CLUSTER DISTRIBUTED EXTERNAL COMPILER

Tomáš BEČICA, Master Degree Programme (5)
Dept. of Computer Systems, FIT, BUT
E-mail: xbecic01@stud.fit.vutbr.cz

Supervised by: Dr. Alexander Meduna

ABSTRACT

Parallel, that's the most frequently spoken world in today's computer sciences. Parallelism exists in many levels of detail, from instructions to processes. In this project I attempted to analyze building up the applications, which distribute the compilation of C and C++ sources between several computation nodes with needless to have installed identical libraries and headfiles in this nodes. The same version of compiler is only recommended. It is even possible to use different OS, under condition of use cross-compiler. The aim is to make the compilation much faster.

1 ÚVOD

Za ne zrovna jasným názvem se ukrývá snaha o vytvoření aplikace, jenž by sloužila jako nástroj k urychlení kompilace aplikací zapsaných v C a C++ programovacích jazycích. Tento projekt se zabývá výstavbou nástroje provádějícího distribuci překladového úkolu na cluster kompilujících uzlů. Zároveň porovnává jeho vlastnosti a implementaci s obdobným *open source* projektem – *distcc*.

Použití externího překladače umožňuje uživateli použít libovolný jiný kompatibilní překladač, aktualizovat jej, vyměnit za jiný atd. Přičemž neplatí omezení, že ve všech uzlech využívaného clusteru by musela být táž kopie kompilátoru, knihoven nebo hlavičkových souborů. Dokonce nevadí ani rozdílnost architektury či operačního systému, ovšem za předpokladu překladače třídy cross-compiler[1].

2 VYUŽITÍ

Možná se někdo pozastaví nad tím, proč se snažit urychlit překlad, když v praxi se uživatel, a i odborný uživatel, s déle trvajícím překladem prakticky nesetká, nebo jen zřídka. Proto si teď zkusme představit situace, kdy nám tento nástroj může pomoci.

[1] Překladač, který pracuje na jisté platformě(architektuře,operačním systému) a produkuje kód pro jinou platformu.

- Skupina vývojářů – pracuje se ve skupině několika lidí, přičemž každý má svůj stroj s určitým výkonem. Vytvářené projekty se časem rozrostou a po jisté době překlad zabere i několik minut. To samozřejmě ruší v práci a ani produktivitě to mnoho neprospívá.
- Zabudované a malé počítače – pracujete na software pro malé stroje, např. PDA a kvůli optimalizaci provádíte vývoj na přibližně stejně výkonném stroji. Tudíž překlad většího množství kódu, zvlášť se zapnutím všech možných optimalizací, je žalostně pomalý.

3 VÝPOČETNÍ CLUSTER

Pro zvýšení výpočetního výkonu existuje mnoho superpočítačů od různých společností, vystavěných nad technologií, kde se víceprocesorová výpočetní jednotka tváří jako jeden stroj, a tak netřeba běžící aplikaci pro větší výkon jakkoliv modifikovat. Má to však jeden háček, a tím je výše pořizovacích nákladů. Zde přicházejí na řadu právě clustery. Ty takovouto pokročilou technologií nedisponují, jsou proto mnohem levnější, komponentami jsou běžně dostupné univerzální počítače a tudíž je schopnost rozkladu zátěže řešena například nadstavbou systému pomocí technologie MPI[2].

4 PRINCIPY IMPLEMENTACE

Nástroj *cdec* představuje v podstatě kompilační driver. Vlastní překlad ze souboru soubor.c do soubor.o lze rozdělit do dvou základních kroků. Preprocessing (ze soubor.c do soubor.i) a kompilace (ze soubor.i do soubor.o).

Část preprocesingu je relativně rychlá, ikdyž může ve výjimečných případech zabírat procentuelně větší část času než kompilace. Záleží na zdrojovém kódu a zatížení lokálního systému. Během preprocesingu jsou čteny všechny includované soubory a je vytvářen jeden velký výsledný soubor s příponou .i. Krok preprocessingu je přímo závislý na vstupním souboru (*.c a *.h), parametrech přišedších z příkazové řádky a v neposlední řadě také na verzi použitého kompilátoru. Právě rozdílnost verzí kompilátoru může vést k nečekaným událostem a je významným kritickým bodem.

V současné době dochází k plnému provedení preprocesingu, takže výsledný soubor může být celkem dost velký pro přenos po síti. Berličkou, která by mohla odlehčit tomuto problému je implementace komprimace do transportní vrstvy. I tak bych doporučil používání spíše na lokálních sítích než přes internet. Není to sice vyloučeno, takové omezení program nezná, přínos by však byl minimální.

Při opětovné kompilaci není nutné rekompilovat objektové soubory, ale postačí je vyhledat v *cache*. Nástroj *cdec* má implementován potřebný mechanizmus.

Druhý krok – kompilace souborů předzpracovaných preprocessorem do objektových souborů je distribuována na cluster o *n* jednotkách. Každá jednotka dostane předzpracovaný soubor z kroku 1 a provede jeho překlad. Výsledek poté vrátí zpět zadávajícímu uzlu.

Celý systém je postaven na úkolování serveru clientem (viz bod 5) a následném přerozdělení zadaných úkolů mezi komunikujícími servery. Na základě informací o sobě i ostatních se server rozhodne, kde se bude překlad provádět. Předpokládá se implementace

[2] Message Passing Interface - komunikační protokol, de facto standard pro komunikaci mezi uzly provádějícími paralelní program s distribuovanou pamětí. MPI představuje knihovnou subrutin, jež mohou být volány z Fortranu, C a C++ programů.

dvou algoritmů tohoto rozdělováni. Jednoduchý *RoundRobin* a *FastestFirst*. Aby server mohl učinit takováto rozhodnutí, musí každý z účastníků dodat informaci o svém aktuálním výkonu. To se provede tak, že každý daemon při svém startu si provede malý test své výkonnosti a za pomocí indexu zatížení procesoru poté podá zprávu jaká je jeho aktuální výpočetní síla.

Aby server věděl, koho má v týmu, je potřeba mu to nějak sdělit. Ve hře bylo několik možností. Použití multicastových adres jsem zamítnul kvůli nárokům na speciální konfiguraci technických i softwarových prostředků. Broadcast vypadl pro změnu z důvodu nepotvrzovaného protokolu UDP. Nakonec jsem se přiklonil k variantě přihlašování se. Při spuštění daemona řekneme parametrem ke kterému serveru se má přidat a pomoci s kompilací. Lze tak vytvářet různé skupiny spolupracujících daemonů.

5 ALGORITMUS ÚKOLOVÁNÍ SYSTÉMU SERVERŮ POMOCÍ KLIENTA

- načtení proměnných z prostředí (server, port, logfile …), kontrola zadaných parametrů
- pokus o spojení se zadaným serverem
    - nedojde-li ke spojení, vytvoří se nový proces, jenž spustí úkol lokálně
    - dojde-li ke spojení:
        - kontrola, zda je distribuce překladu možná
        - připraví se commandline parametry pro dvouprůchodové zpracování
        - vytvoření dočasných souborů pro mezivýsledky
        - zjištění aktuální absolutní cesty k překladači
        - vytvoření mezisouboru *.i* lokálním překladačem
            - nepodařilo se, vytvoří se nový proces, jenž spustí úkol lokálně
            - podařilo se:
                - zaslání serveru hodnoty hash, jméno a verzi překladače, jména vstupních a výstupních souborů
                - zaslání předzpracovaného souboru .i
                - čeká se na odezvu serveru
    - přeložené soubory, ty jsou uloženy do tempu
    - jednotlivé komponenty jsou slinkovány dohromady

LITERATURA

[1] Wikipedia, on-line encyklopedie (http://www.wikipedia.org)