ФОРМАЛЬНАЯ СПЕЦИФИКАЦИЯ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ В СИСТЕМАХ С АРХИТЕКТУРОЙ ТИПА ГИПЕРКУБ

С.В. Панков

Южный Федеральный университет, Ростов-на-Дону

Рассматривается класс распределенных алгоритмов типа SPMD (Single Program Multiply Data), реализуемых на параллельных вычислительных системах с гиперкубовой архитектурой. Исследуются схемы взаимодействия процессов, характеризующиеся высокой степенью регулярности.

Демонстрируется подход к формальной спецификации поведения распределенных алгоритмов на базе формализма *L*-программ. Этот формализм включает в себя логико-математическую модель параллельных вычислений – *L*-программы (описанные впервые в [1]) и средства анализа *L*-программ (разработанные в [2]). Подход направлен на автоматизацию анализа свойств распределенных алгоритмов как функционального характера, так и логического, типа отсутствия тупиков. Он обеспечивает компактность спецификации, независимость ее вида от числа процессов, что влечет независимость анализа системы от ее размера.

1. Распределенные вычисления в гиперкубе

Распределенный алгоритм задает систему из 2^n процессов, развивающихся на вычислительных узлах – вершинах единичного гиперкуба пространства R_n (где $n>0$). Каждый процесс работает над собственной памятью и может взаимодействовать только с n другими процессами по жестко закрепленным за ними *линиям связи* с номерами от 1 до n (ребрам гиперкуба) через передачу сообщений синхронного типа. Набор координат $\overline{u}=<u_1,..,u_n>$ (где $u_i \in \{0,1\}$ для всех $1 \le i \le n$) произвольной вершины гиперкуба задает *имя процесса*, соответствующего этой вершине.

Гиперкубовую структуру связей будем задавать с помощью функции *connect*, которая определяется следующим образом: для имен процессов $\overline{u}$, $\overline{a}$ и номера линии связи i справедливо $\overline{a} = connect(\overline{u}, i)$ тогда и только тогда, когда наборы $\overline{u}$ и $\overline{a}$ совпадают поэлементно, за исключением i-го элемента.

В качестве демонстрационного примера рассмотрим распределенный алгоритм *MAX*, который находит максимум из 2^{n+1} чисел (чисел, содержащихся в переменных V1 и V2 каждого из 2^n процессов). Сначала каждый процесс находит максимум из своих значений переменных V1 и V2 в свою переменную Vmax. Затем каждый процесс с именем вида $\overline{u}=<1,u_2,..,u_n>$ передает значение своей переменной Vmax процессу с именем $\overline{a}=<0,u_2,..,u_n>$ (где $u_i \in \{0,1\}$ для всех $2 \le i \le n$) по линии связи с номером 1. При этом процесс $\overline{a}$ принимает данное значение в переменную V1 и переопределяет значение своей переменной V2 на значение переменной Vmax. На этом заканчивается первая *волна* вычислений в гиперкубе размерности n. Далее процессы с именами вида $\overline{u}=<1,u_2,..,u_n>$ (где $u_i \in \{0,1\}$ для всех $2 \le i \le n$) завершают работу.

Затем каждый процесс с именем вида $\overline{u}$ =<0,u_2,..,u_n> (где $u_i \in \{0,1\}$ для всех $2 \le i \le n$) находит максимум из значений переменных V1 и V2 в переменную Vmax. После этого каждый процесс с именами вида $\overline{u}$ = <1,1,u_3,..,u_n> передает значение переменной Vmax процессу с именем $\overline{a}$ = <0,0,u_3,..,u_n> (где $u_i \in \{0,1\}$ для всех $3 \le i \le n$) по линии связи с номером 2. Процесс $\overline{a}$ принимает это значение в переменную V1 и переопределяет значение переменной V2 на значение своей переменной Vmax. На этом заканчивается вторая волна вычислений в гиперкубе размерности n-1, и т.д. После n-й волны вычислений работающим остается один процесс $\overline{O}$ ($\overline{O}$ – имя процесса, состоящее только из нулевых элементов). Этот процесс находит максимум из значений переменных V1 и V2 в переменную Vmax (это и будет максимум всех исходных чисел) и завершает работу всего распределенного алгоритма.

Взаимодействие процессов осуществляется на основе запроса на прием сообщения ?(x,y,v) и запроса на передачу сообщения !(x,y,v'). Здесь x (y) – имя запрашивающего (запрашиваемого) процесса, v – переменная, в которую должно быть записано полученное сообщение, а v' – переменная, значение которой должно быть передано.

2. Формальная спецификация распределенных вычислений

Формальная спецификация параллельных вычислений на основе *L*-программ предполагает разработку языка предметной области – *L* – для исследуемого класса параллельных систем. При этом *L* является проблемно-ориентированным языком логики предикатов первого порядка. Определим язык *L* для рассматриваемого здесь класса распределенных алгоритмов (в том числе, для алгоритма *MAX*).

2.1. Язык предметной области *L*

Сорта: *PROC* – множество имен процессов <u_1,..,u_n>, где $u_i \in \{0,1\}$ для всех $1 \le i \le n$;

NUM – множество номеров линий связи {1,2,...,*n*} между процессами;

DATA – числовые данные;

VAR – множество имен переменных процесса {V1, V2, Vmax,I}, переменная I будет хранить номер используемой в данный момент линии связи между процессами.

Функции:

s: $PROC \times VAR \to DATA$ – представляют состояние памяти процесса; s_0 – начальное состояние;

connect: $PROC \times NUM \to PROC$ – задает гиперкубовую структуру (определена в разделе 1);

_ [_]: $PROC \times NUM \to \{0,1\}$ – выбирает в имени процесса координату с заданным номером;

max: $DATA \times DATA \to DATA$ – выбирает максимальное из двух заданных значений;

next: $PROC \to \{1,2\}$ – вспомогательная функция, используемая для задания порядка действий в алгоритме *MAX* (1 означает, что процесс может находить максимум своих значений переменных V1 и V2; 2 – процесс может активизировать запрос на взаимодействие).

Предикаты:

?: $PROC \times PROC \times VAR$ – выражает активность запроса на прием сообщения;

!: $PROC \times PROC \times VAR$ – выражает активность запроса на передачу сообщения;

Язык *L* также содержит обычные арифметические функции и отношения. Знаками $\wedge$, $\vee$, $\neg$, $\to$, $\forall$ и $\exists$ будем обозначать обычные логические связки и кванторы.

Переменные

x,y: PROC; v,v',i:: DATA; u,u': VAR.

Будем использовать запись max(v,v')=u как сокращение формулы (v≥v'∧u=v ∨ v<v'∧u=v'), определяющей функцию max.

2.2. Формализм *L*-программ

Произвольная *L*-программа задается конечной совокупностью правил вида *условие* ⇒ *действие*, где *условие* – произвольная *L*-формула, *действие* – *L*-формула специального вида. Она определяет систему переходов $S=(Q,T)$, где Q – множество *L*-структур (т.е. алгебраическая система, определяемая носителем и интерпретацией функциональных и предикатных символов языка *L* на этом носителе), а $T \subseteq Q \times Q$ – отношение перехода (или отношение непосредственной достижимости). Например, *L*-структура q из Q может описывать состояние клеточного автомата, здесь она будет описывать состояние памяти процессов и активность или неактивность запросов на взаимодействие. Переход из состояния q в другое состояние заключается в переопределении функций и отношений *L*-структуры q посредством асинхронного исполнения действий правил. При этом условия правил определяют, какое переопределение возможно. Такая организация *L*-программ привносит в логику первого порядка динамизм, необходимый для описания поведения параллельных вычислений. *L*-программа завершает свою работу в *L*-структуре, на которой становятся ложными условия всех правил.

2.3. Специфицирующая *L*-программа

Начальное состояние удовлетворяет следующему предусловию **P**:

P: $\forall x(\exists v s(x,\mathrm{V1})=v \wedge \exists v s(x,\mathrm{V2})=v) \wedge \forall x s(x,\mathrm{I})=0 \wedge$
$\forall x,y,v(\neg !(x,y,v) \wedge \neg ?(x,y,v)) \wedge \forall x(next(x)=1)$.

Эта формула утверждает, что у каждого процесса переменные V1 и V2 определены, значение переменной I равно 0 (т.е. ни одна из линий связи в данный момент не используется для передачи сообщений), нет ни одного активного запроса на прием или передачу сообщений, и каждый процесс может приступить к поиску максимума значений своих переменных.

Синхронная *L*-программа

1) $\$\ x,v,i \quad next(x)=1 \wedge v=max(s(x,\mathrm{V1}),s(x,\mathrm{V2})) \wedge i=s(x,\mathrm{I})+1 \wedge$
$(s(x,\mathrm{I})=0 \vee s(x,\mathrm{I})\geq 1 \wedge s(x,\mathrm{I})\leq n \wedge x[s(x,\mathrm{I})]=0) \Rightarrow s(x,\mathrm{Vmax})=v \wedge s(x,\mathrm{I})=i \wedge next(x)=2$

2) $\$\ x,y,v \quad next(x)=2 \wedge x[s(x,\mathrm{I})]=1 \wedge y=connect(x,s(x,\mathrm{I})) \Rightarrow !(x,y,\mathrm{Vmax})$

3) $\$\ x,y,v \quad next(x)=2 \wedge x[s(x,\mathrm{I})]=0 \wedge y=connect(x,s(x,\mathrm{I})) \wedge v=s(x,\mathrm{Vmax}) \Rightarrow$
$?(x,y,\mathrm{V1}) \wedge s(x,\mathrm{V2})=v$

4) $\$\ x,y,u,u' \quad ?(x,y,u) \wedge !(y,x,u')) \wedge y=connect(x,s(x,\mathrm{I})) \wedge v=s(y,u') \Rightarrow$
$s(x,u)=v \wedge \neg !(y,x,v) \wedge \neg ?(x,y,v') \wedge next(x)=1$

Первое правило может исполниться для процесса x, либо в начальный момент (когда справедливо $s(x,\mathrm{I})=0$), либо когда номер текущей линии связи находится в диапазоне от 1 до n (не все еще линии связи были использованы) и в имени процесса координата, соответствующая текущей используемой линии связи, равна 0. При этом вычисляется максимальное значение и записывается в переменную Vmax, увеличивается на 1 номер линии связи и разрешается активизация запросов на взаимодействие для процесса x (установлением $next(x)=2$). Благодаря синхронизатору $\$\ x,v,i$, это происходит одновременно для всех процессов, удовлетворяющих условию правила.

Второе правило может исполниться для процесса *x*, только после первого правила, если в имени процесса координата, соответствующая текущей используемой линии связи, равна 1. Исполнение правила активизирует запрос на передачу значения переменной Vmax процессу *y*, связанному с процессом *x* линией связи с номером *s*(*x*,I). Синхронизатор $ *x,y,v* вынуждает это выполнить одновременно для всех процессов, удовлетворяющих условию правила.

Третье правило может исполниться для процесса *x*, только после второго правила, если в имени процесса координата, соответствующая текущей линии связи, равна 0. При этом активизируется запрос на прием сообщения в переменную V1 процессу *y*, связанному с процессом *x* линией связи с номером *s*(*x*,I), а также переопределяется значение переменной V2 установлением *s*(*x*,V2)=*s*(*x*,Vmax). Синхронизатор $ *x,y,v*, так же, как и в предыдущих правилах, вынуждает выполняться это правило одновременно для всех процессов, удовлетворяющих условию правила.

Четвертое правило может исполниться для процессов *x* и *y*, связанных линией связи с номером *s*(*x*,I), если процесс *x* запрашивает процесс *y* на прием сообщения, а процесс *y* запрашивает процесс *x* на передачу сообщения. Выполнение правила реализует передачу сообщения от процесса *y* процессу *x*, снимает активность запросов на взаимодействие, и разрешает начать выполнение следующей волны вычислений алгоритма *MAX*. Благодаря синхронизатору $ *x,y,u,u'*, это происходит одновременно для всех пар процессов *x* и *y*, для которых справедливо условие правила.

Асинхронная *L*-программа легко получается из синхронной *L*-программы, удалением синхронизаторов, стоящих перед ее правилами. При этом правильность работы *L*-программы не пострадает. Каждое правило на одном шаге работы асинхронной *L*-программы будет выполняться для произвольного множества процессов, удовлетворяющих условию этого правила. Работа этих *L*-программ остановится, когда ни один из процессов не будет удовлетворять ни одному из условий правил.

Необходимо отметить, что вид специфицирующей *L*-программы не зависит от размерности гиперкуба *n*, что позволяет более эффективно применить методы анализа свойств параллельных систем из [2]. В частности, речь идет о свойстве правильности системы с точки зрения вычисляемой функции. Так, в случае алгоритма *MAX* такая функция в качестве значения должна вычислять *s*($\overline{\mathbf{O}}$,Vmax) в качестве результата алгоритма, т.е. максимума всех исходных 2^{n+1} чисел. Также речь идет о свойстве нетупиковости алгоритма, требующего, чтобы заключительное, достижимое из предусловия **P** состояние алгоритма удовлетворяло формуле $\forall x,y,v(\neg !(x,y,v)\wedge\neg ?(x,y,v))$. Эта формула требует, чтобы в заключительном состоянии не было ни одного активного запроса на взаимодействие.

Литература

1. Крицкий С.П. Модель асинхронных вычислений в структурах и языки программирования // Методы трансляции. Ростов-на-Дону. 1981. С. 92-100.
2. Крицкий С.П., Панков С.В. О верификации асинхронных программ продукционного типа // Программирование. 1994. № 5. С. 40-52.