Különböző szintű absztrakcióval megvalósított JIT technológiák összehasonlítása

Készítette: Kelemen Balázs (KEBNAAT.SZE). Szak, évf.: műszaki informatika, 5.

Napjaink informatikai fejlődésének egyik alapköve a web. A fejlődés iránya az, hogy egyre több szolgáltatást valósítanak meg web-es keretek között, illetve hogy egyre szélesebb az a hardver-paletta, amely alkalmas a web használatára. Ezek az igények azt eredményezik, hogy a web egyre inkább a dinamikus technológiák felé tolódik el. A legnépszerűbb ilyen technológia pedig a javascript, így felmerül az igény, hogy a javascript végrehajtó motorok legyenek minél hatékonyabbak. Ma már ott tartunk, hogy a legfejlettebb interpreterek JIT (just in time compiling) technológiát használnak a teljesítmény fokozására. A JIT annyit tesz, hogy az interpreter futás közben generál gépi kódot, amit aztán közvetlenül hajt végre a CPU. Tekintve, hogy a javascript egy igen összetett és dinamikus nyelv, a JIT-telt végrehajtás igen bonyolult feladat. Kutatásom témája egyrészt implementálni egy ilyen rendszert a WebKit böngészőmotor számára, másrészt összehasonlítani azt más megvalósításokkal, elsősorban a WebKit jelenlegi megvalósításával.

A javascript végrehajtó környezetek kezdetben egyszintűek voltak. Az engine elemezte a szkriptet, felétített belőle egy absztrakt szintakszisfát (AST), majd a fa bejárása közben az egyes node-okhoz a megfelelő szemantikus akciót hajtotta végre. Ennek a koncepciónak megvannak a korlátai. Egyrészt a fa bejárása rengeteg memóriaműveletet jelent, és erősen rontja a lokalitást (mivel a fa elemei a memóriában nem egybefüggően helyezkednek el). Emellett a különböző optimalizációs lehetőségek számára a szintakszis fa nem hatékony reprezentáció, mert nehéz az egyszerre több szintaktikus elemre illeszthető optimalizációs minták felismerése, illetve alkalmazása. Ezt ismerte fel több projekt (többek között a WebKit), amikor úgy döntöttek, áttérnek egy olyan virtuális gép alapú végrehajtásra, ami egy az AST-ből generált, szekvenciálisan végrehajtható bájtkódon operál. Ez a koncepció többszörösére növelte a teljesítményt. A virtuális gép alapú végrehajtásnak a legsúlyosabb korlátja a dispatch, azaz a következő bájtkódot végrehajtó kódrészre való ugrás. Az interpreter magját egy olyan switch szerkezet adja, amelyben az aktuális bájtkód alapján határozódik meg hogy melyik ág fut le. Ez azt jelenti, hogy minden egyes bájtkód-utasításhoz ki kell választani a megfelelő ágat, ami igen költséges. Az ideális valójában az lenne, ha egy a végrehajtott gépi utasítások egybefüggően helyezkednének el. Pontosan ez az, ahol a JIT képbe kerül. Ha a bájtkódból gépi utasítások egybefüggő sorozatát generálom, majd ezt hajtatom a CPU-val végre, akkor eltűnik a dispatch költsége.

A JIT megvalósítás sarokpontjai hogy mit, mikor és hogyan JIT-teljünk. Az egyik lehetőség az, hogy a végrehajtás tisztán JIT-telt kóddal történik, a másik pedig az, ha a hagyományos interpretert megtartva csak a teljesítmény szempontjából kritikus részek kerülnek JIT-telésre. Ez utóbbi megoldás egy jó kompromisszum lehet memóriahasználat és teljesítmény között. A reprezentáció szempontjából a feljlődés lépéseinek megfelelően adódik a háromszintű reprezentáció lehetősége, azaz a végrehajtó motor először AST-t épít, ebből bájtkódot generál, amiből gépi kódot hoz létre. Ha a motor tisztán JIT alapú, akkor az AST-ből közvetlenül is előállítható a JIT-telt kód. A gépi kód generálásához olyan modulra van szükség, amely egyrészt egy jól elkülönülő modul a virtuális gépen belül, másrészt könnyen portolható különböző architektúrákra.

A saját kutatásaim során elsősorban az utolsó kérdéskörre, azaz a gépi kód generálásának a módjára kívánok koncentrálni. Két ilyen modult fogok megvizsgálni. Az egyik a Mozilla és Tamarin projektek által fejlesztett nanojit, a másik pedig a WebKit MacroAssembler modulja. Két társammal egyetemben (Ágoston Róbert és Szirbucz Tamás) a WebKit-ben újraimplementáljuk a gépi kód generálását nanojit alapokon. Ezt össze fogom hasonlítani futásidő és memóriahasználat szempontjából a MacroAssembler alapú implementációval, X86 és ARM architektúrákon. Továbbá egy átfogó elemzést fogok készíteni a különböző JIT-et használó javascript/ecmascript motorokról (Tracemonkey, Tamarin, WebKitJavaScriptCore, V8), a további lehetséges fejlődési irányokat is elemezve.

Kelt.: Szeged, 2009. november 30. Kelemen Balázs