs
20.     1            # of recursive panel fact.
```

```
21.     1 0 2        RFACTs (0=left, 1=Crout, 2=Right)
22.     1            # of broadcast
23.     0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
24.     1            # of lookahead depth
25.     0            DEPTHs (>=0)
26.     2            SWAP (0=bin-exch,1=long,2=mix)
27.     256          swapping threshold
28.     1            L1 in (0=transposed,1=no-transposed) form
29.     1            U  in (0=transposed,1=no-transposed) form
30.     0            Equilibration (0=no,1=yes)
31.     8            memory alignment in double (> 0)
```

В приведенном файле строки 1,2 служат для идентификации файла и более никакой роли не играют (нужно заметить, что они переносятся в файл результата HPL.out). Строки 3,4 определяют, каким образом будет осуществляться вывод результатов теста. Строки 5,6 содержат перечисление размерностей задач, которые будут решаться в ходе теста. Строки 7,8 определяют различные варианты параметра *NB*. Следует отметить, что тест устроен таким образом, что перебирает всевозможные варианты заданных параметров, таким образом для трех различных размерностей и двух вариантов *NB* тест будет выполнен шесть раз. Поскольку число различных параметров в конфигурационном файле велико, следует быть осторожным и не забывать об этой особенности. Строки 10,11,12 определяют различные варианты сетки $P \times Q$, для которых будет выполнен тест. Строка 13 задает константу α. Остальные строки (14-31) задают другие параметры алгоритма, которые в данном разделе не рассматривались (обратим лишь внимание на строку 23, задающую один из вариантов алгоритма обмена, о которых речь шла выше)

Результатом работы теста является достаточно объемный файл, в котором для каждого набора параметров, определенном в конфигурационном файле указана достигнутая производительность на тесте, а также имеющаяся погрешность решения.

Ниже приведен фрагмент этого файла (в приведенном примере запуск был осуществлен на одном узле):

```
================================================================================
HPLinpack 1.0a  --  High-Performance Linpack benchmark  --   January 20, 2004
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Labs.,  UTK
================================================================================

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :    1000     2000     3000
NB     :     112      120
PMAP   : Row-major process mapping
P      :       1        2        1        4
Q      :       1        2        4        1
```

```
PFACT  :    Left
NBMIN  :       4        2
NDIV   :       2
RFACT  :   Crout
BCAST  :   1ring
DEPTH  :       0
SWAP   : Mix (threshold = 256)
L1     : no-transposed form
U      : no-transposed form
EQUIL  : no
ALIGN  : 8 double precision words

----------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual checks will be computed:
   1) ||Ax-b||_oo / ( eps * ||A||_1  * N        )
   2) ||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  )
   3) ||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo )
- The relative machine precision (eps) is taken to be         1.110223e-016
- Computational tests pass if scaled residuals are less than          16.0

============================================================================
T/V                N    NB     P     Q               Time             Gflops
----------------------------------------------------------------------------
W00C2L4         1000   112     1     1               0.99         6.731e-001
----------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N        ) =        1.4543523 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0352991 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0085280 ...... PASSED
============================================================================
T/V                N    NB     P     Q               Time             Gflops
----------------------------------------------------------------------------
W00C2L2         1000   112     1     1               0.79         8.467e-001
----------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N        ) =        1.3432175 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0326017 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0078763 ...... PASSED
============================================================================
```

Первые строки вывода содержат «легенду» с указанием расшифровки используемых сокращений. Затем следует секция с указанием значений диапазонов параметров, используемых при данном запуске. И, наконец, следует секция с результатами тестирования. В этой секции для каждого теста указываются параметры теста, достигнутая производительность (в гигафлопах), а также результат проверки корректности полученного решения.

2.2. NAS Parallel Benchmarks

Набор тестов NAS Parallel Benchmarks (NPB) является, пожалуй, не менее распространенным, чем тест Linpack. Текущую реализацию, выполненную специалистами NASA Ames Research Center, можно загрузить с сайта http://www.nas.nasa.gov/Software/NPB (на момент написания данного материала была доступна версия 3.3).

2.2.1. История теста

В отличие от теста Linpack, первые версии которого были разработаны в конце 70-х годов, комплекс тестов NPB появился сравнительно недавно. Он был разработан в начале 90-х в исследовательском центре NASA Ames Research Center в рамках программы Numerical Aerodynamic Simulation (NAS), целью которой было обеспечение возможности проведения за несколько часов полномасштабного численного моделирования полета космического аппарата. NPB должен был стать универсальным средством для оценки производительности суперкомпьютеров. Поскольку в рамках проекта NAS создавалось вполне определенное программное обеспечение для решения известных задач гидро- и аэродинамического моделирования, тестовый комплекс включает в себя ядра для наиболее распространенных задач этих классов.

Несмотря на полную определенность с содержащимися в тестовом комплексе ядрами, NPB, по замыслу создателей является «*paper and pencil*» тестом, т.е. официально NPB лишь набор правил и рекомендаций, доступных «на бумаге» (таким образом, по приведенной выше ссылке доступна, вообще говоря, всего лишь одна из реализаций этого комплекса, выполненная самими исследователями из NASA Ames Research Center).

Правила NPB достаточно жесткие и определяют практически все вопросы, которые могут возникнуть в процессе разработки конкретной реализации:

- определены допустимые языки программирования;
- приведено исчерпывающее описание всех алгоритмов (ядер, приложений, генерации случайных чисел…);
- оговорены разрядности чисел с плавающей запятой;
- определены разрешения/запрещения для распараллеливания некоторых видов алгоритмов (например, поиск минимума в векторе);
- есть договоренности об организации ввода/вывода;
- регламентированы моменты замеров времени и, следовательно, включаемые в тест операции;
- определены правила публикации результатов тестов;
- сформулированы критерии оценки правильности полученных результатов;

и многое, многое другое.

Как уже упоминалось, на сервере NASA доступны готовые реализации теста. До версии 2.0 программа позиционировалась как пример реализации, чтобы упростить создание своей версии конечными пользователями. После версии 2.0 комплекс представляет собой по сути законченный продукт, настраиваемый для использования как с различными аппаратными платформами/операционными системами, так и с различными технологиями параллельного программирования - OpenMP/MPI. Для получения исходных кодов теста необходимо пройти регистрацию на сайте.

2.2.2. Решаемая задача

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 гидро- и аэродинамики). В

терминологии NPB ядра и приложения могут производить вычисления в определенных классах задач (*Problem Classes*):

- Sample code,
- Class A,
- Class B,
- Class C,
- Class D,
- Class E.

Важно еще раз отметить, что все классы задач содержат один и тот же набор тестов, и отличие классов между собой состоит лишь в объеме обрабатываемых в тестах данных. Другими словами, класс A — это маленькие матрицы, B — большие, C — очень большие, D — огромные, E — гигантские (класс E введен сравнительно недавно и определен не для всех тестов). Например, для теста на LU-разложение это будет размерность исходной матрицы: 12^3, 64^3, 102^3, 162^3, 408^3, 1020^3 для каждого из перечисленных выше классов соответственно.

Таблица 1. Размерности задач, решаемых тестами для различных классов

Тест	Класс A	Класс B	Класс C	Класс D	Класс E
EP	2^{28}	2^{30}	2^{32}	2^{36}	2^{40}
MG	256^3	256^3	512^3	1024^3	2048^3
CG	14000	75000	$1.5x10^5$	$1.5x10^6$	$9x10^6$
FT	256^2x128	256^2x512	512^3	1024^2x2048	$4096x2048^2$
IS	2^{23}	2^{25}	2^{27}	2^{29}	
LU	64^3	102^3	162^3	408^3	1020^3
SP	64^3	102^3	162^3	408^3	1020^3
BT	64^3	102^3	162^3	408^3	1020^3
DT[1]					

В настоящий момент в NPB включено 5 ядер: EP, MG, CG, FT, IS:

- **EP** — *Embarrassing Parallel.* Вычисление интеграла методом Монте-Карло - тест "усложненного параллелизма" для измерения первичной вычислительной производительности плавающей арифметики. Этот тест минимального межпроцессорного взаимодействия и фактически определяет "чисто" вычислительные характеристики узла при работе с вещественной арифметикой. Межпроцессорные 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 ("трехмерная решетка") в частных производных на сетке *N*x*N*x*N* с периодическими граничными условиями (функция на всей границе равна 0 за исключением заданных 20 точек). Размер сетки *N* определяется классом теста. Тестирует возможности системы выполнять как длинные, так и короткие передачи данных.
- **CG** — *solving an unstructured sparse linear system by the Conjugate Gradient method*. Вычисление наименьшего собственного значения больших, разреженных матриц методом сопряженных градиентов. Матрица является положительно

[1] DT (Data Traffic) – новый тест, тестирующий коммуникационную составляющую. Размерность графа, который строится, зависит не только от класса задачи, но и от способа обмена. Тест входит только в версию NPB-MPI.

определенной и симметричной. В тесте используется обратный степенной метод для нахождения наибольшего собственного числа матрицы. Тест применяется для оценки скорости передачи данных при отсутствии какой-либо регулярности.

- **FT** —*3-D Fast-Fourier Transform partial differential equation benchmark.* Вычисление методом быстрого преобразования Фурье трехмерного уравнения в частных производных. Этот тест включает большое количество действий, оказывающих большую нагрузку на сетевое окружение (перемещение массивов данных). При создании программы, реализующей данный тест, могут использоваться библиотечные модули преобразования Фурье различной размерности.
- **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 систем линейных алгебраических уравнений (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.

- **LU** — *LU Solver.* Тест выполняет решение системы уравнений с равномерно разреженной блочной треугольной матрицей 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).
- **SP** — *Scalar Pentadiagonal.* Тест выполняет решение нескольких независимых систем скалярных уравнений - пентадиагональные матрицы с преобладанием недиагональных членов.
- **BT** — *Block Tridiagonal*. Решение серии независимых систем уравнений - блочные трехдиагональные матрицы с преобладанием недиагональных элементов.

2.2.3. Использование теста

Для того, чтобы воспользоваться тестом, необходимо загрузить его дистрибутив - он включает в себя три версии теста – «стандартную», версию для технологии OpenMP и версию для технологии MPI. Все три версии содержат описанные выше ядра и псевдо-приложения. Отличаются версии наличием специализированных тестов для обмена данными в версиях OpenMP и MPI. Загруженные дистрибутивы содержат исходный код и makefile-ы, предназначенные для компиляции под различные платформы. Для компиляции также потребуется наличие какой-либо реализации MPI (в

случае использования MPI-версии), или компилятор, поддерживающий технологию OpenMP (в случае использование OpenMP-версии).

Ниже приведен вывод теста **IS** для случая запуска на одном процессоре, для класса задачи A.

```
NAS Parallel Benchmarks 3.3 -- IS Benchmark

Size:  8388608  (class A)
Iterations:   10
Number of processes:     1

  iteration
       1
       2
       3
       4
       5
       6
       7
       8
       9
       10


IS Benchmark Completed
Class           =                        A
Size            =                  8388608
Iterations      =                       10
Time in seconds =                     7.41
Total processes =                        1
Compiled procs  =                        1
Mop/s total     =                    11.33
Mop/s/process   =                    11.33
Operation type  =              keys ranked
Verification    =               SUCCESSFUL
Version         =                      3.3
Compile date    =              27 Jan 2008

Compile options:
   MPICC        = cc
   CLINK        = $(MPICC)
   CMPI_LIB     = -L/usr/local/lib -lmpi
   CMPI_INC     = -I/usr/local/include
   CFLAGS       = -O
   CLINKFLAGS   = -O


Please send the results of this run to:

NPB Development Team
Internet: npb@nas.nasa.gov

If email is not available, send this to:
```

```
MS T27A-1
NASA Ames Research Center
Moffett Field, CA  94035-1000

Fax: 650-604-3957
```

Первая секция полученного вывода содержит указание на название исполняемого теста (в данном случае - IS), размерность задачи и количество используемых тестом процессов. Следующая секция служит для отображения хода выполнения теста в процессе его работы и отображает текущую выполняемую тестом итерацию (в данном примере было выполнено 10 итераций теста). И, наконец, в последней секции содержится информация о результатах выполнения теста – классе задаче, достигнутой на данном классе производительности (в сумме и в перерасчете на каждый из процессов), дате компиляции и т.д. Кроме этого, указываются важнейшие параметры построения теста, такие как компилятор, версия библиотеки MPI и т.д.

2.3. Тесты НИВЦ МГУ

Важнейшей характеристикой кластерной вычислительной системы, наряду с характеристиками производительности составляющих ее вычислительных узлов, является «производительность» коммуникационной среды, их объединяющей. Поскольку большинство практических задач, требующих для своего решения использования мощных кластеров, использует алгоритмы, предполагающие интенсивные обмены данными, очень часто именно скорость этих обменов является фактором, определяющим производительность всей кластерной системы в целом. Поэтому тестирование производительности коммуникационной среды занимает важное место в общем тестировании и в настоящий момент разработано большое количество тестовых пакетов, такое тестирование осуществляющих. Одним из таких пакетов является набор тестов, разработанных в лаборатории параллельных информационных технологий НИВЦ МГУ (http://www.srcc.msu.su/). Следует отметить, что этот тестовый пакет является высокоуровневым в том смысле, что тестирует производительность операций передачи данных с использованием какой-либо реализации MPI. Таким образом, на результаты тестирования оказывают влияние как собственно физические характеристики среды передачи и параметры всего используемого сетевого оборудования (от сетевых адаптеров до маршрутизаторов), так и настройки всего программного стека сетевых протоколов, а также настройки используемой реализации MPI.

2.3.1. История теста

Первые доступные версии теста появились в 1997 г. Текущая версия была опубликована в 2003 г. Пакет является свободно распространяемым, и в виде исходных текстов программ на C доступен для загрузки по адресу http://parallel.ru/ftp/tests/.

2.3.2. Решаемая задача

Пакет включает в себя четыре теста:

- transfer — тест латентности и скорости пересылок между двумя узлами,
- nettest — тест пропускной способности сети при сложных обменах по различным логическим топологиям,
- mpitest — тест эффективности основных операций MPI,
- nfstest — тест производительности файл-сервера.

Тест **transfer** измеряет основные характеристики быстродействия сети - латентность (*latency*) и пропускную способность (*bandwidth*).

Пропускной способностью сети называется *количество информации*, передаваемой между узлами сети в единицу времени (байт в секунду). *Латентностью* (задержкой) называется время, затрачиваемое программным обеспечением и устройствами сети на подготовку к передаче информации по данному каналу. Полная латентность складывается из программной и аппаратной составляющих.

Тест измеряет пропускную способность однонаправленных пересылок *("точка-точка", uni-directional bandwidth*), и пропускную способность двунаправленных пересылок (*bi-directional bandwidth*), используя для этого различные функции MPI [8].

Тест **nettest** является тестом коммуникационной производительности при сложных обменах между несколькими узлами в различных логических топологиях ("звезда", "полный граф", "кольцо"). Тест может использоваться для проверки того, как ведет себя сетевое оборудование при пиковых нагрузках, а также для определения пиковой пропускной способности коммутаторов.

Тест **mpitest** предназначен для тестирования производительности базовых функций MPI. Может быть использован для сравнения различных реализаций MPI, а

также для предсказания производительности приложений, использующих соответствующие операции.

Тест **nfstest** предназначен для тестирования производительности общей файловой системы. Хотя этот тест не является тестом производительности MPI, он использует некоторые возможности MPI, такие как синхронизация (*MPI_Barrier*) и пересылка данных от головного всем процессам (*MPI_Bcast*). Данный тест также может использоваться для проверки корректности функционирования файл-сервера при больших нагрузках.

2.3.3. Использование теста

Как уже упоминалось выше, тестовый пакет представляет собой набор исходных текстов программ, реализованных на языке C и makefile для их компиляции, доступный по адресу http://parallel.ru/ftp/tests/. Потребуется также какая-либо реализация MPI.

Ниже представлен результат работы теста **transfer,** запущенного в режиме *localonly* для четырех процессов, с параметрами по умолчанию.

Измерены латентность и пропускная способность для блокирующих операций пересылки/приема данных:

```
--- MPI Performance Test Suite
--- Moscow State University 1998-2003


Running MPI Transfer/2 test, 4 processes
messages: 0 to 65536, step 1024, unit is 1 bytes;
 20 times (fix 0)
Process 0 of 4 on host1
Process 1 of 4 on host1
Process 2 of 4 on host1
Process 3 of 4 on host1


Running test: Uni-directional MPI transfer: Blocking Send/Recv (1)
Testing transfer between 0 and other processes

Checking correctness of message passing

Size(b) Transfer (MB/sec)


Iteration 0
[0 -- 1] Latency: 142.595 microseconds (at 20 times)
[0 -- 2] Latency: 150.843 microseconds (at 20 times)
[0 -- 3] Latency: 151.053 microseconds (at 20 times)

1024    5.668  4.47   5.681
2048    9.194  8.991  8.983
3072    11.59  11.28  11.49
4096    13.84  13.22  13.1
5120    12.75  13.88  14.05
6144    13.87  14.37  14.61
7168    14.56  16.12  15.85
8192    16.49  16.82  17.21
```

9216	15.97	17.28	17.36
10240	17.62	17.43	17.15
11264	18.85	18.24	19.26
12288	16.81	18.36	19.44
13312	18.76	18.66	18.98
14336	20.22	19.8	20.12
15360	20.09	19.6	20.6
16384	20.67	20.22	20.59
17408	21.21	20.6	21.3
18432	20.23	20.01	21.01
19456	21.27	20.41	20.84
20480	19.61	20.92	21.14
21504	21.88	20.8	21.42
22528	20.45	21.04	19.75
23552	21.58	21.01	20.92
24576	23.01	22.69	22.54
25600	22.67	22.47	21.48
26624	22.61	21.15	21.18
27648	21.35	21.86	21.64
28672	22.6	22.13	21.92
29696	23.21	19.38	23.58
30720	22.43	21.17	22.4
31744	22.39	22.02	23.11
32768	23.77	22.69	23.08
33792	23.95	22.64	22.04
34816	24.33	24.03	24.05
35840	22.91	22.25	22.85
36864	24.05	23.06	22.75
37888	24.18	23.7	24.77
38912	24.36	22.6	24.1
39936	23.93	22.49	22.25
40960	23.45	23.23	22.3
41984	21.9	22.61	23.87
43008	24.84	24.54	23.68
44032	24.49	24.92	24.18
45056	25.69	24.55	23.63
46080	26.13	23.34	24.08
47104	26.27	25.85	25.29
48128	25.93	24.26	25.76
49152	25.6	23.75	24.95
50176	26.48	24.67	24.04
51200	26.46	25.4	26.95
52224	26.87	25.89	26.93
53248	25.8	26.52	25.28
54272	26.38	26.62	26.42
55296	26.6	26.14	25.94
56320	25.3	25.98	25.95
57344	25.13	26.72	25.09
58368	26.31	25.73	25.12
59392	24.88	26.5	24.03
60416	25.28	26.28	27.11
61440	25.5	26.16	26.07
62464	24.87	27.05	26.34
63488	25.86	26.91	26.69

```
64512   26.09  26.46  25.47
65536   24.8   26.43  26.96

MPI Transfer test complete in 11.7066 seconds
```

Файл результата содержит три секции. В первой секции указываются параметры теста – для каких размерностей и с каким шагом тест запущен, на скольких процессах и какая операция тестом тестируется. Во второй секции приводятся измеренные данные для латентности тестируемой операции, при этом измерения проводятся для взаимодействий нулевого процесса со всеми процессами. Наконец, третья секция представляет собой таблицу, в которой для каждого размера сообщения измеряется скорость его передачи от нулевого процесса к каждому из процессов, участвующих в тестировании.

3. Литература

1. J. Fernandez, J. M. Garcia. Representative benchmarks for commercial workload, September 1999
2. http://www.ixbt.com/cpu/cluster-benchtheory.shtml
3. http://www.netlib.org/benchmark/hpl/
4. The LINPACK Benchmark: Past, Present, and Future. Jack J. Dongarra,Piotr Luszczek, Antoine Petitet. July 2002
5. Performance of Various Computers Using Standard Linear Equations Software. Jack J. Dongarra. Electrical Engineering and Computer Science Department University of Tennessee CS - 89 – 85 September 9, 2007
6. Rob F. Van der Wijngaart. “NAS Parallel Benchmarks Version 2.4”, NAS Technical Report NAS-02-007. October 2002
7. http://www.osp.ru/os/1995/06/178756/
8. http://parallel.ru