

УДК 004.942

## **ОТОБРАЖЕНИЕ ВИРТУАЛЬНОЙ ПАРАЛЛЕЛЬНОЙ СИМУЛЯЦИОННОЙ МОДЕЛИ ДИНАМИЧЕСКОГО ОБЪЕКТА НА ЦЕЛЕВУЮ АРХИТЕКТУРУ**

*Васьковцов К.А.,*

*Донецкий национальный технический университет*

Параллельные машины и параллельное программирование в современном мире информационных технологий еще не получило должного развития, однако предлагаемые перспективы все больше и больше привлекают программистов в эту область. На данном этапе параллельные машины практически себя не окупают, и достаточно редки явления, когда достигаемая цель важнее затрат. В данном докладе будет рассмотрена основная проблема параллельного программирования – переход от виртуальной математической симуляционной модели (ВМСМ) к виду, реализуемому на имеющейся на данный момент архитектуре.

Под ВМСМ мы понимаем набор уравнений и условий, пригодных для численного решения, однако не накладывающих ограничений на решающую архитектуру, т.е. при наличии нескольких тысяч уравнений предполагается наличие такого же количества процессоров.

Условно проблему перехода (распределения) можно разбить на четыре компонента:

- архитектура машины – те физическое внутренне строение системы, оборудование, включающее в себя все компоненты (процессоры, сети связей, память, периферию);
- нагрузка (баланс загрузки) – включает в себя программы и обрабатываемые этими программами данные;
- механизм распределения – определяет способ, по которому будет происходить отображение процессов на процессоры, занимаемую память и тд;
- цель распределения – определяет приоритеты и цели в

программе.

Каждый из описанных компонентов влияет на скорость и точность получаемого решения. Причем, архитектура задает верхний, теоретически возможный предел скорости и точности. Остальные компоненты лишь снижают максимальные возможности архитектуры.

Рассмотрим каждый компонент в отдельности.

Архитектура машины в большинстве случаев описывается с помощью графов типа  $(P, E^P)$ , где  $P$  – множество процессорных элементов, а  $E^P$  – множество прямых межпроцессорных связей. Такие графы называются графами процессорных связей. Графы (топология) делятся на три класса — регулярные, иррегулярные (программируемые) и динамические.

Под регулярными графами понимаются классические модели графов:

- дерево;
- гиперкуб;
- кольцо;
- тор и другие.

Такие модели жестко задают программисту возможные связи между процессорными элементами и накладывают дополнительные ограничения на последующие этапы (в частности на выбор распределения).

Для иррегулярных графов различают общий граф и переменный граф. В первом случае связи графа должны задаваться явно. Во втором – подразумевается программируемая сеть связи, дающая возможность программисту самому определять связи между процессорами. Бывают такие задачи, при которых не важна структура связей. В этом случае граф сводится к одному из вышеописанных регулярных графов.

Также кроме топологии необходимо учесть и характеристики системы, а именно скорость обработки данных процессором и скорость обмена. Причем скорость обмена обычно является более весомым фактором, чем скорость обработки.

Немаловажным фактором является тип памяти. Она может быть локальной или общей. При локальной организации каждый процессор имеет доступ только в свою часть памяти, но зато с чрезвычайно высокой скоростью. Недостатком является малая емкость такой памяти, а также необходимость проводить все обмены через сети связи. Общая память может быть единой (сосредоточенной) или распределенной. При единой общей памяти каждый процессор имеет одинаковое время доступа к каждому участку памяти. Недостатком является механизм блокировок, блокирующий доступ для чтения и записи для используемого в текущий момент участка памяти, но при этом обеспечивающий целостность данных. При распределенной общей памяти каждый процессор имеет свою память, к которой он имеет доступ на высоких скоростях, однако процессор также имеет доступ и к памяти остальных процессоров, но со снижением скорости.

Последним критерием архитектуры является ее динамика. Обычно архитектуру рассматривают как статический объект. Для параллельных систем это не совсем верно. Система может динамически перестраиваться в критических ситуациях, таких как отказ компонента, перегрузка мощностей или перестраивание сети связи, не предусмотренное программистом.

Таким образом, получаем модель архитектуры, изображенный на рис.1.

Баланс загрузки представляет собой описание загруженности целевой системы задачами определенного типа и сложности. Построение такого баланса, а также выбор одного из его типов является важной задачей, поскольку перегрузка системы приведет к отказу в обслуживании или задержке решения, а минимальная загрузка окажется дорогой в ценовом коэффициенте по отношению к выбранной архитектуре.

Для построения баланса загрузки строится модель нагрузки. В общем случае модель состоит из пяти компонентов:

- способ использования целевой системы – однопрограммный или мультипрограммный;



Рисунок 1 – Модель архитектуры машины

- динамика загрузки – статична или динамически зависящая от внешних факторов;
- возникновение задач – централизованно либо децентрализованно.
- однородность – однородная загрузка, либо с пиками и спадами на различных этапах программы;
- программная модель.

Третьим компонентом является собственно распределение процессов на процессоры. Рассматривать распределение необходимо исходя из трех взаимосвязанных элементов:

- множество процессоров (P);
- множество параллельных программ (A);
- множества процессов, из которых состоят параллельные программы (T).

В общем случае распределение программ на процессы - это функция вида

$$\varphi: A \rightarrow f(P)$$

Одна программа может занимать несколько процессоров. Распределение программ на процессоры может быть выполнено обособленно. Таким считается распределение, при котором множества процессоров, занятых разными программами, не пересекаются. Иначе распределение считается выполненным «внахлест».

По отношению к топологии процессоров распределение может быть плотным или распыленным. При плотном распределении процессы одной программы выполняются на соседних в топологии процессорах, вследствие чего упрощается программирование связей между процессами и уменьшается время передачи. При распыленном наоборот.

При учете структуры программы распределение процессов может быть инъекционным либо сдерживающим. При первом типе каждый процесс программы получает в распоряжение свой отдельный процессор. При втором типе возможно последовательное выполнение нескольких процессов на одном процессоре.

Как упоминалось ранее, распределение может учитывать изменение загрузки и/или архитектуры системы. Поэтому различают консервативное и агрессивное распределения. Собственно консервативное или статическое – привязанное и неизменное распределение процессов по процессорам. Агрессивное включает в себя миграцию и вытеснение процессов.

Последней частью проблемы является выбор цели распределения. Те определения базового или основополагающего критерия, по которому определяется допустимость или недопустимость выбранного распределения. Оба аспекта рассматриваются в двух классах цели – оптимизация и ограничения.

Под ограничениями понимаются различные внешние и внутренние требования, накладываемые на модель, структуру

программы и распределение процессов ранее описанными моделями (архитектура, баланс нагрузки) или внешними факторами (необходимая точность, время выполнения). Под оптимизацией понимается извлечение наибольшей выгоды при наименьших затратах. Чаще всего стараются оптимизировать для уменьшения времени решения, поскольку чаще всего интересует результаты расчетов и моделирования, а не промежуточные этапы.

На основании всего вышеперечисленного выбирается алгоритм распределения. Чаще всего проблему распределения рассматривают как проблему оптимизации по одному или нескольким компонентам проблемы. Отсюда вытекает критерий оптимальности алгоритма – алгоритм считается оптимальным, когда гарантирует выполнение сформулированного критерия. Иначе он считается субоптимальным. Под субоптимальными понимаются алгоритмы, которые приводят к хорошим результатам, однако вносящие какую-либо погрешность в решение (аппроксимация, эвристика).

Для выбора алгоритма так же важно учитывать его структуру и методы. В качестве метода алгоритма используются теория графов, математическая оптимизация и другие.

Под структурой понимается способ управления ходом алгоритма – централизованно или децентрализованно.

Таким образом, на основании изложенного можно сделать вывод, что одной из основных задач является разработка и оптимизация методов и алгоритмов перехода от виртуальной параллельной модели к ее программной реализации для конкретной архитектуры. Наиболее необходимым является разработка автоматизированной системы расчета отображения моделей как минимум на существующие архитектуры и сети связей.

## Литература

- [1] Heiss, Hans-Ulrich: Prozessorzuteilung in Parallelerchnerh. Munnheim, Leipzig, Zьrich. 1994