3A-2

distccを利用した分散コンパイル速度に対する分析

桐村 昌行†
三菱電機（株）　情報技術総合研究所†

1．まえがき

近年、スマートフォン、カーナビをはじめとした組込み機器において、顧客が求める機能増加とコスト削減の対策として、オープンソースを利用するケースが増加している。ただし、そのソースコードサイズは大幅に増加しており、たとえば、Android のソースパッケージサイズは 4GB を超え、年々劇的に増加している。そのため、オープンソースに対する品質試験に要するコストも問題となるが、コンパイル時間の増加もソフトウェアの生産性を悪化させる一つの要因となっている。

なお、コンパイル速度を向上させるために、コンパイル用に高速なサーバマシンを用意することや、コンパイル結果をキャッシュし、高速化するccache[1]技術を利用することが考えられるが、前者は導入コストが課題であり、後者は製品化試験で実施するフルビルドではキャッシュが無効となるため、適用が向いていない。そこで、低コストでコンパイル速度を向上する方法として分散コンパイルに着目した。

そこで、本稿では、コンパイル時間の短縮手段として、オープンソースの分散コンパイラである"distcc"[2]に着目し、distcc によるコンパイル時間と使用台数、および、ネットワーク負荷の関係について説明する。

2．distcc について

distcc とは、Martin Pool 氏によって開発された GPL(GNU General Public License) ライセンスの C/C++向け分散コンパイラで、数 100 万行を超える大量のソースコードのコンパイル速度向上を目的として開発された。現在は Google Inc.が distcc の開発管理をしている。distcc は、コンパイル要求を行うクライアントから、ネットワーク上の他の PC(サーバと呼ぶ)に対してプリプロセス処理済みのソースを分配し、コンパイルさせることでコンパイルを高速化することを特長としている。

また、もう一つの特長として、導入が容易な点が挙げられる。組込み機器開発に適用するためには、それぞれのサーバに同一バージョンのクロスコンパイラを用意する必要があるものの、サーバ-クライアント間でソースやライブラリ、ヘッダファイルを共有する必要がない。また、コンパイル命令は gcc コンパイル命令の前に prefix として"distcc"の呼び出しを付加するだけで利用することが可能である。

The Analysis of the distributed compilation speed using distcc.
†Masayuki Kirimura
Information Technology R&D Center,
Mitsubishi Electric Corp.

	CPU	
	CPU名 / Clock	Core数
CL_1	Intel® Core™i5-2400 / 3.1GHz	4
SV_2	Intel® Core™2Duo E8400 / 3.0GHz	2
SV_3	Intel® Core™2Duo-E4500 / 2.2GHz	2

Cross Compiler : Linaro® Tool chain linaro-gcc 4.6

(I)各PCの構成

ファイル数(個)		ソースコード量(L)	
ヘッダ	ソース	ヘッダ	ソース
4466	3631	482151	1224223

(II)ファイルの構成

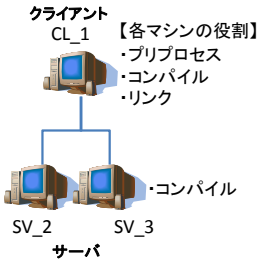

Fig.1 distcc の分析に利用した PC の構成

2.1.distcc の動作概要

distcc の動作について Fig.1 右図で説明する。まず、distcc はクライアントとなる PC(CL_1)で、対象となるソースのプリプロセス処理を実施する。次に、クライアントは事前に設定しておいた分散コンパイルサーバ(SV_2,SV_3)の IP アドレス群を参照し、コンパイル要求とともにプリプロセス済みのファイルを対象サーバに送信する。このとき、サーバがコンパイル要求を受信するためには、distccd デーモンを常駐させておく必要がある。サーバは受信したファイルをコンパイルし、生成したオブジェクトファイルをクライアントに送信する。なお、リンク処理はクライアントで実施する。

また、分散コンパイルを行うサーバ側では、分散コンパイルに対する優先度 (Nice 値)設定を行うことが可能である。そのため、開発者の PC を分散コンパイルサーバとして利用する場合は、その配分を調整することで、開発者の作業に影響を与えることなく、効率良くコンパイルを行うことが可能である。また、distcc ではコンパイル要求に対するタイムアウト時間も設けており、コンパイル要求先のサーバの負荷によって、コンパイル要求先を調整する機能を有している。

なお、分散コンパイルではネットワーク上でコンパイル対象ファイルの送受信を行うため、そのネットワークオーバーヘッドも課題となる。

2.2.pump モード

pump モードとは Google Build Tools Team によって改良された distcc 速度向上アルゴリズムで、2008 年にリリースされた distcc 3.0 より導入されている。pump モードはソースファイルだけでなく、ヘッダファイルを送信し、プリプロセス処理をサーバ側に要求することを特長としており、コンパイル処理負荷の平準化を行うことが可能である。なお、対象となるソース構成にも依存するが、pump モードを使用することにより、distcc 使用時よりもさらにコンパイル速度が向上することが報告されている[3]。

平均コンパイル時間(sec)			CL_1を1とした時の時間比率(換算Core数)		
CL_1	SV_2	SV_3	CL_1	SV_2	SV_3
0.98	**2.34**	**3.45**	**1(4)**	**2.39(1.67)**	**3.52(1.14)**

(III) 各PCの1ファイルあたりの平均コンパイル時間**(マシン単体)**

平均コンパイル時間(sec)				CL_1を1とした時の時間比率			
1台	2台		3台	1台	2台		3台
CL_1	CL_1,SV_2	CL_1,SV_3	CL_1,SV_2,3	CL_1	CL_1,SV_2	CL_1,SV_3	CL_1,SV_2,3
0.98	**0.86**	**0.93**	**0.72**	**1**	**0.88**	**0.95**	**0.73**

(IV) 1ファイルあたりの平均コンパイル時間 **(distcc)**

平均コンパイル時間(sec)				CL_1を1とした時の時間比率			
1台	2台		3台	1台	2台		3台
CL_1	CL_1,SV_2	CL_1,SV_3	CL_1,SV_2,3	CL_1	CL_1,SV_2	CL_1,SV_3	CL_1,SV_2,3
0.98	**0.74**	**0.79**	**0.63**	**1**	**0.76**	**0.81**	**0.64**

(V) 1ファイルあたりの平均コンパイル時間 **(distcc + pump)**

Fig.2 分散コンパイル台数と平均コンパイル時間の関係

	CL_1を1とした時の時間比率(換算Core数)			
	1台	2台		3台
	CL_1 (4)	CL_1,SV_2 (5.67)	CL_1,SV_3 (5.14)	CL_1,SV_2,3 (6.81)
(A)計測結果(distcc+pump)	**1**	**0.76**	**0.81**	**0.64**
(B)換算Core数による期待値	**1**	**0.71**	**0.78**	**0.59**
(C)ネットワーク遅延が占める割合	**0**	**6.58%**	**3.7%**	**7.81%**

Fig.3 ソースコード量とネットワーク負荷の関係

3. コンパイル速度の分析

3.1. 実験環境

distcc を用いた最適な分散コンパイル環境を評価するため、Fig.1(I)に示す分散コンパイルマシンを用意し、ARM®向けクロスコンパイラとして Linaro®[4]が提供する linaro-gcc4.6 を用意した。なお、それぞれの PC は 100Base-T の Ethernet でスイッチングハブを介してローカル接続している。

また、distcc の設定として、クライアントを CL_1、サーバ SV_2、SV_3 として、すべてのマシンでコンパイルを行うように設定した。そのため、クライアント CL_1 はコンパイルの他に、プリコンパイル、ソース送信処理をすべて行う。また、サーバ側では要求したコンパイル処理を最大限行うように、優先度設定を行った。

なお、今回はソースファイルとして Fig.1(II)に示す規模のプロジェクトを用意し、コンパイル時間は 3 回の計測結果の平均値を求めることとした。

3.2. 計測結果と考察

Fig.2(III)は 3.1 項の実験環境で CL_1、SV_2、3 で計測した 1 ソースファイルあたりの平均コンパイル時間と CL_1 を 1 とした時のコンパイル時間比率である。このとき、L2 キャッシュや HDD のアクセス速度、RAM のサイズの影響は無視する。このとき、単純に CPU の周波数と Core 数に反比例して平均コンパイル時間が短縮できると仮定した場合、CL_1 の処理時間を基準にすると SV_2 は 1.67Core、SV_3 は 1.14Core 相当になると仮定する。

例えば、CL_1 と SV_2 で分散コンパイルを行う場合、(4+1.67)Core マシンでコンパイルするのと同義になる。

また、Fig.2(IV)は distcc を利用して、CL_1 をクライアント、SV_2 および SV_3 をサーバとして分散コンパイルを行った結果である。このとき、3 台利用することで最大約 3 割のコンパイル時間を短縮していることが分かる。Fig.2(V)はさらに pump モードで distcc を利用した結果である。pump モードによって最大約 4 割のコンパイル時間を短縮していることが分かる

ただし、換算 Core 数でのコンパイル時間よりも若干コンパイル時間が遅くなっている。このとき、プリプロセス処理にかかる時間はどの組み合わせにおいても共通であるため、コンパイルデータをネットワーク上で送受信することによるオーバーヘッドが影響しているものと考えられる。

3.3. ネットワークによるオーバーヘッドの影響

Fig.2(V)の pump モードでのコンパイル時間計測結果(Fig.3(A))において、CL_1 でのコンパイル時間を 1 とした時のコンパイル時間比率を元にして、Fig.2(III)で求めた換算 Core 数でコンパイル時間の期待値を求めると Fig.3(B)のようになる。

このとき、(A)(B)の差分をすべてネットワークオーバーヘッドによるものと考えた場合、Fig.3(C)のようになる。つまり、2 台で分散コンパイルした場合は最大 6.58%、3 台の場合は 7.81%となり、若干ではあるもののネットワークオーバーヘッドがコンパイル時間に占める割合が増えていることが分かる。

このことから、2Core のマシンが 1 台増える毎に 20%程度のコンパイル速度向上が望めるものの、1.23%程度のネットワークオーバーヘッドの影響を受けることから、20/1.23=16 台程度になるとコンパイル速度の向上は見込めないことが想定できる。

ただし、今回の実験では 3 台という少ない台数での想定であるため、正確な台数を求めるためにはさらに台数を増やしてフィールド実験を行う必要がある。

5. まとめ

本稿では、大規模な組込み機器を構成するソフトウェアの開発生産性を高める手段の一つとして分散コンパイルに着目し、分散コンパイラ"distcc"を利用し、異なるスペックの PC3 台によるコンパイル速度の計測を実施した。そして、コンパイル速度と台数、ネットワークオーバーヘッドの関係を考察し、その傾向について分析した。今後は、ヘッダ・ソースファイル数やライブラリ数と分散コンパイル速度の関係の分析や、ccache と distcc の併用によるコンパイル速度分析を行うことで、開発生産性の向上に役立てていく予定である。

参考文献

1.ccache, http://ccache.samba.org/
2.Martin P.,"distcc, a fast free distributed compiler", Linux Conf. 2004,(2004).
3.Nils K.," distcc's pump mode: A New Design for Distributed C/C++ Compilation",Google Open Source Blog,(2008).
4.linaro, http://www.linaro.org/