Entendendo Papadimitriou e Yannakakis

Carlos Morais de Oliveira Filho

16/12/2009

MAC0461 - Introdução ao Escalonamento e Aplicações

Professor: Alfredo Goldman

Resumo

Este trabalho é um resumo de um dos trabalhos mais importantes em escalonamento de aplicações (Papadimitriou e Yannakakis, 1990). O algoritmo apresentado é típico para máquinas paralelas com memória distribuída. Aqui as principais demonstrações e definições do citado artigo são detalhadas, visando fácil compreensão. Desse modo, a importância do trabalho é evidenciada.

1 Introdução

Um dos problemas mais importantes em programação paralela e distribuída é o escalonamento com atraso de comunicação, pois muitos problemas podem ser modelados dessa forma. O problema que se deseja resolver é $P_\infty|prec; p_j; c_{jk}; dup|C_{max}$, onde:

- C_{max} indica que o objetivo é minimizar o *makespan*, ou seja, o tempo de término da última tarefa.
- *prec* indica que há um grafo dirigido acíclico (DAG - Directed Acyclic Graph) representando as relações de precedência entre as tarefas.
- *dup* indica que pode há a possibilidade de executar a mesma tarefa mais de uma vez, para evitar atrasos de comunicação.

- p_j são os tempos de execução de cada tarefa.
- c_{ij} são os atrasos de comunicação entre o término da tarefa p_i e o início da tarefa p_j em outra máquina.
- P_∞ indica que estão disponíveis um número ilimitado de processadores idênticos (que executam as tarefas com mesma velocidade).

O número de máquinas não foi considerado como sendo um parâmetro importante, pois em geral a principal medida de complexidade do DAG é sua profundidade, e não sua largura. O parâmetro considerado mais importante no artigo foi o tempo de comunicação, que em geral é muito grande comparado aos tempos de processamento.

Este trabalho foi motivado pela dificuldade dos alunos da disciplina *MAC0461/MAC5758 — Introdução ao Escalonamento e Aplicações* em entender as demonstrações presentes no artigo analisado. As provas apresentadas aqui são formas ligeramente modificadas e extensamente detalhadas das demonstrações originais, visando fácil compreensão por parte dos alunos e fácil explicação por parte do professor (que provavelmente usará este trabalho da próxima vez que lecionar a disciplina :-).

2 Demonstrações

O artigo é estruturado da forma descrita a seguir. É analisada uma versão simplificada do problema, onde todas as tarefas têm tempo de execução unitário e os atrasos de comunicação são iguais a um inteiro τ. Demonstra-se que tal problema é NP-completo. Uma função $e(p_j)$ é definida e se demonstra que ela é um ótimo para o tempo de execução da tarefa p_j. Um algoritmo de aproximação é produzido, onde todas as tarefas são alocadas em tempo $2e(p_j)$. Esse algoritmo é então ampliado para resolver o problema geral.

2.1 Definição do problema

É dado um grafo dirigido acíclico $D = (U, A)$. Os vértices em U representam as tarefas, todas com tempo de execução unitário. Os arcos em A representam as precedências

funcionais e temporais. É dado também $\tau \in \mathbb{N}$ que representa o tempo pago na comunicação, ou seja, o tempo que leva para que uma informação produzida em uma máquina possa ser usada em outra.

Definição: Seja um conjunto de triplas $S \subset U \times \mathbb{N} \times \mathbb{N}$. Uma tripla $(u, p, t) \in S$ indica que a tarefa u é alocada no processador p no instante t, em unidades de processamento. S é um escalonamento se as seguintes condições forem verdadeiras:

- $\forall u \in U, \exists (u, p, t) \in S$ (todas as tarefas foram alocadas)
- $(u, p, t) \in S \wedge (u', p, t) \in S \Rightarrow u = u'$ (não existem duas tarefas diferentes alocadas no mesmo instante e no mesmo processador)
- $(u, v) \in A \wedge (v, p, t) \in S \Rightarrow (\exists (u, p, t') \in S, t' \leq t-1) \vee (\exists (u, p', t') \in S, t' \leq t-1-\tau)$ (os predecessores de uma tarefa foram alocados na mesma máquina pelo menos 1 unidade antes ou em outra máquina pelo menos $\tau + 1$ unidades antes)

O problema consiste em, dado $T_{max} \in \mathbb{N}$, decidir se existe ou não um escalonamento S viável, tal que nenhuma tarefa é alocada depois de T_{max}, ou seja, $\forall (u, p, t) \in S, t \leq T_{max}$.

2.2 Redução do clique

Nesta seção demonstraremos que o problema definido na seção anterior é NP-completo através de uma redução do problema do clique. O clique é um problema NP-completo bastante conhecido na literatura. Consiste em, dado um grafo $G = (V, E)$ e dado $k \in \mathbb{N}$, decidir se existe ou não em G um clique de tamanho k. Um clique é um subgrafo $G' = (V', E'), V' \subset V, U' \subset U$ tal que $(v_1, v_2) \in V' \times V' \Rightarrow (v_1, v_2) \in E'$. Um clique de tamanho k tem $\binom{k}{2}$ arestas.

TEOREMA: Decidir se existe escalonamento viável é um problema NP-completo.

Prova: Reduziremos o problema do clique ao problema do escalonamento. Dados o grafo $G = (V, E)$ e $k \in \mathbb{N}$, devemos construir $D = (U, A)$, τ e T_{max} tais que exista escalonamento viável se e somente se G tem clique de tamanho k.

Definimos $\tau = |E|(k-1) + \binom{k}{2}$, $T_{max} = |E|(k+1)$ e construimos D da seguinte forma:

- $U = \{u_{v,i} | v \in V, i \in \{1, ..., |E|\}\} \cup \{c_e | e \in E\} \cup \{t\}$

- $A = \{(u_{v,i}, u_{v,i+1}) | i \in \{1, ..., |E| - 1\}\} \cup \{(u_{v,|E|}, c_e) | v \in e\} \cup \{(c_e, t) | e \in E\}$

Ou seja, para cada vértice $v \in V$ existe em D um caminho de tamanho $|E|$. Para cada aresta $e \in E$ existe em U um vértice c_e. Existem em A arcos que saem do último vértice do caminho relativo a v para c_e se v for adjacente a e. Além disso, existe em U um vértice t e arcos que partem de todos os c_e para t.

(1)$\exists S \Rightarrow \exists clique$

Para que haja escalonamento S que cumpre T_{max} é suficiente e necessário que t seja escalonado no máximo no instante T_{max}, pois ele é descendente de todos os outros vértices de U.

Seja $d = |E|k + \binom{k}{2}$. Considere os seguintes fatos:

(i) Todos os $c_e, e = [v_1, v_2]$ que foram escalonados antes de d devem ter os caminhos relativos a v_1 e v_2 executados no mesmo processador. Caso c_e receba comunicação de outra máquina que executou um caminho de tamanho $|E|$, não poderá ser alocado antes de $|E| + \tau = d$

(ii) Todos c_e devem ser executados no mesmo processador que t. Um c_e que execute antes de d deve executar 2 caminhos na mesma máquina de acordo com (i); portanto, o tempo que levaria para a informação chegar de outra máquina é pelo menos $2|E|+1+\tau = |E|(k+1)+1+\binom{k}{2} = T_{max} + \binom{k}{2} + 1 > T_{max}$. Um c_e que execute depois de d tem ainda menos tempo.

Portanto, a máquina que executa t em T_{max} deve ser ocupada com todos os $|E|$ c_e, restando apenas $|E|k$ unidades de tempo, onde, segundo (i), deve ser possível alocar todos os caminhos adjacentes aos $\binom{k}{2}$ c_e executados antes de d. Como o escalonamento é possível, então as $\binom{k}{2}$ arestas são adjacentes a apenas k vértices, e portando existe clique de tamanho k em G.

(2)$\exists clique \Rightarrow \exists S$

Caso exista um clique, um escalonamento possível é:

- execute no processador 1 todos os caminhos relativos aos vértices do clique, então todos os c_e correspondentes às arestas do clique, terminando no instante d;

- execute cada um dos caminhos relativos aos vértices que não estão no clique em uma máquina diferente e pague τ para comunicar com o processador 1. Toda informação chega no processador 1 no instante d;
- execute no processador 1 as arestas que não são do clique, terminando em T_{max};
- no processador 1, execute t no instante T_{max}; $\square$

2.3 Função de otimalidade

Dado o DAG $D = (V, A)$, definiremos $e : V \to \mathbb{N}$ que pode ser computada percorrendo o DAG em profundidade. A definição é feita indutivamente:

(1) Se v é fonte, $e(v) = 0$.

(2) v não é fonte. Seja a o número de ancestrais de v. Ordenar os ancestrais em ordem decrescente de e: $e(u_1) \geq e(u_2) \geq ... \geq e(u_a)$. Seja $k = min\{a, \tau + 1\}$. Então $e(v) = e(u_k) + k$

Vamos provar que $e(v)$ é um limite inferior para o tempo de execução de v.

LEMA: $\forall S$ escalonamento, $(v, p, t_v) \in S \Rightarrow t_v \geq e(v)$

Prova: A prova é feita por indução:

(1) Base: v é fonte. Pela definição de escalonamento $t_v \geq 0 = e(v)$.

(2) Passo: v não é fonte. Sejam u_i os ancestrais de v ordenados de acordo com a definição de $e(v)$. A hipótese indutiva é que $(u_i, p, t_{u_i}) \in S \Rightarrow t_{u_i} \geq e(u_i)$.

Vamos supor, por absurdo, que $t_v < e(v)$. Tome os k primeiros ancestrais de v: $e(u_1) \geq ... \geq e(u_k)$. Então $t_v < e(v) = e(u_k) + k \leq e(u_k) + \tau + 1 \leq e(u_i) + \tau + 1$, para $i \in \{1, ..., k\}$. Logo, pela hipótese indutiva, $t_v < t_{u_i} + \tau + 1$, ou seja, os k primeiros ancestrais de v executam menos que $\tau + 1$ unidades de tempo antes de v e portanto precisam ser executados no mesmo processador. Para que seja possível executar todos os k ancestrais no mesmo processador, é necessário que $t_v \geq t_{u_k} + k$. Usando de novo a hipótese indutiva, $t_v \geq e(u_k) + k = e(v)$, o que contradiz a hipótese do absurdo. Portanto $t_v \geq e(v)$. $\square$

2.4 Algoritmo de aproximação

Vamos provar que sempre é possível obter escalonamento com fator de aproximação 2.

LEMA: $\exists S$ escalonamento tal que $(v, p, t_v) \in S \Rightarrow t_v \leq 2e(v)$

Prova: De novo, prova por indução:

(1) Base: v é fonte. Como v não tem ancestrais, pode ser escalonado em $t_v = 0 \leq 0 = 2e(v)$.

(2) Passo: v não é fonte. Sejam u_i os ancestrais de v ordenados de acordo com a definição de $e(v)$. A hipótese indutiva é que é possível encontrar S onde vale $(u_i, p, t_{u_i}) \in S \Rightarrow t_{u_i} \leq 2e(u_i)$.

(caso 1) $a \leq \tau + 1 \Rightarrow k = a \Rightarrow e(v) = e(u_a) + a = a$. Nesse caso, um escalonamento possível é escalonar todos os ancestrais de v e então o próprio v no mesmo processador.

(caso 2) $a > \tau+1 \Rightarrow k = \tau+1 \Rightarrow e(v) = e(u_{\tau+1})+\tau+1 \Rightarrow e(v)-\tau-1 = e(u_{\tau+1}) \geq e(u_i)$ para $i \geq \tau + 1$. Logo, $2e(v) - 2\tau - 2 \geq 2e(u_i), i \geq \tau + 1$. Usando a hipótese indutiva, $t_{u_i} \leq 2e(v) - 2\tau - 2, i \geq \tau + 1$. Ou seja, todos os ancestrais de v com exceção dos τ primeiros podem ser executados em tempo $2e(v) - 2\tau - 2$. Pagando a comunicação, a informação chega em tempo $2e(v) - \tau - 1$. Então há tempo de executar os τ primeiros no mesmo processador que v, executando v em tempo $2e(v)$. □

A demonstração desse último lema sugere o seguinte algoritmo de escalonamento:

1 Escalonar v em $2e(v)$;

2 Escalonar os τ primeiros ancestrais de v logo antes, na mesma máquina que v;

3 As informações dos outros ancestrais $u_i, i \geq \tau + 1$ poderá ser recebida do processadores que os escalonaram em tempo $2e(u_i)$ usando os passos 1 e 2;

3 Considerações finais

Na generalização do algoritmo, a função $e(v)$ é definida em termos de $x(v)$ (tempo de execução do vértice v) e $\tau(u, v)$ (tempo de comunicação no arco (u, v)). É mencionado no

final do artigo que, no caso onde não existe duplicação (ou recomputação), embora não se conheça algoritmo de aproximação, uma prova de NP-completude estabelece que não há aproximação com fator melhor que 2 (a não ser que $P = NP$). Desse ponto de vista, fica evidente que uma aproximação de fator 2 é muito boa.

Observa-se que o algoritmo produzido gera muitas duplicações, mas é possível aplicar um outro algoritmo à solução encontrada para diminuir o número de repetições, o que é possível na grande maioria dos casos.

Referências

PAPADIMITRIOU, Christos H.; YANNAKAKIS, Mihalis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, v. 19, n. 2, p. 322–328, Abril 1990.