УДК 519.6

О ПАРАЛЛЕЛИЗМЕ С РАЗНЫХ СТОРОН

В.П. Кутепов
Московский Энергетический Институт (Технический Университет)
Россия, 111250, Москва, Е-250, Красноказарменная улица, дом 14
E-mail: kutepov@appmat.ru

Рассматриваются наиболее важные особенности параллелизма, которые существенны для построения языков параллельного программирования и для разработки параллельных программ. Предлагается обзор языков и сред параллельного программирования. Изложены подходы к созданию методов и программных средств, предназначенных для эффективного управления процессами выполнения параллельных программ на компьютерной системе.

PARALLELISM THROUGH DIFFERENT PERSPECTIVES / V.P. Kutepov (Moscow Power Engineering Institute (Technical University), st. Krasnokazarmennaya 14, Moscow, 111250, Russia). Significant features of parallelism that important for parallel programming languages composition and parallel programs development are considered. The review of parallel programming languages and development environments is suggested. The approach for development of techniques and software tools that are used for efficient control of execution of parallel programs at computers is described.

Введение.

История развития компьютерной индустрии, старту которой мы обязаны фон Нейману, сегодня вполне может быть удовлетворена своими достижениями. На смену первым компьютерам, занимающим неколько комнат, с быстрнодействием в несколько тысяч операций в секунду, пришли книжного размера компьютеры с быстродействием в несколько миллиардов операций в секунду и компьютерные системы, насчитывающие несколько сотен тысяч компьютерных элементов. Они способны решать сложные научно-технические задачи, требующие быстродействия $10^{15} \div 10^{17}$ операций в секунду.Напомним, что в 60-е годы после изобретения Л.В. Канторовичем методов линейного программирования экономисты пытались решить задачу оптимизации перевозок в рамках страны с помощью компьютеров. Математическая модель этой задачи, описанная системой линейных уравнений, содержала 10^6 переменных и 10^4 ограничений и не могла быть решена в приемлемое время на компьютерах того времени. Да и сегодня потребуется большое искусство программирования и большая компьютерная система (КС), чтобы достичь желаемого времени решения этой задачи, используя параллелизм.

Конечно, сегодня нет особых проблем увеличить масштаб компьютеной системы, тиражируя стандартные многопроцессорные или многокомпьютерные узлы, используя расширямые коммуникации. Однако, обычно заявленная пиковая производительность таких систем будет, как правило, весьма далека от реальной при решении конкретной сложной задачи. При недостатке оперативной памяти придется обращаться к дискам, а это уже 1,5 мсек из-за латентности для каждого обращения; при частых обменах данными между узлами компьютерной системы узким «горлом», заметно удаляющим нас от пиковой производительности, станут каналы и инфраструктура коммуникаций [1].

Поэтому так не проста работа программиста по выбору метода решения задачи, разработке и оптимизацции параллельной программы и организации ее выполнения на конкретной КС.

Не будем здесь говорить о том, какие нас ожидают трудности, если мы будем пытаться оптимизировать наши программы для выполнения на GRID системах и т.п.

Сегодня достаточно хорошо исследована традиционная поледовательность решения на компьютерах задач вычислительной математики, включающая этапы: выбор метода и языка, разработка и оптимизация программы, выполнение. При этом у программиста есть возможность использования подходящего языка последовательного программирования (FORTRAN, PASKAL, C, C++ и др.), если довериться ОС в том, что ресурсы во время выполнения его программы будут эффективно использованы.

У КС пока эти возможности сильно ограничены, и программисту самому приходится искать ответы на принципиально новые и более сложные вопросы: как определитьо масштаб КС, обеспечивающий достижение требуемого времени выполнения программыы, как найти наилучший метод решения задачи, каким языком необходимо воспользоваться, чтобы он был адекватен задаче и позволял отразить возникший параллелизм в методе ее решения, как опрелить максимальную зернистость (степень распараллеливания задачи [2]) в параллельной программе, наконец, как организовать выполнение программы (синхронизировать процессы и распределить ресурсы), чтобы достичь желаемого эффекта.

Поиск ответов на все эти вопросы далеко не завершен и, по-видимому, потребуется немалое время для проб и ошибок, чтобы найти приемлемые решения перечисленных проблем.

Ниже мы попытаемся более предметно поговорить об особенностях и способах задания параллелизма, языках и средах параллельного программирования, управлении параллельными процессами и операционных средствах, необходимых для его реализации.

1. Особенности, способы представления и характеристики параллелизма.

Понятие параллелизма введено в статье С. Гилла [3] с целью изменения парадигмы программы как последовательного алгоритмического процесса и создания языков, позволяющих описывать параллельные действия, естественные в любом алгоритме. Однако, концепция последовательной программы фон Неймана оказалась настолько простой, как и ее компьютерная реализация, что и сейчас параллелизм часто рассматривается как нечто специфическое и искусственно вводимое в программу дополнение, направленное на ускорение ее выполнения.

Однако каждый програмист хорошо знает, какие трудности возникают при программировании на последовательных языках задач моделирования, управления распредленными системами, операционных систем, многих многоаспектных вычислительных задач и др. Поэтому при создании языков параллельного программирования идут двумя путями: либо пытаются расширить последовательный язык за счет включения в него средств описания параллелизма, сохраняя тем самым привычные и неплохо отработанные приемы последовательного программирования, либо пытаются изначально создавать язык параллельного программирования. Векторные команды, примитивы создания ветвей fork и join (первые широко использовались в системах ILLIAC, вторые – в системах БЭРРОУЗ и Эльбрус), API средства для описания параллельных процессов: PVM, MPI и Multithreading – примеры расширения последовательных языков с целью возможности описания параллелизма. На другом конце – попытки создания языков параллельного программирования на основе определенных базовых конструкций, адекватных тем общим формам параллелизма, которые возникают в реальных задачах.

Ясно, что в конечном итоге (в реализации) параллелизм – это одновременное протекание и взаимодействие процессов. Способы порождения и взаимодействия процессов, их асинхронность и синхронизация, одновременность и строгое упорядочивание – основополагающие понятия процессных моделей [4-7], на которых стоят хорошо известные яыковые средства для описания параллелизма на уровне процессов: PVM, MPI, Multithreading.

На задачном уровне фундаментальным условием параллелизма является информационная независимость (независимость по данным) фрагментов декомпозиции задачи при построении алгоритма ее решения, будь это подзадачи или более мекие элементы, например, команды или операторы программы.

На этой основе мы создали язык граф-схемного потокового параллельного программирования [8,9], который принципиально изменяет парадигму последовательной программы и основывается на явном задании в программе зависимостей по данным (и только

их) между фрагментами программы. Как следствие, независимые фрагменты могут выполняться одновременно.

Другие важные особенности этого языка:

- возможность визуального проектирования параллельной программы в виде граф-схемы, одновременно отражающей посредством введения модулей декомпозицию задачи и связи по данным между модулями программы;
- возможность построения рекурсивных граф-схем и отражение иерархии при декомпозиционной стратегии пошаговой разработки программы;
- простое описание потоковой обработки, в частности, при применении программы к потоку (в том числе поступающему в реальном времени) тегированных данных;
- естественное представление вычислений с упреждением;
- использование различных языков последовательного программирования для построения программ модулей.

Другой способ неявного отражения параллелизма на задачном уровне – использование функциональной нотации. Ясно, что для любой n-арной функции $f(x_1, x_2,..., x_n)$, $n \geq 1$, можно одновременно вычислять значения функций, подставленных вместо ее аргументов. Условный оператор *if* $P(x)$ *then* $f_1(x)$ *else* $f_2(x)$ также может рассматриваться как специальная тернарная функция $Y(P(x), f_1(x), f_2(x))$, из чего следует возможность одновременного вычисления функций $P(x), f_1(x), f_2(x)$.

Мы создали язык функционального параллельного программирования FPTL [10], в котором в отличие от функциональных языков ML, Haskell и других, основанных на lambda-исчислении, используются традиционные математические формы задания функций.

Введенные в этом языке операции композиции функций позволяют в схемной форме отражать параллелизм в функциональной программе. В языке FPTL можно определять абстрактные типы данных, использовать библиотечные функции, организовывать модульные функциональные программы. Фукнции и данные в FPTL определяются в общем случае в виде систем функциональных и реляционных уравнений, а поскольку рекурсия является более мощным средством задания параллелизма по сравнению с определением функций посредством циклических конструкций, это придает языку большую выразительную силу [11].

Уникальной является среда проектирования функциональных программ на FPTL, в которой помимо традиционных инструментов отладки программ созданы не имеющие аналогов подсистемы эквивалентных преобразований программы, позволяющие автоматически приводить ее к максимально параллельной форме, а также подсистемы анализа структурной и вычислительной сложности и верификации [10,11].

Реализация языка на многопроцессорных КС [10,11], созданные средства для эффективного планирования порождаемых при выполнении функциональной программы процессов и распределения ресурсов полностью избавляют программиста от необходимости этой работы, как это делается при использовании MPI, Multithreading и других средств параллельного программирования.

Отметим здесь также, что стремление приблизить языковые средства к проблемной среде явилось результатом создания целого ряда проблемно-ориентированных языков параллельного программирования: параллельный FORTRAN, DVM, MPC (ориентация на решение задач линейной алгебры), PARLOG (параллельный логический вывод), OCCAM (взаимодействие параллельно выполняемых процессов) и других.

Ниже мы рассмотрим те наиболее важные особенности паралелизма, которые существенны для построения языков параллельного программирования и для разработки параллельных программ.

1) Явный и неявный параллелизм.

В последовательных программах, функциональных языках и других системах параллелизм представлен неявно, и необходимы соответствующие решения для его выявления на стадии трансляции или интерпретации. PVM, MPI, Multithreading, примитивы задания векторного и нитевого (fork и join) параллелизма – примеры явного задания параллелизма на процессном уровне.

2) Коммутативный и некоммутативный параллелизм.

Первое предполагает, что возможен произвольный порядок выполнения фрагментов программы, как, например, для векторных команд. В условном операторе *if P(x) then* $f_1(x)$ *else* $f_2(x)$, начав вычисление с $f_1(x)$ или $f_2(x)$, мы не всегда можем получить существующий результат (пусть вычисление $f_1(x)$ длится неограниченно, а $P(x)$ ложно и результат $f_2(x)$ определен).

Результативность достигается, если по крайней мере вычисление $P(x)$ не откладывется до получения результата $f_1(x)$ и $f_2(x)$.

3) Упреждающий и наследственный параллелизм.

В условном операторе мы можем вычислить с упреждением значение функций $f_1(x)$ и $f_2(x)$, стремясь добиться большего ускорения (в лучшем случае при равенстве времен вычисления значений $P(x)$, $f_1(x)$, $f_2(x)$ оно равно двум). Однако, какой результат $f_1(x)$ или $f_2(x)$ потребуется, станет ясно только после вычисления $P(x)$. Ясно, что выполнение условного оператора легко сводится к последовательной форме.

Однако, существует огромное множество так называемых параллельных функций (функций с наследственным параллелизмом [12]), которые нельзя корректно вычислить простым сведением вычислений их составляющих к последовательной форме. Приведем хорошо известный в телефонии пример функции голосования $f(x_1, x_2, x_3)$, такой, что ее значение определено, если определены любые два ее аргумента (положим, что вместо них подставлены другие функции), при этом значения функции является значением одного из пары равных аргументов; в противном случае значение функции не определено. Очевидно, при строго последовательной форме вычисления значений этой функции алгоритм должен быть устроен так, что вычисление значений любых функций, стоящих на месте аргументов x_1, x_2 и x_3, не должно откладываться на неограниченное время. Это хороший пример для

того, чтобы еще раз напомнить о нетривиальной реальности во «взаимоотношениях» последовательной и параллельной парадигм программирования.

4) Потоковый параллелизм.

Этот вид параллелизма возникает при параллельной обработке данных, типичным и простым примером которого является конвейерная обработка (обработка потоков данных, производство деталей и т.п.).

Программа, как правило, применяется к множеству данных и у нас есть возможность организовать ее копии, применяя их одновременно к различным данным, или построить процесс таким образом, чтобы данные (обычно тегированные) продвигались от одного фрагмента к другому, зависящему от данного фрагмента, и инициализация выполнения любого фрагмента осуществлялась по готовности поступающих данных. Так устроены граф-схемные программы [8,9], модули которых одновременно могут применяться ко всем поступившим данным путем копирования их подпрограмм и применения каждой из них к соответствующему кортежу поступивших тегированных данных. Теория систем массового обслуживания создавалась с целью описания процессов функционирования и исследования эффективности огромного многообразия реальных производственных и обслуживающих систем, с которыми мы сталкиваемся почти на каждом шагу. Граф-схемный язык позволяет естественно описывать такие системы и возникающий в них параллелизм.

5) Нужно отметить и другие такие важные особенности параллелизма, как например асинхронный и синхронный параллелизм.

Первое предполагает реализацию без задержек всех возникающих в процессе выполнения программы возможностей одновременного выполнения ее фрагментов независимо от их длительности. Второй вид параллелизма, как правило, обусловлен либо логическими причинами (процесс может начаться только после завершения нескольких других процессов) или используется с целью упрощения описания и выполнения параллельных фрагментов (векторные команды – типичный пример).

Проблемная среда, обычно характеризуемая используемыми средствами для описания присущих ей задач (функциональное, логическое, процессное и др. описание), диктует свои способы выявления и представления параллелизма.

Перейдем к характеристикам параллелизма, играющим определенную роль в его реализации на многомашинных и многопроцессорных компьютерных системах.

6) Ограниченный и неограниченный параллелизм.

Мы говорим, что параллелизм в программе (в методе решения задачи) ограничен, если при любом ее выполнении на множестве ее входных значений количество могущих одновременно выполняться фрагментов ограниченно. В противном случае мы говорим, что параллелизм в программе не ограничен. Приведем пример параллельной функции вычисления *n!*:

$$F(i,j) = if\ i = j\ then\ i\ else\ F(i, \lfloor \frac{i+j}{2} \rfloor) \times F(\lfloor \frac{i+j}{2} \rfloor + 1, j),$$

где $\lfloor a \rceil$ – ближайшее целое к a.

Очевидно, $F(1, n) = n!$, и фрагменты $F(i, \lceil \frac{i+j}{2} \rceil)$, $F(\lceil \frac{i+j}{2} \rceil + 1, j)$ можно вычислять одновременно. Модель подобного параллельного вычисления значений рекурсивных функций детально описана в [10,11]. Для данного примера параллельное вычисление можно представить в виде динамческого развертывания и свертывания двоичного дерева, в упрощенном виде сводящего вычисление к форме, изображенной на рис. 1.

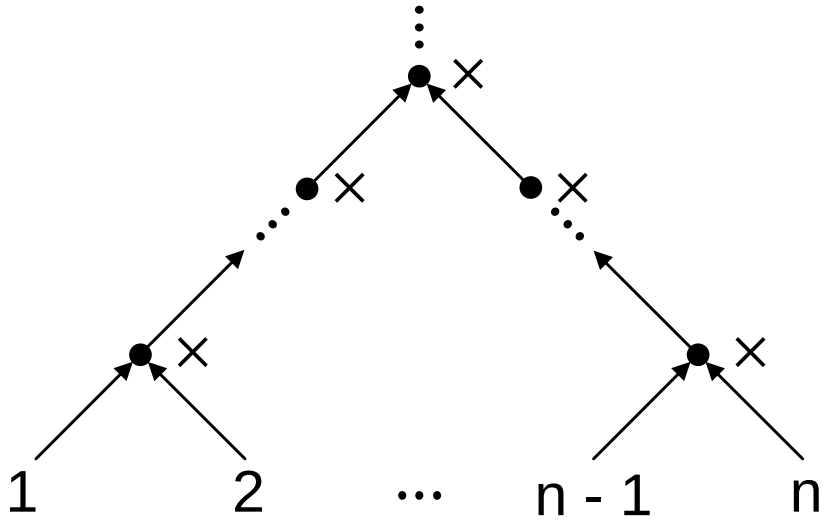

Рис. 1. Параллельное вычисление *n!*

Очевидно, что количество шагов последовательного вычисления *n!* имеет порядок *n*, а параллельного $log_2(n)$.

Ускорение $n / log_2(n)$ при неограниченном увеличении *n* растет неограниченно. Это означает, что при неограниченном количестве компьютерных ресурсов можно сколь угодно ускорять вычисление *n!* с увеличением *n*.

Хотя теоретики в поисках NP-полных проблем придумывают замысловатые примеры плохо распареллеливаемых задач, реальная практика показывает, что почти любая сложная задачалегко декомпозируется и, как правило, хорошо распараллеливается.

7) Глубина или степень распараллеливания задачи – следующая важная характеристика параллелизма, которая позволяет, с одной стороны, охарактеризовать предельное ускорение, которое может быть получено. С другой стороны, варьируя этот параметр, программист (или система управления выполнением программы) достигает минимизации времени выполнения программы и используемых при этом ресурсов на конкретной системе.

В вычислении *n!*, если глубина распараллеливания, определяемая как усредненная сложность фрагментов программы, выбрана с точностью сложности выполнения операции умножения, то нельзя надеяться, что будет получен хороший эффект при таком распараллеливании. Причина – велика степень распараллеливания и, как следствие, слишком велико время на управление вычислениями. Здесь мы, конечно, имеем в виду, что для вычисления *n!* используется приведенная выше рекурсивная функция, а не строится для

каждого n по соответствующему дереву последовательность, например, векторных команд умножения.

Как можно «регулировать» в программе степень параллелизма? Общий прием – параметризация, т.е. введение некоторой целочисленой константы, задающей степень параллелизма. Напрамер, для вычисления *n!* такая программа может быть следующего вида:

$$F_1(n,k,i,j) = if\ n = k\ then\ F(i,j) else\ F(i, i + \lceil \frac{j-i}{2} \rceil) \times F_1(n+1,k,i+\lceil \frac{j-i}{2} \rceil + 1, j),$$

Где *F(i, j)* – ранее описанная функция для вычисления *n!*, *k* – параметр, определяющий количество отрезков разбиения интервала $[i - j]$, на которых выполняется одновременное вычисление произведений, заключенных в них целых чисел. Очевидно, что F(1, k, 1, m) = m!.

Аналогичные функции с подобной параметризацией, задающей степень распараллеливания, легко строятся для вычисления $\sum_1^n a_i$, $\prod_1^n a_i$, численного интегрирования функций на отрезке, перемножения матриц (здесь k определяет количество одновременно перемножаемых блоков матриц) и др.

Гораздо сложнее подобная параметризация осуществляется для параллельных вычислений по сложным рекуррентным зависимостям, рекурсивных функций и др.

Глубина и степень распараллеливания всегда ограничены. Например, максимальный параллелизм при перемножении матриц A и B достигается, когда все элементы $c[i,j] = \sum_1^n a[i,j] \times b[i,j]$ результирующей матрицы вычисляются одновременно при одновременном вычислении произведений и последующих сложений в приведенной формуле. Хотя при максимальном распараллеливании эффект часто негативный и может быть достигнут в идеальном случае и, возможно, на компьютерах и компьютерных системах, построенных на других принципах и элементах.

2. Языки и среды параллельного программирования.

Введенное в свое время различение Скоттом и Стречи в языках программировния денотационной и операционной семантик для языков параллельного программирования имеет принципиальное значение.

Очевидно, что универсализм языка с позиций представления параллелизма предполагает возможность отражения в программе любых форм его проявления как на задачном уровне, т.е. данотационными средствами языка, так и на процессном уровне средствами операционной семантики. Из этого ясно, что если мы, например, ограничиваем себя на задачном уровне возможностью представления только векторного параллелизма или параллелизма ветвей (конструкциями fork и join), а в качестве операционных средтв будем использовать Multithreading с достаточно большим набором описателей параллельных процессов, то это будет стрельба из пушек по воробьям. Также бесполезно пытаться реализовать с помощью MPI-1 рекурсивные параллельные процессы, которые представлены в программе на FPTL или средствами Multithreading.

Ясно также, что средства описания параллельного программирования процессов в MPI-1 более слабы, чем в MPI-2 и Multithreading в части порождения и способов взаимодействия параллельных процессов. За правильность денотата или значения параллельных программ на

процессных языках MPI и Multithreading несет ответственность сам программист, доказывая, по крайней мере самому себе, что программа будет правильно выполнять описанные в ней функции, обладать однозначностью, не содержать неприятностей, таких как блокировки и зацикливание и т.п. [7].

Языки с формальной денотационной семантикой по своей сути содержат две языковые модели: одна задачная, а другая процессная. Это позволяет, в часности для вычислительных задач, разделить аспекты разработки параллельной программы, приблизить языковые средства к задачной, т.е. проблемной области. При этом, если параллельная программа с самого начала не связана с описанием процессов, программист может вовсе не интересоваться тем, как будет выполняться его параллельная программа.

Примерами таких двухаспектных языков являются язык граф-схемного потокового параллельного программирования [8,9], язык функционального параллельного программирования FPTL [10,11], отчасти параллельный FORTRAN, DVM и др.

В языке граф-схемного потокового параллельного программирования на задачном уровне (в его формальной денотационной семантике) реализована принципиально отличная от языков последовательного программирования парадигма (см. п.1), основанная на представлении фрагментов декомпозиции задачи в виде функций (программ модулей) и явном указании (через граф-схему) связей по данным (и только их) между подпрограммами. Как следствие, независимые фрагменты могут выполняться одновременно. Операционная семантика языка, определяющая правила параллельного выполнения граф-схемной программы, достаточно проста и основана на инициализации выполнения любого ее фрагмента по поступавшим на его входы тегированным данным.

Реализованная на многомашинных и многопроцессорных системах операционная семантика [8,9] в принципе избавляет программиста от необходимости планирования процессов выполнения программы и динамического распределения ресурсов компьютерной состемы.

Денотационная семантика параллельного FORTRAN`a, DVM и др. языков легко прослеживается в силу их узкого предназначения для описания легко распараллеливаемых задач линейной алгебры, где векторизация и распараллеливание «по циклам» – основные механизмы отражения параллелизма.

В языке FPTL [10,11] используется функциональная нотация для описания метода решения задач, а реализация параллелизма основана на модели асинхронного параллельного вычисления значений функций на КС [11]. Параллелизм в этом языке отражается в функциональной программе явно через используемые в ней операции композиции функций.

В расширении языков HASKELL, ML, последовательных языков с целью задания в программе параллелизма используют специальные примитивы задания параллелизма (мы уже говорили о векторных командах, операторах fork-join) или программисту предоставлена возможность по тексту программы вводить комментарии, какие ее фрагменты и каким образом должны быть распараллелены на стадии компиляции (OpenMP). MPI и Multithreading – более общие решения этого же плана, претендующие на статус стандартных средств описания параллельных процессов.

Ясно, что «масштаб» использования любого языка программирования определяется набором инструментов, которые обычно называют средой программирования и которые упрощают работу программиста.

В [13] мы обсуждали, какие механизмы в процессе перехода от задачи к ее программе носят творческий характер и полностью зависят от искусства программиста и какие из них можно выполнять с помощью соответствующих программных инструментов.

Поиск метода решения задачи, разработка и оптимизация программы и др. всегда будут прерогативой человека. Однако постоянно накапливаемый опыт, базы данных и библиотеки, аккумулирующие достигнутые в этих областях результаты, как мы можем видеть, уже сегодня часто существенно упрощают решение этих творческих задач.

Однако многое в программировании имеет вполне рациональный, т.е. алгоритмический характер.

Это касается поддержки проектных решений при разработке программы, документирования этого процесса, разделения аспектов с целью обеспечения условий для коллективной разработки программы, не говоря уже о привычных инструментах отладки программ и их тестирования.

При разработке параллельных программ, как правило решения сложных задач, эффективность программы (время ее выполнения, требуемые ресурсы) – изначально центральная проблема. Поэтому так важны инструменты прогонок программы на конкретной КС, сбор и обработка статистических данных, определяющих эффективность ее выполнения, средства оптимизации программы, проверки ее правильности как по содержанию, так и при выполнении (отсутствие тупиков, зацикливаний и т.п.).

Не менее важными являются инструменты, поддерживающие технологию декомпозиционного программирования (программирование сверху-вниз) и сборочного (снизу-вверх), коллективную разработку программы и др. [2]. Очевидно, все это заставит пересмотреть и существенно расширить те многобразные инструменты поддержки процессов программирования, созданные сегодня Microsoft, IBM и др., чтобы их можно было применять и в параллельном программировании.

3. Реализация параллелизма на компьютерных системах.

Помимо собственно обеспечения логического управления процессами параллельного выполнения программы на КС, предопределенного операционной семантикой языка, управление предполагает наличие в нем механизмов планирования процессов и распределения для них ресурсов, от чего в большой степени зависит и время выполнения программы, и использование ресурсов [14]. К сожалению, эта задача пока далека от удовлетворительного решения, в связи с чем программист сам заранее должен спланировать (определить приоритеты) выполнения процессов и даже распределить их на соответствующие узлы КС (пример, MPI).

В [14] мы изложили наши подходы к созданию методов и программных средств, предназначенных для эффективного управления процессами выполнения параллельных программ на КС. Принципиальный вывод из выполненных исследований состоит в том, что

несмотря на случайный характер протекающих при выполнении параллельных программ процессов, в событиях которых отражено использование различных ресурсов КС (процессоров, пямяти, коммуникаций), эти процессы вполне управляемы [14]. Мы можем достаточно точно измерять их характеристики и прогнозировать их изменения во времени и на основе этого строить эффективные адаптивные алгоритмы управления.

В создаваемых нами методах и программных средствах, направленных на эффективное управление процессами и оптимальное использование ресурсов КС, следующие решения имеют принципиальное значение.

В задаче управления выделяются две подзадачи: планирование процессов и распределение ресурсов. Решение первой подзадачи – упорядочение порождаемых процессов в соответствии с их особенностями, существенно влияющими на эффективность управления. Ясно, что процессы, выполняемые с упреждением должны иметь более низкий приоритет, чем безусловные процессы. Также важно поддерживать на максимальном уровне фронт работ – количество готовых для выполнения процессов. Разработанные нами эвристики позволяют решать успешно обе эти проблемы. Подзадача управления ресурсами КС существенно усложнена по причине вероятностной природы порождения процессов при выполнении параллельных программ.

В отсутствии априорной информации о характеристиках этих процессов приходится прибегать к использованию изощренных методов измерения и сглаживания (подавления высших гармоник) в протекающих процессах, чтобы можно было с достаточной точностью определять использование ими во времени различных ресурсов КС и прогнозировать его изменения [14]. При этом важно, чтобы сложность этих процедур не превышала эффект, который они в принципе дают.

Наконец, выбор самой организационной структуры управления, формы которой могут изменяться от централизации до полной децентрализации управления, становится принципиальным для больших КС. Объективно обусловленная иерархическая организация управления, максимальное перенесение центра тяжести в оптимизации загруженности компонентов КС на нижние уровни управления, в частности, существующие и хорошо отработанные средства управления процессами ОС, по-видимому, единственное правильное решение этой задачи [13,14].

Комплексное решение проблемы, создание эффективных алгоритмов и программных средств для управления процессами в КС и сетях позволит существенно облегчить задачу программиста при разработке параллельных програмсм, удовлетворяющих заданным временным и ресурсным ограничениям.

Заключение.

Следуя известному комментаторскому заключению, мы хотели бы верить, что вынесенные для обсуждения проблемы параллелизма действительно актуальны и рассмотрены с нужных сторон. А как это у нас получилось – судите сами.

Список источников.

- Лю Лян, Исследование эффективности реализации параллельных вычислений на кластере МЭИ, Вестник МЭИ, М.:2007, N5,с.90-96;
- В.П. Кутепов, В.Н. Фальк, Формы, языки представления, критерии и параметры сложности параллелизма. Программные продукты и системы (в печати);
- S. Gill, Parallel programming. The computer journal, 1958, N1, p. 2-10;
- R. Miller, A calculus for communicating systems. Lecture notes in computer science, Springer-Verlag, New York, 1980, p.92;
- В.Е. Котов, Сети петри, М.: Наука, 1984, с. 1-158;
- Ч. Хоар, Взаимодействующие последовательные процессы, Издательство Мир, М.: 1989, с.240;
- Журнал для разработчиков MSDN MAGAZINE, М.: 2008, 11 (83) (www.microsoft.com/rus/msdn/magazine);
- Д.В. Котляров, В.П. Кутепов, М.А. Осипов, Граф-схемное потоковое параллельное программирование и его реализация на компьютерных системах, Изд. РАН, Теория и системы управления, 2005,N1, с.75-96;
- V.P. Kutepov, V.N. Malanin, N.A.Pankov, An apporoach to the development of programming software for distributed computing and information systems, ICSOFT-08, International conference on software and data technologies, Porto, Portugal, 2008, p. 83-90;
- С.Е. Бажанов, В.П. Кутепов, Д.А. Шестаков, Язык функционального параллельного программирования и его реализация на кластерных системах, Изд РАН, Программирование, 2005, N5, с. 18-51.
- С.Е. Бажанов, М.М. Воронцов, В.П. Кутепов, Д.А. Шестаков. Структурный анализ и планирование процессов параллельного выполнениыя функциональных программ. Изв. РАН, Теория и системы управления, 2005, №6, с. 111-126.
- Б.А. Трахтенброт, Обогащение алгоритмических языков параллельными функциями. Авт. на соискание уч. степени к.ф.м.н., Киев, 1978.
- В.П. Кутепов, Об интеллектуальных компьютерах и больших компьютерных системах. Изв. РАН, Теория и системы управления, 1996, №5.
- В.П. Кутепов, Интеллектуальное управление процессами и загруженностью в вычислительных системах. Изв. РАН, 2007, №5, с.58-73.