Т-система—среда программирования с поддержкой автоматического динамического распараллеливания программ[1]

© 1998 г. С.М. Абрамов, А.И. Адамович
Институт программных систем РАН
152140 Переславль-Залесский

Статья посвящена описанию Т-системы—среды программирования, поддерживающей автоматическое динамическое распараллеливание программ. Статья открывается описанием базовой вычислительной модели Т-системы. Описываются основные понятия модели организации вычислений в Т-системе—процесс, сетевой вызов функции, отношение “поставщик-потребитель”, неготовое значение,—а также операции с ними. Далее описываются программные решения, использованные при реализации Т-системы на платформе “IP-сеть Unix-компьютеров”. Рассматриваются основные функции Т-системы и реализующие их компоненты Т-системы. В следущем разделе обсуждается программирование для Т-системы.

Введение

Широкое распространение различных параллельных аппаратных платформ сделало очевидным тот факт, что на сегодняшний день проблема реализации параллельных вычислений в большей мере является проблемой программного обеспечения, а не аппаратных средств.

Самый распространенный подход к разработке параллельных программ, основанный на использовании программных пакетов типа PVM или MPI и ручном статическом распараллеливании, требует от разработчика повышенного объема специальных знаний и навыков и значительных затрат времени как на этапе разработки программ, так и при их отладке. Автоматизация этого подхода с использованием распараллеливающих компиляторов, снижая затраты на этапах разработки и отладки, всё же не позволяет избавиться от основного недостатка статических методов: чрезвычайно высокая алгоритмическая сложность задач, связанных с оптимальным статическим распараллеливанием программ может в общем случае привести к недостаточной глубине распараллеливания.

Реализация подхода, при котором большая часть решений по распараллеливанию принимается в процессе выполнения программ—в динамике—могла бы дать шанс на преодоление трудностей, свойственных статическим методам и, как следствие, значительно расширить круг людей, способных эффективно использовать параллельные программно-аппаратные комплексы.

В данной работе рассматривается разработанная в Исследовательском центре мультипроцессорных систем (ИЦМС) ИПС РАН *система автоматического динамического распараллеливания программ—Т-система.*

Вычислительная модель

Для реализации концепции автоматического динамического распараллеливания программ нами была предложена новая модель организации вычислительного процесса [4, 5], основанная на следующих базовых принципах:

- В качестве основной парадигмы рассматривается *функциональное программирование*. Программа представляет собой набор чистых (без побочных эффектов) функций. Каждая функция может иметь несколько аргументов и несколько результатов. В то же время *тела*

[1] Работа выполнена при поддержке Российского фонда фундаментальных исследований, грант № 97-01-00172

функций могут быть описаны в императивном стиле (на языках типа FORTRAN, C и т. п.) Важно только, чтобы:

— всю информацию извне функция получала только через свои аргументы;

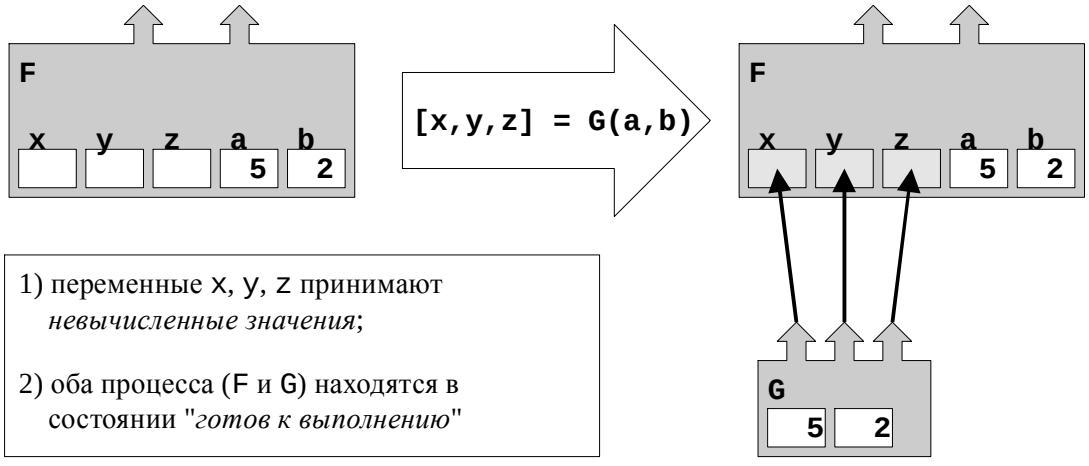
— вся информация из функции передавалась только через явно описанные результаты.

- Вызов функции G

  `[x,y,z] = G(a,b)`,

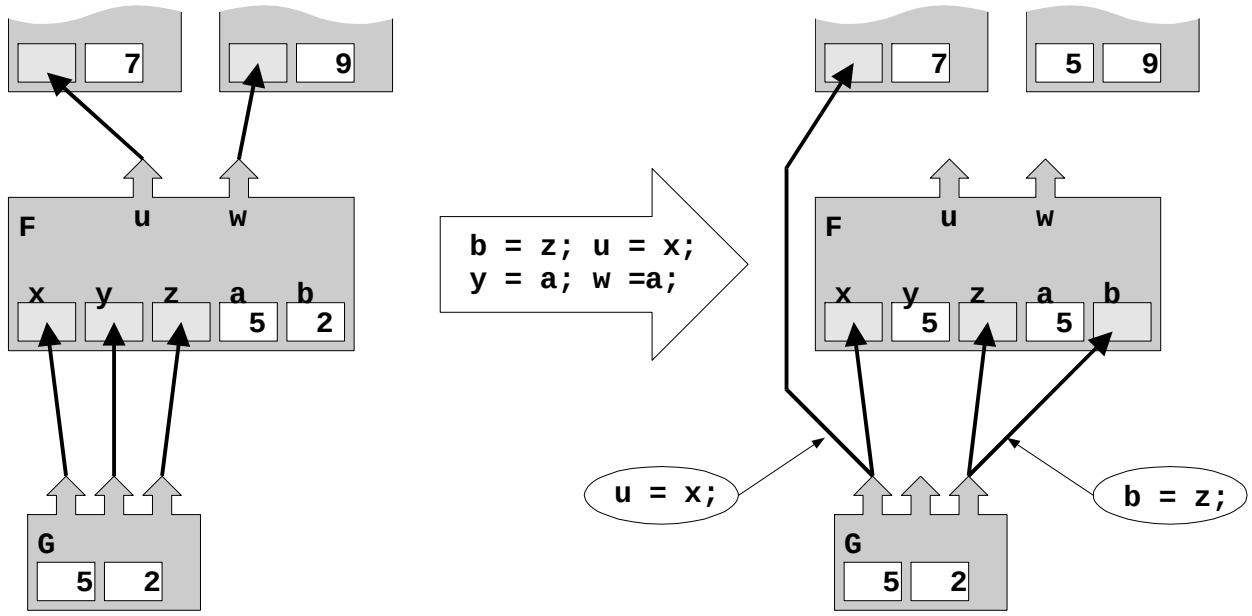
  производимый в процессе вычисления функции F выполняется нетрадиционным способом (т.н. *сетевой вызов* функции). При этом порождается новый *процесс* с несколькими *входами* (в соответствии с числом аргументов функции G) и несколькими *выходами* (в соответствии с числом результатов функции G). Выходы нового процесса связываются с соответствующими переменными процесса F—в нашем примере это x,y,z—отношением *"поставщик потребитель"*, и тем самым, переменные-потребители принимают *неготовые (не вычисленные) значения*. Порожденный процесс G должен вычислить функцию G и заменить неготовые значения у всех своих потребителей на соответствующие результаты функции G (см. рис. 1).

- Все процессы вычисления функций порождаются в состоянии *"готов к вычислению"*. Так например описанный выше вызов функции G из процесса F не ведет к потере процессом F *готовности*, т.е. способности выполняться. Единственное событие, приводящее *к потере готовности (к засыпанию)* процесса—это попытка процесса выполнить с неготовым значением любую операцию, отличающуюся от операций передачи значений; операции передачи неготовых значений не приводят к потере процессом готовности и реализуются за счет изменения вычисляемой сети (см. Рис. 2).
- Если некоторый процесс предпринял попытку выполнить с неготовым значением операцию, отличную от операции передачи данных, то такой процесс *теряет готовность—засыпает*. Готовность процесса будет восстановлена—*процесс будет разбужен*—в тот момент, когда соответствующее неготовое значение (причина засыпания) будет заменено поставщиком на обычное значение.
- Таким образом, в каждый момент времени состояние вычисления представлено *вычисляемой сетью*, *узлами* которой являются запущенные процессы и структуры данных, а *дугами*—отношения поставщик-потребитель. В процессе выполнения программы *вычисляемая сеть автотрансформируется*: в моменты вызовов функций в ней появляются новые узлы, в моменты завершения вычисления всех потребляемых результатов функций узлы исчезают, при выполнении допустимых операции с неготовыми значениями появляются и исчезают дуги.



Семантика функционального вызова.

Рис. 1.



1) операции *передачи невычисленных значений* не приводят к потери процессом состояния "*готов к выполнению*"

2) операции передачи обычных значений тоже могут изменить вычисляемую сеть

Трансформации вычисляемой сети при передаче значений.

Рис. 2.

Автотрансформация вычисляемой сети как модель организации вычислений поддерживает *возможность* автоматического динамического распараллеливания программ—для большого числа приложений в каждый момент времени при такой организации вычислений в вычисляемой сети находится достаточное количество готовых к выполнению процессов, которые можно выполнять параллельно. Модель предусматривает единственный механизм синхронизации процессов—описанный выше механизм засыпания и побудки процессов.

Наиболее близкие аналоги данной вычислительной модели использовались в работах Э.Х. Тыугу, А.С. Нариньяни, в параллельных реализациях reduction machine (см., например, [2]), системе МДА (мультипроцессор с динамической архитектурой) [1], проекте Daisy/DSI [3] и системе программирования Cilk [6].

Т-система

Т-система является средой программирования, поддерживающей описанную в предыдущем разделе вычислительную модель на мультипроцессорной платформе.

Программно-аппаратная платформа для Т-системы

Текущая реализация Т-системы в качестве программно-аппаратной платформы использует мультикомпьютер, построенный как сеть рабочих станций (в англоязычных публикациях часто используется аббревиатура NOW—от “Network Of Workstations”).

Каждый из процессорных элементов мультикомпьютера представляет собой компьютер семейства IBM PC, работающий под управлением операционной системы Linux—клона ОС UNIX. Заметим, что в составе мультикомпьютера в качестве процессорных элементов могут быть использованы как монопроцессорные компьютеры семейства IBM PC, так и SMP-мультипроцессоры—симметричные мультипроцессоры класса IBM PC, удовлетворяющим спецификациям Intel SMP 1.1/1.4.

В качестве сети, объединяющей процессорные элементы в единую вычислительную систему, используется локальная сеть с протоколом TCP/IP.

Основным преимуществом архитектуры NOW является возможность использования широко распространенных процессорных элементов и сетевых технологий массового применения. Промышленные отрасли, связанные с производством этих изделий, в настоящее время проходят этап бурного технологического развития, что позволяет постоянно поддерживать значительную вычислительную мощность NOW-мультикомпьютера, обеспечивая низкое соотношение цена/производительность.

В то же время NOW/Intel/Linux не является единственной платформой, пригодной для реализации T-системы: в наши планы развития T-системы входит перенос T-системы под другие архитектуры и клоны UNIX (возможно—под Microsoft Windows NT).

Основные функции T-системы

Основной функцией T-системы является поддержка вычислительной модели “автотрансформация вычисляемой сети”. В рамках этой функции T-система поддерживает основные структуры данных и операции, связанные с реализацией вычислительной модели, в частности:

- сетевой вызов функции;
- рассылка вычисленного результата функции потребителям;
- операции работы с неготовыми значениями (передачи неготовых значений);
- поддержка механизмов неявной синхронизации процессов (засыпание, побудка и т. п.)

В мультикомпьютере на каждом процессорном элементе работает свой экземпляр T-системы—*соисполнитель*. Пользовательская задача выполняется под управлением и при поддержке совокупности всех соисполнителей. С утилизацией мультипроцессорной платформы связана следующая группа функций ядра:

- поддержка вычислительной модели в условиях размещения фрагментов вычисляемой сети в памяти различных соисполнителей—реализация на межпроцессорном уровне следующих механизмов и понятий:
  - — отношения “поставщик-потребитель”;
  - — механизмы синхронизации процессов;
  - — доступ к значениям аргументов функции;
  - — рассылка результатов потребителям.
- внешнее планирование—механизмы запроса и передачи готового к исполнению процесса от одного соисполнителя другому, работающему на простаивающем процессорном элементе мультикомпьютера.

Основные компоненты T-системы

Текущая реализация T-системы с точки зрения её структуры состоит из трех относительно независимых компонент:

- TThreads—компонента, поддерживающую множественные потоки управления в рамках единого процесса ОС Linux;
- LMCE—компонента, обеспечивающую распределённый запуск соисполнителей, обмен сообщениями между ними и их согласованное завершение;
- T-ядро, реализующее вычислительную модель с использованием средств, предоставляемых TThreads и LMCE.

Tthreads. С каждым процессом в T-системе связан свой поток управления (в англоязычной терминологии—thread), в терминах вычислительной модели соответствующий выполнению каждой функции пользовательской программы. Поскольку процесс-родитель в момент порождения нового процесса не завершается, то в рамках одного соисполнителя, являющегося с точки зрения ОС Linux обычным пользовательским процессом, должны сосуществовать множественные потоки управления.

Реализуемый компонентой TThreads аппарат множественных потоков управления асимметричен: помимо обычных потоков управления, в соисполнителе всегда присутствует един-

ственный выделенный поток управления, которому отводится роль “ядра”. TThreads предоставляет средства для создания, завершения и планирования выполнения “ядром” обычных потоков управления в рамках одного соисполнителя, т.е. средства внутреннего планирования. Кроме того, TThreads поддерживает механизм взаимодействия между “ядром” и обычными потоками управления, аналогичный механизму системного вызова: “ядро” выполняет заказы, переданные ему обычным потоком управления и возвращает потоку-заказчику результат.

LMCE. Компонента LMCE реализует базовый уровень взаимодействия между соисполнителями. В состав LMCE входит набор программных модулей, включаемых в состав исполняемого файла задачи пользователя на этапе редактирования связей, и отдельный исполняемый файл, играющий роль программного коммутатора.

Стартовав на одном из процессорных элементов мультипроцессора, задача пользователя, используя механизм UNIX-сокетов, устанавливает соединение с выделенным ей программным коммутатором, исполняющимся на том же процессорном элементе. В совокупности эта пара процессов ОС Linux образует “центральный” соисполнитель. Используя предоставляемые ОС Linux средства удаленного запуска задач, центральный соисполнитель производит запуск аналогичных соисполнителей на других процессорных элементах. Программные коммутаторы, входящие в состав различных соисполнителей, взаимодействуют по принципу “каждый с каждым”, используя механизм UDP-сокетов—один из программных интерфейсов протокола IP.

На основе механизмов UNIX-сокетов и UDP-сокетов компоненты LMCE реализуют единый LMCE-протокол обмена сообщениями между соисполнителями. Этот протокол включает в себя набор предопределенных типов сообщений, используемых для обеспечения базовых функций LMCE (в качестве примера можно привести необходимость обеспечения согласованного завершения соисполнителей в случае окончания вычислений или возникновения критической ошибки в одном из соисполнителей). Кроме того, LMCE предоставляет возможности для расширения базового LMCE-протокола дополнительными типами сообщений, специфичными для программной системы, разрабатываемой с использованием LMCE.

T-ядро. Реализуя основные функции T-системы, T-ядро для представления вычисляемой сети использует списковые структуры специального вида (T-структуры). T-структуры состоят из звеньев—тегированных фрагментов памяти. Звенья T-структур расположены последовательно в выделенной для них области памяти соисполнителя—звеньевой памяти. В частности, каждому процессу T-системы (T-процессу), помимо потока управления, соответствует T-структура, включающая в себя аргументы, результаты и локальные T-переменные T-функции, вычисляемой данным процессом.

Для управления памятью используется собственная дисциплина выделения памяти и сборка мусора.

Взаимодействие между T-процессами и T-ядром, которое функционирует как выделенный поток управления, осуществляется с использованием механизма “системных вызовов” компоненты TThreads. Каждому типу производимых T-процессом атомарных действий по преобразованию вычисляемой сети, существенно изменяющих её состояние, соответствует отдельный системный вызов:

- сетевому вызову функции—системный вызов *spark*;
- ожиданию одного или нескольких неготовых значений—системный вызов *alt*;
- рассылке потребителям вычисленного функцией значения одного из её результатов—системный вызов *send*, и т.д.

T-ядро расширяет LMCE-протокол дополнительными типами сообщений. Эти типы сообщений предназначены для поддержки взаимодействия экземпляров T-ядра, входящих в состав различных соисполнителей.

С этой же целью T-ядром поддерживается механизм *разъёмных пар*. Каждая из разъёмных пар используется для представления дуги вычисляемой сети на межпроцессорном уровне и состоит из двух *разъёмов*: *входного* и *выходного*. Входной разъём пары является активным, и по его запросу выходной разъём, расположенный в памяти другого соисполнителя, активизирует процесс передачи необходимых данных. В тот момент, когда запрошенные данные окажутся вычисленными, они будут преобразованы в *транспортную форму*, позволяющую

осуществлять передачу фрагментов вычисляемой сети между соисполнителями. После преобразования в транспортную форму данные передаются тому экземпляру Т-ядра, которому принадлежит входной разъём.

Аналогичный механизм используется и для передачи Т-процесса от одного соисполнителя другому: получив от простаивающего соисполнителя сообщение-запрос на передачу процесса, Т-ядро может принять решение об удовлетворении запроса и произвести поиск процесса-кандидата для передачи. В случае, если этот поиск заканчивается успехом, Т-процесс преобразуется в транспортную форму, и единым сообщением передается соисполнителю, выдавшему запрос.

Программирование в Т-системе

Программы, разрабатываемые для исполнения в среде Т-системы должны быть написаны в функциональном стиле. То есть Т-программа является определением набора чистых (без побочных эффектов) Т-функций.

Учитывая то обстоятельство, что сегодня императивные языки чаще используются для разработки прикладных систем, чем функциональные, при разработке языка программирования для Т-системы (рабочее название: *язык t2cp*) было принято решение использовать расширение языка С небольшим числом новых конструкций (Т-конструкций):

- *описание заголовка Т-функций*—определяет имя Т-функции, список имен ее аргументов, результатов и вспомогательных Т-переменных;
- *операторы передачи Т-значений* (в том числе—неготовых): *присваивание* Т-переменной нового Т-значения и *выдача одного из результатов* Т-функции;
- *оператор вызова Т-функции*—приводит к порождению процесса);
- *операции* над Т-структурами—конструкторы и селекторы для Т-структур: операции работы со списками и массивами Т-звеньев, операции размещения С-данных в Т-структурах и извлечения С-данных из Т-структур.

Язык t2cp реализуется несложным препроцессором, преобразующим входную Т-программу в программу на языке С: перечисленные выше Т-конструкции заменяются на соответствующие вызовы Т-системы; остальная часть программы оставляется без изменения. Результат препроцессирования обрабатывается стандартным в ОС Linux компилятором языка С и собирается с библиотеками Т-системы в исполняемый файл, пригодный для запуска на любой вычислительной установке вида “TCP/IP-сеть Linux-машин”, содержащей любое число процессоров. При запуске исполняемого файла считывается конфигурационный файл с перечислением IP-адресов и характеристик (монопроцессор или SMP-мультипроцессор) узлов сети. За счет механизма rcp/rsh на указанных узлах вычислительной установки запускаются экземпляры Т-ядра, образующих общую среду исполнения для Т-задачи.

С точки зрения программиста, язык t2cp мало чем отличается от языка С: большая часть текста Т-программы является обычным текстом на языке С, и только небольшая часть—Т-конструкциями. Разработку Т-программ удобно вести “сверху вниз”:

- Сначала планируется *верхний уровень Т-задачи*—небольшой набор Т-функций, в совокупности реализующий задачу.
- Затем разрабатываются—на языке С с использованием Т-конструкций—тела Т-функций. При этом, *в рамках реализации тела одной Т-функции*, программисту доступны все средства языка С, в том числе и те, которые принято считать несоответствующими функциональному стилю: операторы goto, вспомогательные С-функции с побочным эффектом, ограниченным рамками тела реализуемой Т-функции, и т.п.

Тем самым, разработка верхнего уровня Т-программы ведется исключительно в функциональном стиле, а при реализации тел Т-функций программист волен придерживаться того стиля использования языка С, к которому он привык.

Как показывает наша практика, во многих случаях хорошо структурированные программы на языке С несложно переписать под Т-систему—во многих случаях для этого достаточно переоформить (записать в виде Т-конструкций) верхний уровень программы.

Использование чисто функционального стиля на верхнем уровне разработки программы влечет за собой все его достоинства и недостатки: данный уровень программы должен быть тщательнейшим образом спланирован и учитывать особенности исполнения разрабатываемых Т-функций в Т-системе. В частности:

- Необходимо следить, чтобы Т-функции реализовывали достаточно ёмкие (с точки зрения объема вычисления) фрагменты задачи.
- Если вычисление некоторой Т-функции может привести к выдаче большого потока[2)] данных и/или к выполнению большого количества вызовов Т-функций, приводящих к порождению большого числа Т-процессов, то возможно необходимо пересмотреть функциональное описание задачи. При этом, в большинстве случаев достаточно определить некоторые функции ленивыми (lazy) или ленивыми с потоковыми результатами[3)].

Описанный выше язык t2cp и его реализация отражают сегодняшний этап развития Т-проекта. Возможными дальнейшими шагами в развитии языковых средств Т-системы являются:

- реализация аналогичных препроцессорных расширений для других императивных языков программирования ОС Linux (например, для языка Фортран);
- разработка и реализация Т-языка как диалекта (а не препроцессируемого расширения). императивных языков программирования;
- реализация компиляторов с существующих функциональных языков программирования.

ЗАКЛЮЧЕНИЕ

Исследования параллельных вычислений проводятся в мире и нашей стране достаточно давно и в данной области достигнуты значительные результаты. Параллельные компьютеры успешно используются для задач, связанных с большим объемом регулярных (матричных, векторных и т.п.) численных расчетов и в этой области применения суперкомпьютеров разработаны достаточно эффективные средства организации параллельного счета.

Однако, при всей своей исключительной важности, данный класс задач не покрывает всей области возможного применения суперкомпьютеров. Вне этой области остаются программы со сложной логикой развития процесса вычисления, программы, связанные с обработкой нечисловых данных и данных, имеющими сложное (не матричное) представление—динамически порождаемые списки, деревья, графы и т.п. Многие задачи из таких областей, как компьютерная алгебра, интеллектуальные системы, моделирование сложного поведения и др. связаны с большим объемом обработки данных (то есть, *требуют* большой производительности вычислительной установки) и обладают потенциальным *параллелизмом*, однако этот параллелизм часто *скрыт* до времени исполнения программы и проявляется только в динамике. Например, во многих игровых программах как момент, так и сама возможность порождения процесса анализа некоторой игровой ситуации (или класса ситуаций) принципиально не могут быть предсказаны до запуска программы.

Именно на такой класс задач—задач с *динамическим* (*скрытым* до момента запуска программы) *параллелизмом*—и была первоначально ориентирована разработка, рассматриваемая в данной статье. Уточнение класса задач, для которых использование Т-системы целесообразно, является предметом дальнейших исследований и во многом будет определяться результатами использования Т-системы для решения практических задач.

На сегодняшний день на Т-языке написан ряд задач—как демонстрационных, так и практически значимых—имеющих различную алгоритмическую организацию и относящихся к различным прикладным областям. Среди прочего:

- Построение методом трассировки лучей качественных изображений 3D-сцен по их описанию.

---

[2] В Т-языке поддержано понятие *потоковый результат* Т-функции.

[3] В Т-языке поддержаны понятия *ленивая (lazy) Т-функция* и *ленивая Т-функция с потоковыми результатами*.

- Вычисление *sl*- и *so*-полиномов для 3-графов [7]—реальная задача компьютерной алгебры с высокой вычислительной сложностью: вычисление Т-программой so-полинома для 3-графа с 24 вершинами на установке с 12 процессорами Pentium Pro/200 заняло 55.23 часов.
- Фильтрация линейного потока данных динамически порождаемыми фильтрами—на примере построения потока простых чисел методом “решето Эратосфена”.
- Построение множества данных, заданного рекурсивным определением—на примере алгоритма построения бесконфликтных расстановок ферзей на шахматной доске.
- Игровая программа—переборный алгоритм—с отсечениями, эвристикой и учетом ограничений на время счета—выбора очередного хода в игре Pousse (усложненный вариант “крестиков-ноликов”).

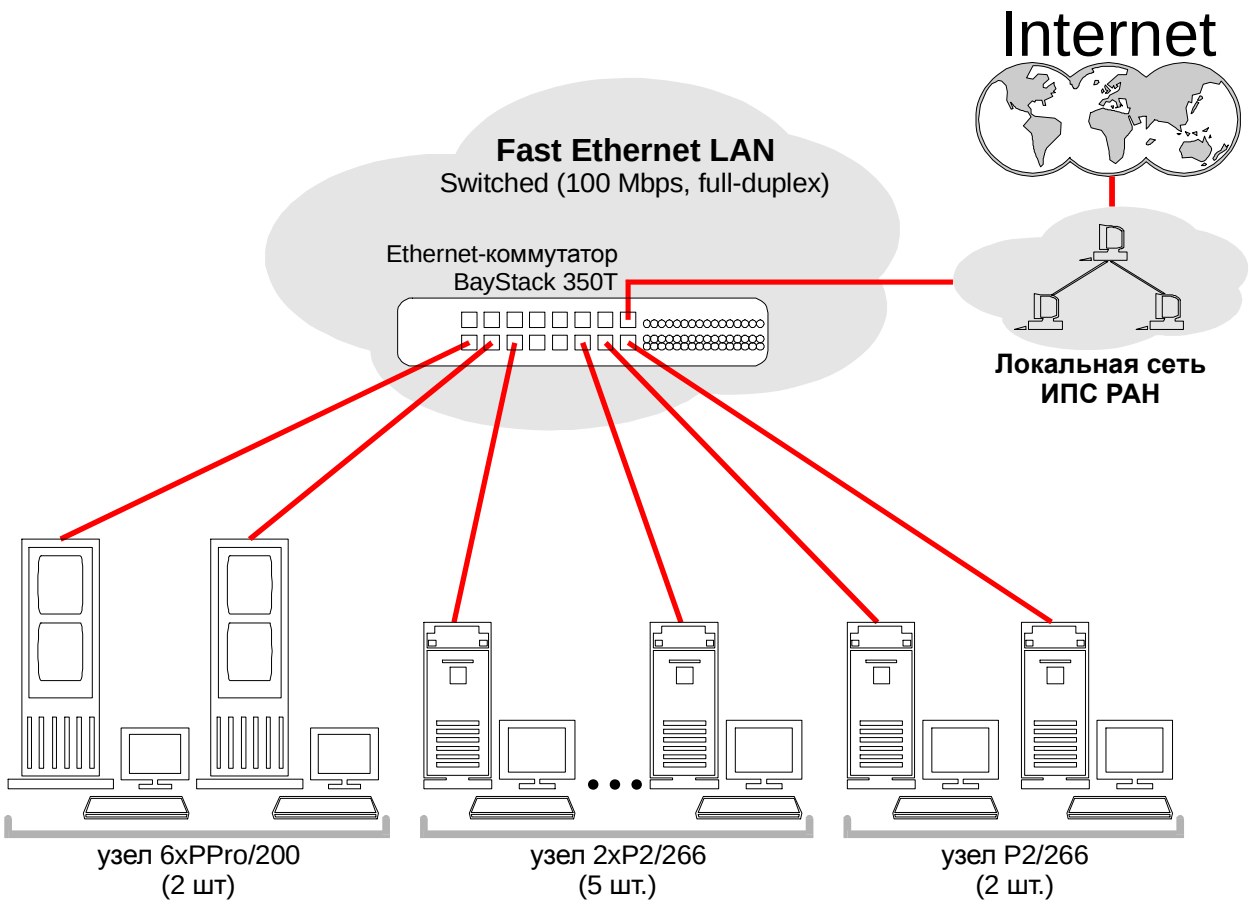


Рис. 3. Структурная схема комплекса аппаратных средств мультипроцессора

Таблица 1. Укрупненная спецификация комплекса аппаратных средств мультипроцессора Т-системы

	Число узлов	Число процессоров в узле	Тип процессора	Теоретический пик процессора (MFlops)	Теоретический пик узла (MFlops)	RAM (MB)	HDD (GB)
узел 6xPPro/200	2	6	PPro/200	200	1,200	256	12.8
узел 2xP2/266	5	2	P2/266	266	532	128	6.4
узел 1xP2/266	2	1	P2/266	266	266	128	6.4
ВСЕГО:	**9**	**24**	**—**	**—**	**5,592**	**1,408**	**70.4**

В результате выполнения этих задач на мультикомпьютере достигался высокий уровень распараллеливания программ Т-системой. В этих экспериментах использовался мультикомпьютер (см. Рис 3 и Таб. 1), реализованный как быстрая локальная сеть (Switched Fast

Ethernet 100 Mbps) из 9 различных Linux-машин (в том числе мультипроцессорные SMP-компьютеры двух типов: шестипроцессорные ALR 6x6 и двухпроцессорные рабочие станции класса IBM PC). Всего в данной установке 24 процессора Pentium Pro/200 и Pentium II/266, 1,408 Mb оперативной памяти и 70.4 Gb дисковой памяти. Суммарная теоретическая пиковая производительность установки—5.5 GFlops.

Подробное описание языка t2cp, примеры T-программ и технические отчеты по экспериментам с T-системой заинтересованный читатель найдет по URL:

`ftp://ftp.botik.ru/pub/local/Sergei.Abramov/T-system/`

СПИСОК ЛИТЕРАТУРЫ

1. Плюснин В.У, Торгашёв В.А., Страхов В.Г. *Процессор с динамической архитектурой (ДАР)—новый класс проблемно-ориентированных процессоров для ЕС ЭВМ*// Тез. докл. 2-го Всесоюз. совещ. “Высокопроизводительные вычислительные системы”, Москва., 1984.
2. B.F. Goldberg. *Multiprocessor execution of functional programs*// Ph.D. thesis, Yale University, Dept. of Computer Science, 1988. Доступна как технический отчет YALEU/DCS/RR-618.
3. Eric R. Jeschke. *An Architecture for Parallel Symbolic Processing Based on Suspending Construction*//Ph.D. thesis, Indiana University, Computer Science Department, 1995. Доступна как Indiana University Computer Science Department Technical Report TR 445.
4. Abramov S.M., Adamowitch A.I., Nesterov I.A., Pimenov S.P., Shevchuck Yu.V. *Autotransformation of evaluation network as a basis for automatic dynamic parallelizing*//Proceedings of NATUG 1993 Meeting “Transputer: Research and Application”, May 10-11, 1993.
5. Abramov S.M., Nesterov I.A., Shevchuk Yu.V. *T-language. Preliminary description*//RCMS Tech. Report #09/18/1994. Доступен в Интернет: ftp://pub/local/RCMS/SSPA/tlp.ps (а также как tlp.ps.Z, tlp-ps.zip).
6. Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. *Cilk: An Efficient Multithreaded Runtime System*//In 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP '95), July 19-21, 1995, Santa Barbara, California, pp. 207-216.
7. D. Bar-Natan, *On the Vassiliev knot invariants*//Topology, 34 (1995) 423--472.