멀티메쏘드 디스패치 구현을 위한 향상된 압축알고리즘 개발

장문종[0] 이봉재 조선구
한국전력공사 전력연구원
{mjjang, bjlee, csk9306}@kepri.re.kr

Development of the efficient compression algorithm for the multi-method dispatch implementation

Moonjong Jang[0] Bong-Jae Lee, Sun-Ku Cho
KEPCO KEPRI

요 약

대부분의 객체 지향 언어들은 여러 가지 연산을 구현하기 위한 방법으로 메쏘드(method)를 이용한다. 메쏘드는 생성함수(generic function)의 타입에 관련된 행동을 정의한다. 이런 메쏘드들이 서로 구별되는 인수를 이용해서, 생성 함수가 프로그램 수행시에 불려질 때 인수의 타입에 따라 어느 메쏘드를 수행할지 결정한다. 이런 메쏘드의 선택 과정은 가장 적용되기 가까운 메쏘드(most specific applicable method, MSA)를 구하기 위해서 단일의 인수를 사용할 것이냐, 모든 인수를 활용할 것이냐에 따라 단일 메쏘드와 다중 메쏘드로 구분된다. 이런 메쏘드를 선택하기 위해서 구현하는 방법으로는 캐싱이나 인라인, 디스패치 테이블, 컬러링 기법이 있다. 본 논문에서는 디스패치 테이블을 이용한 기법에서 공간의 낭비없이 효율적으로 멀티메쏘드 디스패치를 구현할 수 있는 점진적 압축알고리즘을 제시한다.

1. 서 론

대부분의 객체 지향 언어들에서는 여러 가지 연산을 구현하기 위한 방법으로 메쏘드(method)를 이용한다. 메쏘드는 생성 함수(generic function)의 타입에 관련된 행동을 정의한다. 이런 메쏘드들이 서로 구별되는 인수를 이용해서, 생성 함수가 프로그램 수행시에 불려질 때 인수의 타입에 따라 어느 메쏘드를 수행할지 결정한다.

이런 메쏘드의 선택 과정은 가장 적용되기 가까운 메쏘드(most specific applicable method, MSA)를 구하기 위해서 오직 하나의 인수만을 이용하느냐 또는 모든 인수를 이용하느냐에 따라서 단일 메쏘드(single method)와 다중 메쏘드(multi-method)로 구분된다.

단일 메쏘드의 경우 메쏘드를 선택하는 기법으로 캐싱이나 인라인 같은 방법들이 있다. 이들 방법을 이용할 경우 상수시간만에 메쏘드를 선택할 수 없다. 그래서, 상수시간만에 메쏘드를 선택할 수 있는 디스패치 테이블 기법이 나왔다. 이 방법은 C++에서 Stroustrup[1]에 의해 소개된 기법을 이용한 것이다.

그러나, 이런 디스패치 테이블은 공간 낭비가 심하다. 타입과 생성함수의 모든 쌍에 대해서 하나의 테이블 엔트리를 차지하기 때문이다. 그러므로, 컬러링(coloring)이라는 디스패치 테이블의 크기를 줄이는 압축 기법이 개발되었다[2][3]. 이 기법의 기본 생각은 같은 타입에는 절대로 수행되지 않는 함수들의 그룹을 식별해서, 이런 함수 그룹은 테이블에서 같은 인덱스를 가지도록 하며, 이런 방법으로 행들을 합치게 된다.

다중 메쏘드의 경우에는 단지 하나의 인수가 아니라 모든 인수의 타입을 이용해서 메쏘드 선택을 한다. 따라서, 다중 메쏘드는 단일 메쏘드의 일반화된 형태로 볼 수 있다. Common-Loops과 CLOS에서 처음으로 도입된 다중 메쏘드는 이제 Polyglot[4]이나 Kea, Cecil[5] 같은 최근의 객체 지향 언어에서 중요한 사양이 되고 있다. 심지어는 표준화된 데이터베이스 질의 언어를 위한 SQL3 제안서의 일부로도 통합되어 있다.

그러나, 디스패치 테이블에 의해 야기되는 공간 낭비 문제는 다중 메쏘드의 경우에는 더 심각해진다. 이런 공간 낭비 문제를 해소하기 위해서 Amiel[6]은 다중메쏘드에서 디스패치 테이블이 차지하는 공간을 줄이는 알고리즘을 제시했다. 이 알고리즘을 통해 공간의 낭비를 상당히 줄일 수 있었으나, 충분히 모든 공간을 줄이지는 못 했다.

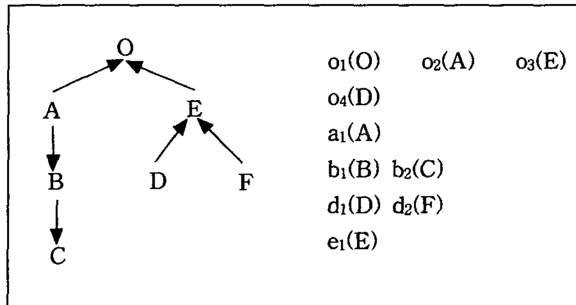
따라서, 본 논문에서는 다중 메쏘드의 경우에 디스패치 테이블을 압축하는 기존의 알고리즘을 향상시킨 알고리즘을 제시한다. 본 논문에서는 n차원의 디스패치 테이블을 각각의 생성함수에 대하여 만드는 알고리즘을 제시한다.

2 디스패치 테이블 기법

일반적으로 각각의 타입 T는 메쏘드 사전을 가지고 있다. 이 사전은 메쏘드와 T가 타겟 인수가 되는 각 생성함수를 연결시키는 역할을 한다. 이를 위한 방법은 여러 가지가 있으며 본 절에서는 디스패치 테이블 기법에

대해서 살펴본다.

디스패치 테이블 기법에서는 MSA 메쏘드가 모든 생성함수와 해당 타입의 쌍에 대해서 컴파일시에 미리 계산된다. 그리고, 그 결과는 디스패치 테이블에 저장된다. 그러므로, 모든 가능한 인보케이션에 해당하는 MSA 메쏘드들은 캐쉬과는 달리 항상 상수시간만에 접근하는 것이 가능하다.



[그림 1] 단일 메쏘드의 스키마

[예 1] 그림 1의 메쏘드에 대한 글로벌 디스패치 테이블은 표 1의 배열에 있다. 행에는 생성함수가 있고 열에는 타입이 있다.

[표 1] 글로벌 디스패치 테이블

	O	A	B	C	D	E	F
o	o_1	o_2	o_2	o_2	o_4	o_3	o_3
a	-	a_1	a_1	a_1	-	-	-
b	-	-	b_1	b_2	-	-	-
d	-	-	-	-	d_1	-	d_2
e	-	-	-	-	e_1	e_1	e_1

디스패치 테이블의 장점으로는 빠른 상수시간의 메쏘드 선택이 가능하다는 것이다. 각 타입이 자신의 단일 메쏘드 테이블(_vtbl로 유명하다)을 가지는 단일 상속 C++의 경우를 생각해 보자. MSA 메쏘드를 찾는 것은 목표로 하는 인수에 저장되어 있는 디스패치 테이블의 베이스 주소를 찾는 것이다. 그 후에, 생성함수의 인덱스를 이용해서 한 번에 배열의 해당 항목 접근을 수행한다. 따라서, 두 번의 인다이렉션(indirection)과 한 번의 접근으로 MSA 메쏘드를 찾을 수 있다.

3. 점진적 압축 알고리즘

본 절에서는 디스패치 테이블을 압축하기 위한 기준을 제시한다. 그리고, 제시된 기준을 이용하여 실제로 압축을 위한 알고리즘을 제시할 것이다. 그러기 위해서 먼저 새로운 용어들을 정의하고 이에 따른 각종 성질들을 나열한다. 그 후 이를 이용한 점진적 압축 알고리즘을 제시한다.

3.1 디스패치 테이블의 효율적 압축을 위한 주요 관점들

디스패치 테이블을 효율적으로 생성하기 위해서 행과 열을 검사하여 점진적으로 압축된 테이블을 만들 것이다. 이를 위해 다음과 같은 두 가지 관점을 이용한다.

- 어느 행과 열이 제거될 수 있는가?
- 어느 행과 열이 함께 그룹화될 수 있는가?

우선 제거 가능한 조건을 살펴보자.

- 디스패치 테이블 D의 i 번째 차원에 있는 타입 T의 행이 제거되려면,

For $\forall\,(T_1,\cdots,T_{i-1},T_{i+1},\cdots,T_n)\in\theta^{n-1}$

$\nexists$ MSA for $m(T_1,\cdots,T_{i-1},T,T_{i+1},\cdots,T_n)$

(θ : set of all types, $\theta^n=\theta^{n-1}\times\theta$)

이면 된다.

그리고, 그룹화가 될 수 있는 조건을 살펴보면 다음과 같다.

- 디스패치 테이블 D의 i 번째 차원에 있는 타입 T와 T'의 행이 합쳐지려면,

For $\forall\,(T_1,\cdots,T_{i-1},T_{i+1},\cdots,T_n)\in\theta^{n-1}$

MSA($m(T_1,\cdots,T_{i-1},T,T_{i+1},\cdots),=T_n)$

MSA($m(T_1,\cdots,T_{i-1},T',T_{i+1},\cdots),T_n)$

(θ : set of all types, $\theta^n=\theta^{n-1}\times\theta$)

이면 된다.

3.2 정의 및 성질들

m_1부터 m_l까지의 메쏘드가 주어졌을 때 i 번째 능동타입은 다음과 같이 구할 수 있다.

(ith active type) $\mathrm{active}_i(\{m_1,\cdots,m_l\})$

① $T\in\mathrm{static}_i(\{m_1,\cdots,m_l\})$

② For $\forall$ Signature,
find $G_i(T_1,\cdots,T_{i-1},T_{i+1},\cdots,T_n)$

(loop) For $\forall$ T, such that
|filter_active(supertype(T))| ≥2,

If for $\forall\,T'\in\mathrm{active}_i(m_1,\cdots,m_l)$,
|filter_active(supertype(T))|
≠|filter_active(supertype(T'))| and,
∄unique T'=
precedence($G_i(T_1,\cdots,T_{i-1},T_{i+1},\cdots,T_n)$)

then $G_i(T_1,\cdots,T_{i-1},T_{i+1},\cdots,=T_n)$
$G_i(T_1,\cdots,T_{i-1},T_{i+1},\cdots,T_n)\cup\{T\}$,
$\mathrm{active}_i(m_1,\cdots,m_l)\ni$

$active_i(m_1, \cdots, m_l) \cup \{T\}$,
$chameleonic_i(m_1, \cdots, m_l) =$
$chameleonic_i(m_1, \cdots, m_l) \cup \{T\}$.
For $\forall T' \in subtype(T)$, goto (loop)

else For T', $passive_i(m_1, \cdots, m_l, T') = \cup \{T\}$

where filter_active({T1,…,Tn})=
{T1,…,Tn} $\cap active_i(m_1, \cdots, m_l)$

m_1부터 m_l까지의 메쏘드가 주어졌을 때 i 번째 수동타입은 다음과 같이 구할 수 있다.

(ith passive type) $passive_i(m_1, \cdots, m_l, T_a)$

① $\{Ti | Ti \le Ta,\ T_a \in active_i(m_1, \cdots, m_l)$
$| filter_active(supertype(Ti)) | < 2$,
$\nexists T'$ such that
$T' \in active_i(m_1, \cdots, m_l) T_i \le T' < T_a$

② For $\forall T \in descendant(T_a)$
such that $| filter_active(supertype(Ti)) | \ge 2$,
$\exists$unique T'=
$precedence(G_i(T_1, \cdots, T_{i-1}, T_{i+1}, \cdots, T_n)) \rightarrow$
$passive_i(m_1, \cdots, m_l, T') =$
$\cup passive_i(m_1, \cdots, m_l, T)$

임의의 타입 T가 주어졌을 때 타입 T의 능동타입은 다음과 같이 구할 수 있다.

(finding ith active type of T) Findi({m1,…,ml},T)

Findi({m1,…,ml},T) = Ta
such that $T_a \in active_i(m_1, \cdots, m_l)$
$T \in passive_i(m_1, \cdots, m_l, T_a)$

3.3 점진적 압축 알고리즘

이제 점진적 압축 알고리즘을 살펴보면 다음과 같다.

For each ith position of generic function m,

① Find $active_i(m_1, \cdots, m_l)$ and,
for each $T_a \in active_i(m_1, \cdots, m_l)$
find $passive_i(m_1, \cdots, m_l, T_a)$

② Give a unique index to each
passivei({m1,…,ml},Ta).

③ For $\forall$ T,
m_argi[T.type] =
index of passivei({m1,…,ml},Ta),
where Ta = Findi({m1,…,ml},T).

④ For $\forall$ signature($T_1, \cdots, T_n$)
$\in \Pi^n_{i=1} passive_i(\{m_i, \cdots, m_l\}, T_a)$,
Dc[m_arg1[O1.type],…,m_argn[On.type]]
= MSA(m(T1,…,Tn)).

모든 생성함수들의 i 번째 인수들에 대해서, 첫 번째로 $active_i(m_1, \cdots, m_l)$를 구하고 각각의 능동타입 Ta 에 대해서 $passive_i(m_1, \cdots, m_l, T_a)$를 구한다. 두 번째로 각각의 $passive_i(m_1, \cdots, m_l, T_a)$에 서로 다른 인덱스를 부여한다. 세 번째로 모든 T에 대해서 인수 배열의 해당 항목에 타입 T가 속한 $passive_i(m_1, \cdots, m_l, T_a)$의 인덱스를 저장한다. 네 번째로 모든 시그너처에 대해서 압축된 디스패치 테이블의 각 항목에 MSA를 저장한다.

4. 결 론

본 논문에서는 주로 객체 지향에 관한 관점에서 다중 메쏘드를 어떻게 하면 효율적으로 처리할 수 있을지에 초점을 두었다. 즉, 객체 지향 언어에서 메쏘드의 디스패치가 발생할 경우에 이를 어떻게 처리하여 가장 적용하기에 가까운 메쏘드를 찾느냐에 중점을 두고 있었다.
그렇지만, 이와 유사한 형태로 존재하는 다른 응용 프로그램들에도 충분히 적용이 가능하다. 다중 메쏘드를 지원하는 형태의 시스템에는 본 알고리즘을 적용하여 프로그램 수행시 차지하는 공간의 낭비를 막을 수 있다.

[참 고 문 헌]

[1] Bjarne Stroustrup, "The C++ Programming Language", Addison-Wesley. Reading, Mass., 1986.
[2] R.Dixon, T. McKee, P. Schweizer, and M. Vaughan. "A fast method dispatcher for compiled languages with multiple inheritance", Proc. OOPSLA, 1989.
[3] Pascal Andre and Jean-Claude Royer. "Optimizing method search with lookup caches and incremental coloring", Proc. OOPSLA, 1992.
[4] Rakesh Agrawal, Linda G. DeMichiel, and Bruce G. Lindsay, "Static Type Checking of Multi-Methods", Proc. OOPSLA, 1991.
[5] Craig Chambers. "Object-Oriented Multi-Methods in Cecil", ECOOP, 1992.
[6] Eric Amiel, Olivier Gruber, and Eric Simon, "Optimizing Multi-Method Dispatch Using Compressed Dispatch Tables", Proc. OOPSLA, 1994.